计算机
组成

第六章　计算机的运算方法

Contents

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

6.5 算术逻辑单元

大 纲

(一)数制与编码

1. 进位计算制及其数据之间的相互转换
2. 定点数的编码表示

(二)运算方法和运算电路

1. 基本运算部件

   加法器，算术逻辑部件（ALU）

2. 加/减法运算

   补码加减法运算器，

   标志位的生成

3. 乘除法运算的基本原理，

   乘法电路和除法电路的基本结构

(三)整数的表示和运算

1. 无符号整数的表示和运算
2. 带符号整数的表示和运算

(四) 浮点数的表示和运算

1. 浮点数的表示

   IEEE754标准

2. 浮点数的加/减运算

数制与编码

1. 进位计算制及其相 互转换

1). 基r数制：用r个基本符号（如0,1,2,...,r-1）表示数值N。r为 基数

$$N = D_{m-1}D_{m-2}\ \dots\ D_1\ D_0\ D_{-1}\ D_{-2}\ \dots\ D_{-k}$$

其中， D_i（$-k \le i \le m-1$）为基本符号，小数点位置隐含在D_0 与D_{-1} 之间。

2). 有权基r数制：每个D_i的单位都赋以固定权值w_i，w_i为D_i的权。

如果该数制是逢r进位，则有

$$N = \sum_{i=-k}^{m-1} D_i \times r^i$$

其中r^i为位权，称该数制为r进位数制，简称r进制，

3). 进制转换

(1) R进制数 => 十进制数

按“权”展开 (a power of R)

例1: $(10101.01)_2=1\times 2^4+1\times 2^2+1\times 2^0+1\times 2^{-2}=(21.25)_{10}$

例2: $(307.6)_8=3\times 8^2+7\times 8^0+6\times 8^{-1}=(199.75)_{10}$

例1: $(3A.1)_{16}=3\times 16^1+10\times 16^0+1\times 16^{-1}=(58.0625)_{10}$

(2)十进制数 => R进制数

整数部分和小数部分分别转换

① 整数(integral part)----“除基取余，上右下左”

② 小数(fractional part)----“乘基取整，上左下右”

二、十进制转换

例1:$(835.6785)_{10}=(1101000011.1011)_2$

整数----“除基取余，上右下左”

		余数	
2	835	1	低位
2	417	1	
2	208	0	
2	104	0	
2	52	0	
2	26	0	
2	13	1	
2	6	0	
2	3	1	
2	1	1	
	0		高位

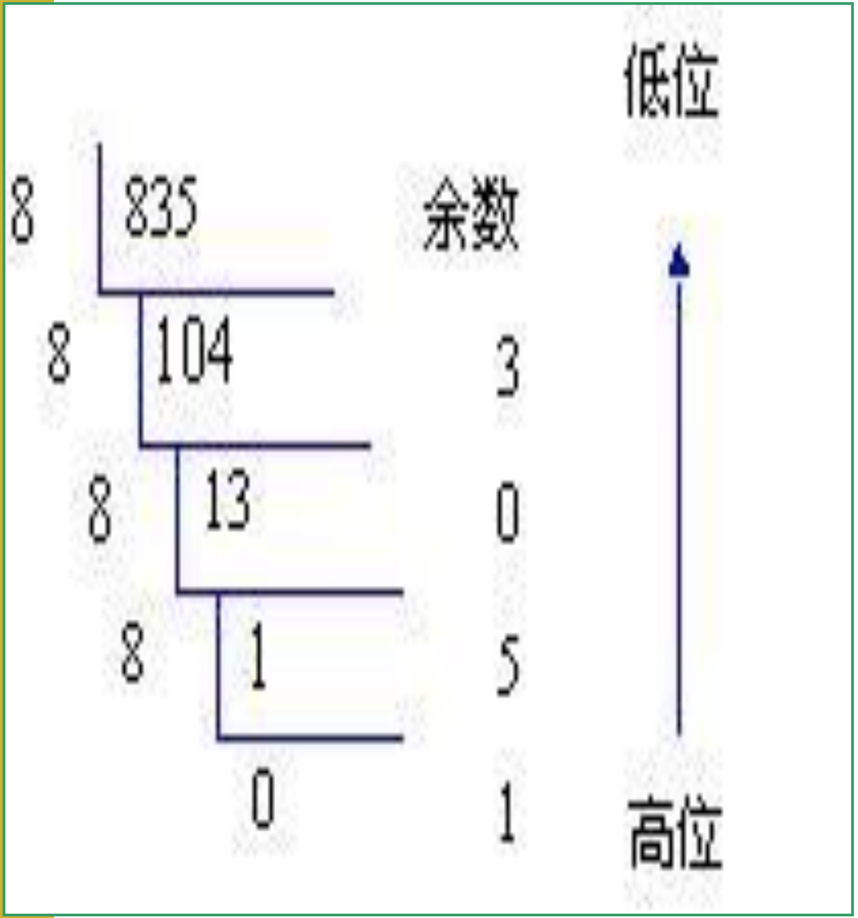
小数----“乘基取整，上左下右”

0.6875×2=1.375	整数部分=1	(高位)
0.375×2=0.75	整数部分=0	↓
0.75×2=1.5	整数部分=1	↓
0.5×2=1.0	整数部分=1	(低位)

例2:$(835.63)_{10}=(1503.50243\ldots)_8$

整数----“除基取余，上右下左”小数----“乘基取整，上左下右”

有可能乘积的小数部分总得不到0 ，此时得到一个近似值。



$0.63\times8=5.04$ 整数部分=5 (高位)

$0.04\times8=0.32$ 整数部分=0

$0.32\times8=2.56$ 整数部分=2

$0.56\times8=4.48$ 整数部分=4

$0.48\times8=3.84$ 整数部分=3 (低位)

(3) 二/八/十六进制数的相互转换

① 八进制数转换成二进制数

$(13.724)_8=(\ 001\ 011\ .\ 111\ \ 010\ \ 100\ \ )_2=(1011.1110101)_2$

② 十六进制数转换成二进制数

$(2B.5E)_{16} = (00101011\ .\ 01011110)_2 = (101011.0101111)_2$

③ 二进制数转换成八进制数

$(0.10101)_2 = (\ \ 000\ .\ 101\ \ 010\ \ )_2 = (\ 0.52)_8$

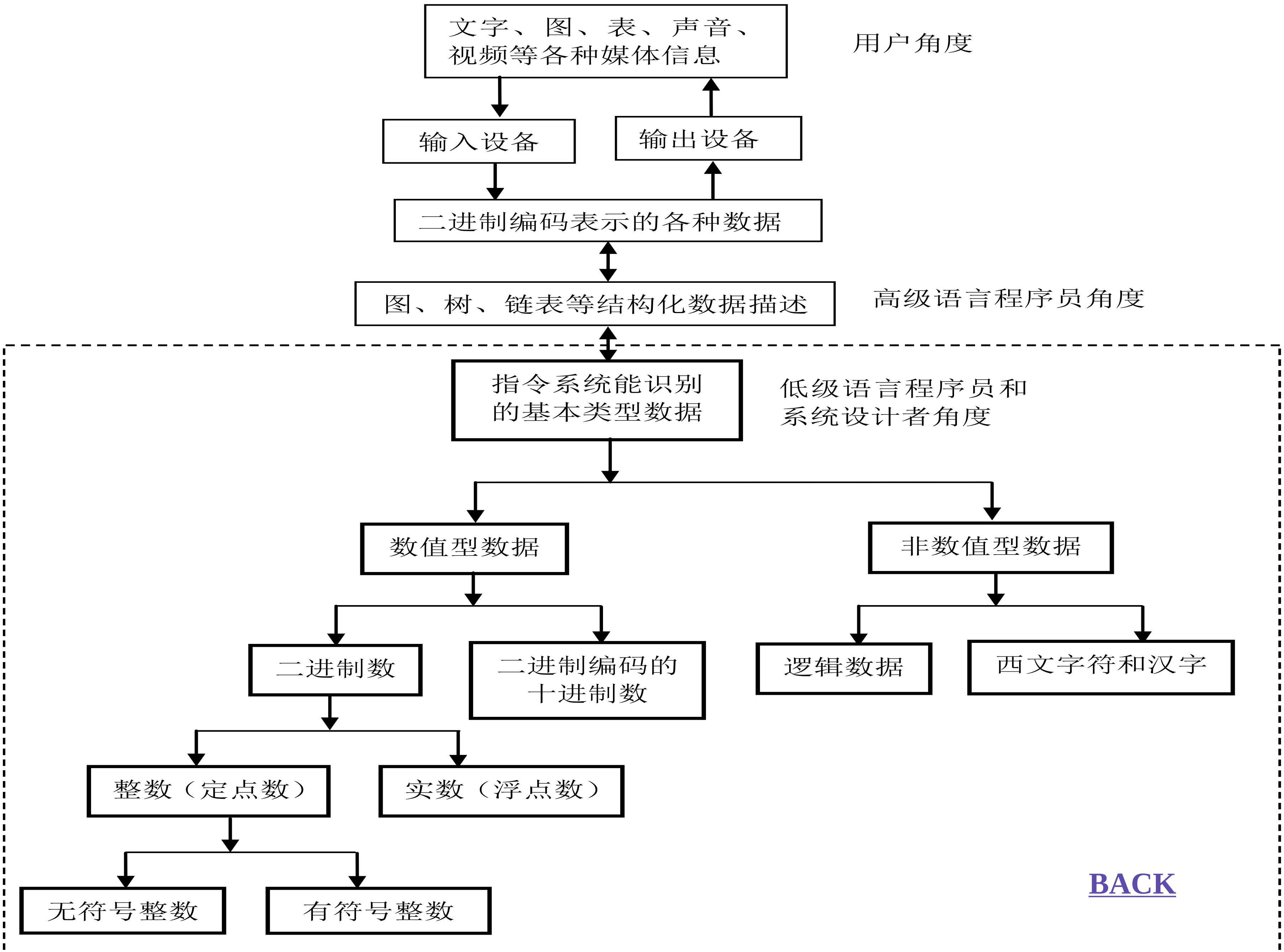
④ 二进制数转换成十六进制数

$(11001.11)_2 = (\ \ 0001\ \ 1001\ .\ 1100\ \ )_2 = (\ 19.C\ )_{16}$

信息的二进制编码

- 计算机的外部信息与内部机器级数据
- 机器级数据分两大类：
  - 数值数据：无符号整数、带符号整数、浮点数（实数）、十进制数
  - 非数值数据：逻辑数（包括位串）、西文字符和汉字
- 计算机内部所有信息都用二进制（即：0和1）进行编码
- 用二进制编码的原因：
  - 制造二个稳定态的物理器件容易
  - 二进制编码、计数、运算规则简单
  - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算

数值数据的表示



文字、图、表、声音、视频等各种媒体信息
用户角度
输入设备
输出设备
二进制编码表示的各种数据
图、树、链表等结构化数据描述
高级语言程序员角度
指令系统能识别的基本类型数据
低级语言程序员和系统设计者角度
数值型数据
非数值型数据
二进制数
二进制编码的十进制数
逻辑数据
西文字符和汉字
整数（定点数）
实数（浮点数）
无符号整数
有符号整数
BACK

数值数据的表示

- 数值数据表示的三要素
  - 进位计数制
  - 定、浮点表示
  - 如何用二进制编码

  即：要确定一个数值数据的值必须先确定这三个要素。

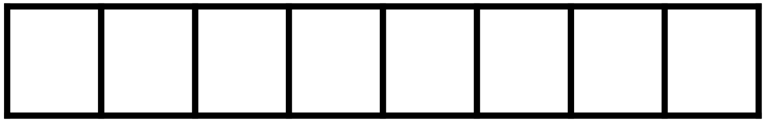
  例如，机器数 01011001的值是多少？ 答案是：不知道！
- 进位计数制
  - 十进制、二进制、十六进制、八进制数及其相互转换
- 定/浮点表示（解决小数点问题）
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- 定点数的编码（解决正负号问题）
  - 原码、补码、反码、移码 （反码很少用）

6.1 无符号数和有符号数

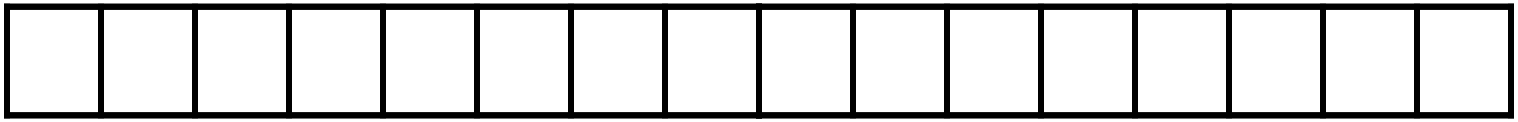
一、无符号数

寄存器的位数（机器字长）

反映无符号数的表示范围



8 位　　　　　　0 ~ 255



16 位　　　　　　0 ~ 65535

二、有符号数 6.1

1. 机器数与真值

真值	机器数
带符号的数	符号数字化的数
+ 0.1011	0 ‖ 1011 小数点的位置
– 0.1011	1 ‖ 1011 小数点的位置
+ 1100	0 ‖ 1100 小数点的位置
– 1100	1 ‖ 1100 小数点的位置

2. 原码表示法

(1) 定义

整数

$$[x]_{原} = \begin{cases} 0, \ x & 0 \le x < 2^n \\ 2^n - x & -2^n < x \le 0 \end{cases}$$

x 为真值　　n 为整数的位数

如　$x = +1110$　　$[x]_{原} = 0，1110$

用 逗号 将符号位和数值位隔开

$x = -1110$　　$[x]_{原} = 2^4 + 1110 = 1，1110$

带符号的绝对值表示

小数

$$[x]_{原} = \begin{cases} x & 1 > x \geq 0 \\ 1 - x & 0 \geq x > -1 \end{cases}$$

x 为真值

如

$x = +0.1101$ $\quad [x]_{原} = 0\,.\,1101$

用 小数点 将符号位和数值部分隔开

$x = -0.1101$ $\quad [x]_{原} = 1-(-0.1101) = 1\,.\,1101$

$x = +0.1000000$ $\quad [x]_{原} = 0\,.\,1000000$

用 小数点 将符号位和数值部分隔开

$x = -0.1000000$ $\quad [x]_{原} = 1-(-0.1000000) = 1\,.\,1000000$

(2) 举例

例 6.1　已知 $[x]_{原} = 1.\underline{0011}$　求 x　$-$ 0.0011

解：由定义得

$$x = 1 - [x]_{原} = 1 - 1.0011 = -0.0011$$

例 6.2　已知 $[x]_{原} = 1,\underline{1100}$　求 x　$-$1100

解：由定义得

$$x = 2^4 - [x]_{原} = 10000 - 1,1100 = -1100$$

(2) 举例

例 6.1　已知 $[x]_{原} = 1.0011$　求 x　-0.0011

解：由定义得

$$x = 1-[x]_{原} = 1-1.0011 = -0.0011$$

例 6.2　已知 $[x]_{原} = 1,1100$　求 x　-1100

解：由定义得

$$x = 2^4-[x]_{原} = 10000-1,1100 = -1100$$

原码的特点：简单、直观　6.1

但是用原码作加法时，会出现如下问题：

要求	数1	数2	实际操作	结果符号
加法	正	正	加	正
加法	正	负	减	可正可负
加法	负	正	减	可正可负
加法	负	负	加	负

能否 只作加法？

找到一个与负数等价的正数 来代替这个负数

就可使 减 ⟶ 加

3. 补码表示法

6.1

(1) 补的概念

- 时钟　　逆时针　　顺时针

$$\begin{array}{r} 6 \\ -\ 3 \\ \hline 3 \end{array} \qquad \begin{array}{r} 6 \\ +\ 9 \\ \hline 15 \\ -\ 12 \\ \hline 3 \end{array}$$

可见 – 3 可用 + 9 代替　减法 ⟶ 加法

称 + 9 是 – 3 以 12 为模的补数

时钟以12为模

记作 $-3 \equiv +9 \pmod{12}$

同理 $-4 \equiv +8 \pmod{12}$

$-5 \equiv +7 \pmod{12}$

结论

- 一个负数加上"**模**"即得该负数的补数
- 两个互为补数的数 它们绝对值之和即为 **模** 数

  - 计数器（模 16） $1011 \longrightarrow 0000$?

11

$$\begin{array}{r} 1011 \\ -\ 1011 \\ \hline 0000 \end{array} \qquad\qquad \begin{array}{r} 1011 \\ +\ 0101 \\ \hline \boxed{1}0000 \end{array}$$

自然去掉

可见 -1011 可用 $+0101$ 代替

记作 $-1011 \equiv +0101 \quad (\mathrm{mod}\ 2^4)$

同理 $-011 \equiv +101 \quad (\mathrm{mod}\ 2^3)$

$-0.1001 \equiv +1.0111 \quad (\mathrm{mod}\ 2)$

(2) 正数的补数即为其本身 6.1

两个互为补数的数 $-1011 \equiv +0101$ $(\mathrm{mod}\,2^4)$

分别加上模 $+10000$ $+10000$

结果仍互为补数 $+0101 \equiv +\boxed{1}0101$ 丢掉

$\therefore +0101 \equiv +0101 \quad (\mathrm{mod}\,2^4)$

可见 $+0101 \rightarrow +0101$

? $\rightarrow -1011$

? $\boxed{0}$,0101 $\rightarrow +0101$

? $\boxed{1}$,0101 $\rightarrow -1011$

$2^{4+1} - 1011 = 100000$ $(\mathrm{mod}\,2^{4+1})$

$$\begin{array}{r} 100000 \\ -1011 \\ \hline 1,0101 \end{array}$$

用 **逗号** 将符号位和数值位隔开

(3) 补码定义

整数

$$[x]_{补} = \begin{cases} 0,\ x & 2^n > x \geq 0 \\ 2^{n+1} + x & 0 > x \geq -2^n \pmod{2^{n+1}} \end{cases}$$

x 为真值　　　　n 为整数的位数

如　$x = +1010$　　　　$x = -1011000$

$[x]_{补} = 0,1010$

用 **逗号** 将符号位和数值位隔开

$$\begin{aligned} [x]_{补} &= 2^{7+1} + (-1011000) \\ &= 100000000 \\ &\ \ -1011000 \\ \hline &\ \ 1,0101000 \end{aligned}$$

小数

$$[x]_{补} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \end{cases} \pmod 2$$

x 为真值

如 $x = +0.1110$ $x = -0.1100000$

$[x]_{补} = 0.1110$

$[x]_{补} = 2+(-0.1100000)$

$= 10.0000000$

$- 0.1100000$

1.0100000

用 小数点 将符号位

和数值位隔开

补码说明

- 补码最高一位是符号位，符号 0 正 1 负
- 补码表示为：2×符号位 + 数的真值
- 零的补码只有一个，故补码能表示 -1
- 补码能很好地用于加减运算，运算时，符号位与数值位一样参加运算。

求特殊数的补码

假定机器数有n位

① $[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0）　（mod 2^n）

② $[-1]_{补} = 2^n - 0\ldots01 = 11\ldots1$（n个1）　（mod 2^n）　整数补码

③ $[-1.0]_{补} = 2 - 1.0 = 1.00\ldots0$（n-1个0）　（mod 2）　小数补码

④ $[+0]_{补} = [-0]_{补} = 00\ldots0$（n个0）

特殊数的补码（续）

- 当补码的位数为n位时，其模为2^n，所以

$[-2^{n-1}]_{补}= 2^n - 2^{n-1} = 10\ldots0$（n-1个0）（mod 2^n）

- 当补码的位数为n+1位时，其模为2^{n+1}，所以

$[-2^{n-1}]_{补}= 2^{n+1} - 2^{n-1} = 2^n + 2^{n-1}=1\ 10\ldots0$（n-1个0）(mod 2^{n+1})

这说明同一个真值在不同位数的补码表示中，对应的机器数不同，因此给定编码表示时，一定要明确编码的位数，在机器内部编码的位数就是机器中运算部件的位数。

(4) 求补码的快捷方式 6.1

设 $x = -1010$ 时

则 $[x]_{补} = 2^{4+1} - 1010 \quad = 11111 + 1 - 1010$

$$
\begin{array}{rr}
= 100000 & \qquad = 11111 + 1 \\
-\ 1010 & -\ 1010 \\
\hline
= 1,0110 & \boxed{10101} + 1 \\
 & = 1,0110
\end{array}
$$

又 $[x]_{原} = \boxed{1,1010}$

当真值为 负 时，补码 可用 原码除符号位外每位取反，末位加 1 求得

(5) 举例

6.1

例 6.5　已知 $[x]_{补} = 0.0001$

求 x

解：由定义得　$x = +0.0001$

例 6.6　已知 $[x]_{补} = 1.0001$

求 x

解：由定义得

$x = [x]_{补} - 2$

$= 1.0001 - 10.0000$

$= -0.1111$

$[x]_{补} \xrightarrow{?} [x]_{原}$

$[x]_{原} = 1.1111$

$\therefore x = -0.1111$

例 6.7　已知 $[x]_{补} = 1,1110$

求 x

解：　由定义得

$x = [x]_{补} - 2^{4+1}$

$= 1,1110 - 100000$

$= -0010$

$[x]_{补} \xrightarrow{?} [x]_{原}$

$[x]_{原} = 1,0010$

$\therefore x = -0010$

当真值为 负 时，原码 可用 补码除符号位外

每位取反，末位加 1 求得

练习 求下列真值的补码 6.1

真值		$[x]_{补}$	$[x]_{原}$
$x = +70 = 1000110$		$0,1000110$	$0,1000110$
$x = -70 = -1000110$		$1,0111010$	$1,1000110$
$x = 0.1110$		0.1110	0.1110
$x = -0.1110$		1.0010	1.1110
$x = \boxed{0.0000}$	$[+0]_{补} = [-0]_{补}$	$\boxed{0.0000}$	0.0000
$x = \boxed{-0.0000}$		$\boxed{0.0000}$	1.0000
$x = -1.0000$		1.0000	不能表示

由小数补码定义 $$[x]_{补} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod{2} \end{cases}$$

$$[-1]_{补} = 2 + x = 10.0000 - 1.0000 = 1.0000$$

4. 反码表示法

(1) 定义

整数

$$[x]_{反} = \begin{cases} 0, \quad x & 2^n > x \geq 0 \\ (2^{n+1} - 1) + x & 0 \geq x > -2^n \ (\mathrm{mod}\ 2^{n+1} - 1) \end{cases}$$

x 为真值　　　　n 为整数的位数

如　　$x = +1101$　　　　$x = -1101$

$[x]_{反} = 0,1101$　　　　$[x]_{反} = (2^{4+1} - 1) - 1101$

$= 11111 - 1101$

$= 1,0010$

用 逗号 将符号位

和数值位隔开

小数 6.1

$$[x]_{反} = \begin{cases} x & 1 > x \geq 0 \\ (2 - 2^{-n}) + x & 0 \geq x > -1 \ (\mathrm{mod}\ 2 - 2^{-n}) \end{cases}$$

x 为真值

如

$x = +0.1101$

$[x]_{反} = 0.1101$

用 小数点 将符号位

和数值位隔开

$x = -0.1010$

$[x]_{反} = (2 - 2^{-4}) - 0.1010$

$= 1.1111 - 0.1010$

$= 1.0101$

例6.8　已知 $[x]_{反} = 0,1110$　求 x

解：　由定义得　$x = +1110$

例6.9　已知 $[x]_{反} = 1,1110$　求 x

解：　由定义得　$x = [x]_{反} - (2^{4+1} - 1)$

$= 1,1110 - 11111$

$= -0001$

例 6.10　求 0 的反码

解：　设 $x = +0.0000$　　$[+0.0000]_{反} = 0.0000$

$x = -0.0000$　　$[-0.0000]_{反} = 1.1111$

同理，对于整数　$[+0]_{反} = 0,0000$　$[-0]_{反} = 1,1111$

$\therefore\ [+0]_{反} \neq [-0]_{反}$

三种机器数的小结

6.1

- 最高位为符号位，书写上用“,”（整数）或“.”（小数）将数值部分和符号位隔开
- 对于正数，原码 = 补码 = 反码
- 对于负数，符号位为 1，其 数值部分
  原码除符号位外每位取反末位加 1 → 补码
  原码除符号位外每位取反 → 反码

例6.11 设机器数字长为 8 位（其中一位为符号位）

对于整数，当其分别代表无符号数、原码、补码和反码时，对应的真值范围各为多少？

二进制代码	无符号数对应的真值	原码对应的真值	补码对应的真值	反码对应的真值
00000000	0	+0	±0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
01111111	127	+127	+127	+127
10000000	128	-0	-128	-127
10000001	129	-1	-127	-126
⋮	⋮	⋮	⋮	⋮
11111101	253	-125	-3	-2
11111110	254	-126	-2	-1
11111111	255	-127	-1	-0

例6.12　已知 $[y]_补$　求 $[-y]_补$ 6.1

解：设 $[y]_补 = y_0.\, y_1 y_2 \cdots y_n$

<I>　$\boxed{[y]_补 = 0.\, y_1 y_2 \cdots y_n}$

$[y]_补$连同符号位在内，　每位取反，　末位加 1

即得$[-y]_补$

$$\boxed{[-y]_补 = 1.\overline{y_1}\,\overline{y_2} \cdots \overline{y_n} + 2^{-n}}$$

<II>　$\boxed{[y]_补 = 1.\, y_1 y_2 \cdots y_n}$

$[y]_补$连同符号位在内，　每位取反，　末位加 1

即得$[-y]_补$

$$\boxed{[-y]_补 = 0.\overline{y_1}\,\overline{y_2} \cdots \overline{y_n} + 2^{-n}}$$

5. 移码表示法

补码表示很难直接判断其真值大小

如

十进制	二进制	补码		
$x = +21$	$+10101$	$0,10101$		错
$x = -21$	-10101	$1,01011$	大	
$x = +31$	$+11111$	$0,11111$		错
$x = -31$	-11111	$1,00001$	大	

$x + 2^5$

$+10101 + 100000 = 110101$ 大

$-10101 + 100000 = 001011$

正确

$+11111 + 100000 = 111111$ 大

$-11111 + 100000 = 000001$

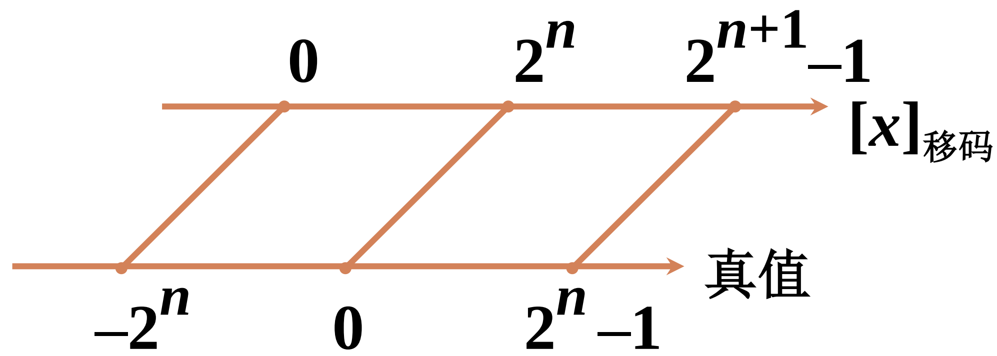
正确

(1) 移码定义

$$[x]_{移} = 2^n + x \quad (2^n > x \geq -2^n)$$

x 为真值，n 为 整数的位数

移码在数轴上的表示



如 $x = 10100$

$$[x]_{移} = 2^5 + 10100 = 1{,}10100$$

$x = -10100$

$$[x]_{移} = 2^5 - 10100 = 0{,}01100$$

用 逗号 将符号位和数值位隔开

(2) 移码和补码的比较

设 $x = +1100100$

$[x]_{移} = 2^7 + 1100100 = 1,1100100$

$[x]_{补} = 0,1100100$

设 $x = -1100100$

$[x]_{移} = 2^7 - 1100100 = 0,0011100$

$[x]_{补} = 1,0011100$

补码与移码只差一个符号位

(3) 真值、补码和移码的对照表

真值 x (n=5)	$[x]_{补}$	$[x]_{移}$	$[x]_{移}$对应的十进制整数
- 1 0 0 0 0 0	1 0 0 0 0 0	0 0 0 0 0 0	0
- 1 1 1 1 1	1 0 0 0 0 1	0 0 0 0 0 1	1
- 1 1 1 1 0	1 0 0 0 1 0	0 0 0 0 1 0	2
⋮	⋮	⋮	⋮
- 0 0 0 0 1	1 1 1 1 1 1	0 1 1 1 1 1	31
± 0 0 0 0 0	0 0 0 0 0 0	1 0 0 0 0 0	32
+ 0 0 0 0 1	0 0 0 0 0 1	1 0 0 0 0 1	33
+ 0 0 0 1 0	0 0 0 0 1 0	1 0 0 0 1 0	34
⋮	⋮	⋮	⋮
+ 1 1 1 1 0	0 1 1 1 1 0	1 1 1 1 1 0	62
+ 1 1 1 1 1	0 1 1 1 1 1	1 1 1 1 1 1	63

(4) 移码的特点

6.1

- 当 $x = 0$ 时 $[+0]_{移} = 2^5 + 0 = 1,00000$

  $[-0]_{移} = 2^5 - 0 = 1,00000$

  $\therefore [+0]_{移} = [-0]_{移}$

- 当 $n = 5$ 时 最小的真值为 $-2^5 = -100000$

  $[-100000]_{移} = 2^5 - 100000 = 000000$

  可见，最小真值的移码为全 0

  用移码表示浮点数的阶码

  能方便地判断浮点数的阶码大小

例：无符号数在C语言中对应**unsigned short**、**unsigned int(unsigned)**、**unsigned long**，带符号整数表示为**short**、**int**、**long**类型，求以下程序段在一个**32**位机器上运行时输出的结果，并说明为什么。

```
1 int x=-1;
2 unsigned u=2 147 483 648;
3 printf ("x=%u=%d\n", x , x);
4 printf ("u=%u=%d\n", u , u);
```

说明：

- 其中**printf**为输出函数，指示符**%u**、**%d**分别表示以无符号整数和 有符号整数形式输出十进制数的值，
- 2 147 483 648=2^{31}

解：因为现代计算机中带符号整数都是用补码表示的，**-1**的补码整数表示为"**11…1**"（共**32**位），当作**32**位无符号数时，因此，十进制表示为 **2^{32}-1=4 294 967 296-1= 4 294 967 295**

2^{31}的无符号整数在计算机中表示为"**10…0**"，当作为有符号整数输出时，其值为最小负数**-2^{32-1}=-** 2 147 483 648

输出结果：

x= 4 294 967 295=-1

u=2 147 483 648=- 2 147 483 648

例：以下为C语言程序，用来计算一个数组a中每个元素的和，当参数len为0时，返回值应该为0，却发生了存储器访问异常。请问这是什么原因造成的？说明如何修改。

```
float  sum_elements  ( float a[ ], unsigned len)
 {
         int     i;
         float  result=0;
          for ( i=0; i<=len-1; i++ )
                  result+=a[ i ] ;
           return result;
 }
```

解： 存储器访问异常是由于对数组a访问时产生了越界错误造成的。循环变量i是int型，而len是unsigned型，当len为0时，执行len-1的结果为32个1，是最大可表示的32位无符号数，任何无符号数都比它小，使循环体不断被执行，导致数组访问越界，因而发生存储器访问异常，应当将len声明为int型。

小数点按约定方式标出

一、定点表示

$\boxed{S_f}\underbrace{\boxed{S_1 S_2 \cdots S_n}}_{\text{数值部分}}$ 或 $\boxed{S_f}\underbrace{\boxed{S_1 S_2 \cdots S_n}}_{\text{数值部分}}$

数符　小数点位置　　　　数符　小数点位置

定点机	小数定点机	整数定点机
原码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$
补码	$-1 \sim +(1-2^{-n})$	$-2^n \sim +(2^n-1)$
反码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$

二、浮点表示 6.2

$N = S \times r^j$　　浮点数的一般形式

S 尾数　j 阶码　r 基数（基值）

计算机中 r 取 2、4、8、16 等

当 $r = 2$　　$N = 11.0101$

二进制表示

$= 0.110101 \times 2^{\boxed{10}}$　规格化数

$= 1.10101 \times 2^{1}$

$= 1101.01 \times 2^{-10}$

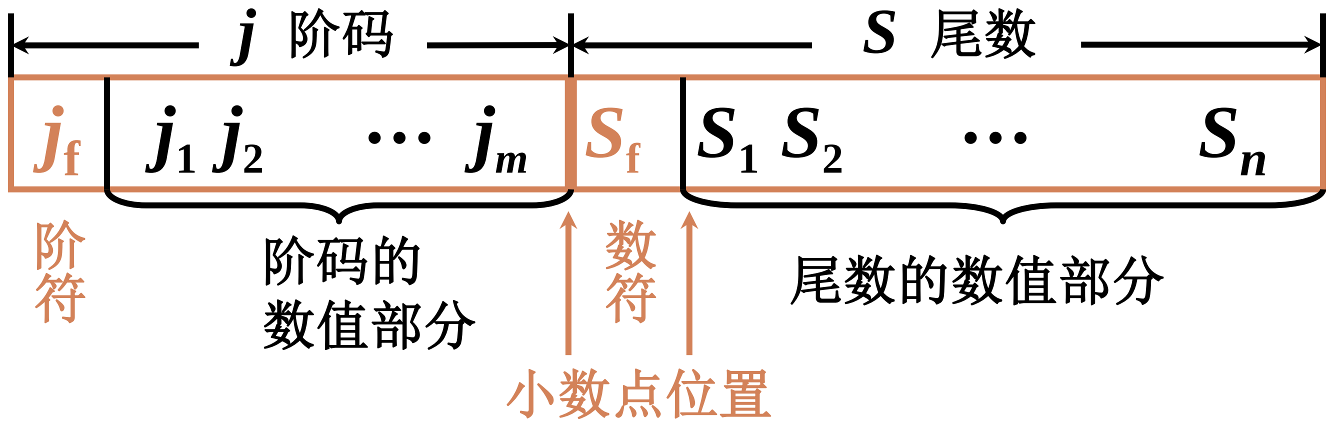
$= 0.00110101 \times 2^{100}$

尾数为纯小数

计算机中　S　小数、可正可负

　　　　　j　整数、可正可负

1. 浮点数的表示形式

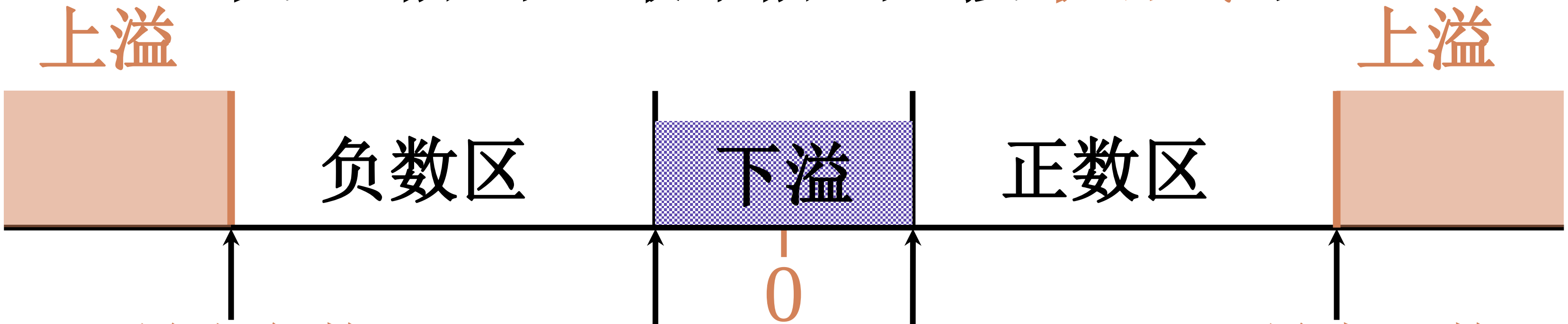


S_f	代表浮点数的符号
n	其位数反映浮点数的精度
m	其位数反映浮点数的表示范围
j_f 和 m	共同表示小数点的实际位置

2. 浮点数的表示范围

上溢　阶码 > 最大阶玛

下溢　阶码 < 最小阶码　按 机器零 处理



最小负数

$$-2^{(2^m-1)} \times (1-2^{-n})$$

$$-2^{15} \times (1-2^{-10})$$

最小正数

$$2^{-(2^m-1)} \times 2^{-n}$$

$$2^{-15} \times 2^{-10}$$

最大正数

$$2^{(2^m-1)} \times (1-2^{-n})$$

$$2^{15} \times (1-2^{-10})$$

最大负数

$$-2^{-(2^m-1)} \times 2^{-n}$$

$$-2^{-15} \times 2^{-10}$$

设 $m = 4$

$n = 10$

练习 6.2

设机器数字长为 24 位，欲表示±3万的十进制数，试问在保证数的最大精度的前提下，除阶符、数符各取1 位外，阶码、尾数各取几位？

解： ∵ $2^{14} = 16384$ $2^{15} = 32768$

∴ 15 位二进制数可反映 ±3 万之间的十进制数

$$2^{\boxed{15}} \times 0.\underbrace{\times\times\times\ \dots\dots\ \times\times}_{18\text{位}}$$

$2^4 = 16$ $m = 4、5、6 \cdots$

满足 最大精度 可取 $m = 4$，$n = 18$

3. 浮点数的规格化形式

$|s| \geqslant 1/r$

$r = 2$　尾数最高位为 1

$r = 4$　尾数最高 2 位不全为 0

$r = 8$　尾数最高 3 位不全为 0

基数不同，浮点数的规格化形式不同

$0.\times\times\times\times\times\times\times\times$

$2^{-1}2^{-2}2^{-3}2^{-4}\ldots$

$r = 4$

$r = 8$

	√	0.0101110	√
√	√	0.1000101	√
		0.0010101	√

4. 浮点数的规格化

$r = 2$	左规	尾数左移 1 位，阶码减 1
	右规	尾数右移 1 位，阶码加 1
$r = 4$	左规	尾数左移 2 位，阶码减 1
	右规	尾数右移 2 位，阶码加 1
$r = 8$	左规	尾数左移 3 位，阶码减 1
	右规	尾数右移 3 位，阶码加 1

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度降低

例如：设 $m = 4$，$n = 10$

尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.\underbrace{1111111111}$ $= 2^{15} \times (1-2^{-10})$

$2^{-15} \times 2^{-10} = 2^{-25}$

最小正数 $2^{-1111} \times 0.1\underbrace{000000000}$ $= 2^{-15} \times 2^{-1} = 2^{-16}$

$-2^{-15} \times 2^{-10} = -2^{-25}$

最大负数 $2^{-1111} \times (-0.1\underbrace{000000000}_{9个0})$ $= -2^{-15} \times 2^{-1} = -2^{-16}$

最小负数 $2^{+1111} \times (-0.\underbrace{1111111111}_{10个1})$ $= -2^{15} \times (1-2^{-10})$

三、举例

例 6.13　将 $+\frac{19}{128}$ 写成二进制定点数、浮点数及在定点机和浮点机中的机器数形式。其中数值部分均取 10 位，数符取 1 位，浮点数阶码取 5 位（含1位阶符）。

解：　设 $x = +\frac{19}{128}$?

二进制形式　　$x = 0.0010011$

定点表示　　$x = 0.0010011\,000$

浮点规格化形式　　$x = 0.1001100000 \times 2^{-10}$

定点机中　　$[x]_{原} = [x]_{补} = [x]_{反} = 0.0010011000$

浮点机中　　$[x]_{原} = 1, 0010;\ 0.1001100000$

$[x]_{补} = 1, 1110;\ 0.1001100000$

$[x]_{反} = 1, 1101;\ 0.1001100000$

例 6.14　将 –58 表示成二进制定点数和浮点数，并写出它在定点机和浮点机中的三种机器数及阶码为移码，尾数为补码的形式（其他要求同上例）。

解：　设 $x = -58$

二进制形式　$x = -111010$

定点表示　$x = -0000\,111010$

浮点规格化形式　$x = -(0.1110100000) \times 2^{110}$

定点机中

$[x]_{原} = 1, 0000111010$

$[x]_{补} = 1, 1111000110$

$[x]_{反} = 1, 1111000101$

浮点机中

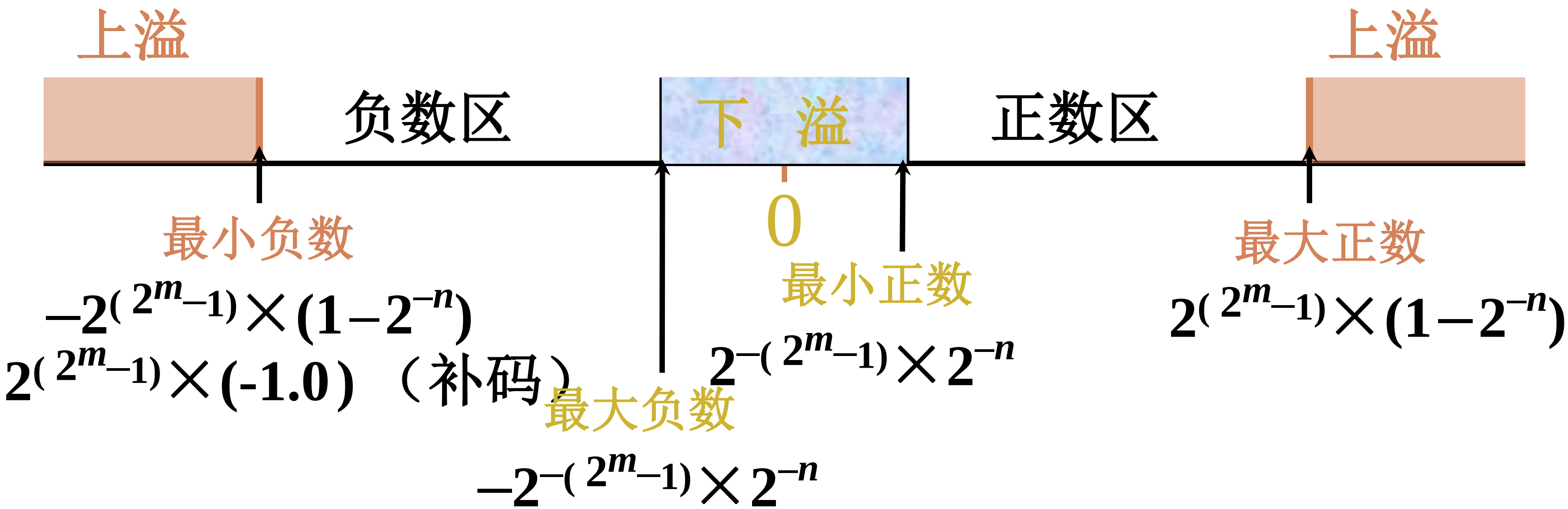
$[x]_{原} = 0, 0110; 1. 1110100000$

$[x]_{补} = 0, 0110; 1. 0001100000$

$[x]_{反} = 0, 0110; 1. 0001011111$

$[x]_{阶移、尾补} = 1, 0110; 1. 0001100000$

例6.15　写出对应下图所示的浮点数的补码形式。设 $n = 10$，$m = 4$，阶符、数符各取 1位。



解：

	真值	补码
最大正数	$2^{15}\times(1-2^{-10})$	0,1111; 0.1111111111
最小正数	$2^{-15}\times 2^{-10}$	1,0001; 0.0000000001
最大负数	$-2^{-15}\times 2^{-10}$	1,0001; 1.1111111111
最小负数	$-2^{15}\times(1-2^{-10})$	0,1111; 1.0000000001
负数边界	$2^{15}\times(-1.0)$	0,1111; 1.0000000000

机器零 6.2

- 当浮点数尾数为 0 时，不论其阶码为何值按机器零处理
- 当浮点数 阶码等于或小于它所表示的最小数时，不论尾数为何值，按机器零处理

如 $m = 4 \quad n = 10$

当阶码和尾数都用补码表示时，机器零为

×, × × × ×；　0. 0 0 …… 0

（阶码 = −16）　1, 0 0 0 0；　×.×× …… ×

当阶码用移码，尾数用补码表示时，机器零为

0, 0 0 0 0；　0. 0 0 …… 0

有利于机器中“ 判 0 ” 电路的实现