

第五章 运算方法与运算器(二)

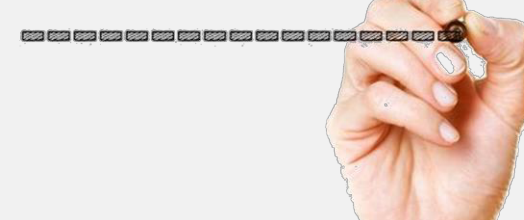
秦磊华 计算机学院

基于补码数据表示研究运算方法和设计运算器(简)

5.2 浮点数据表示

5.3 浮点数加/减运算

CONTENT



为什么要引入浮点数?

- ◆ **问题**: 三位十进制数能表示的最大数?
- ◆ 表示太阳质量 $2 \times 10^{30} \text{kg}$ 需要多少十进制位? 3 or 5
- ◆ 机器字长确定时, 浮点数能表示更大范围或更高精确度的数据;
如电子的质量 $9 \times 10^{-28} \text{g}$
- ◆ 科学记数法用**幂**和**尾数**来表示浮点数, 机器数中采用**阶码**和**尾数**来表示
- ◆ 浮点数可表示**纯整数**和**纯小数**以外的实数。



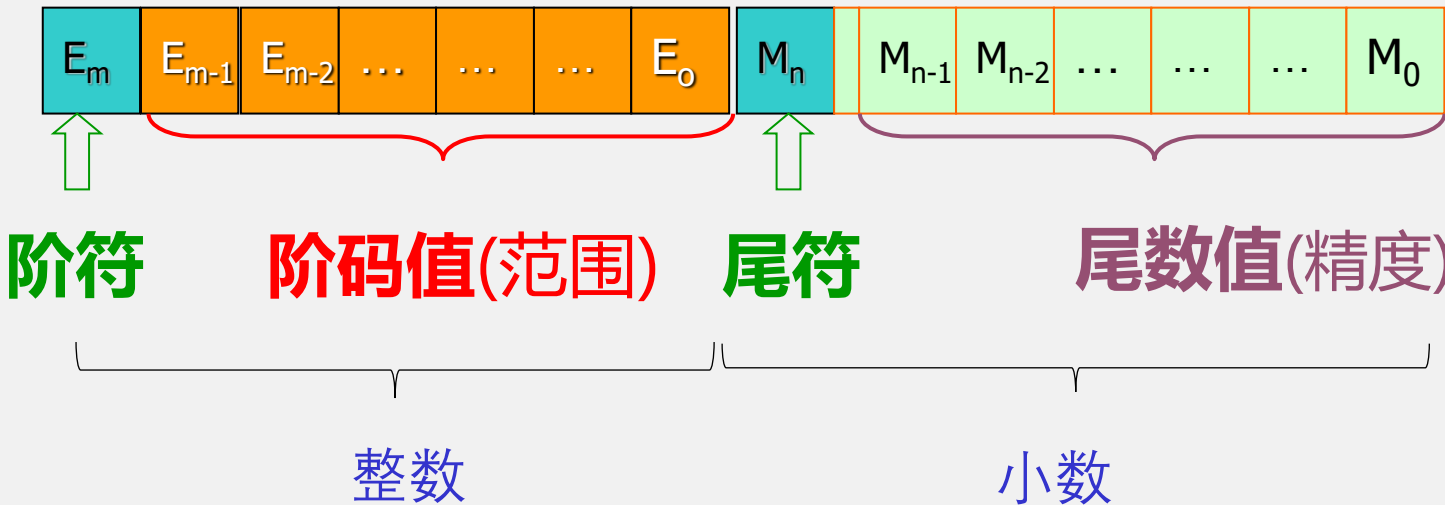
1.浮点数据表示

1) 计算机内浮点数的一般格式

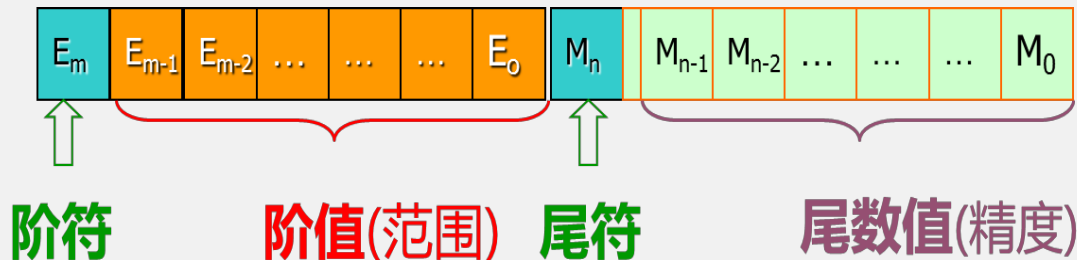
$$N=2^E \times M$$



$$N=2^{\pm e} \times (\pm m)$$



2) 计算机内浮点数的一般格式特点分析



- ◆ 机器字长确定的情况下，尾数位越长则阶码越短，如何取舍？
- ◆ 早期各计算机公司不同型号计算机，浮点数表示千差万别；
- ◆ 数据交换、软件可移植性、计算机协同工作等都受到影响。

3) IEEE 754格式

1985年完成浮点数标准IEEE 754制定，至今仍被许多CPU浮点运算采用。



UC Berkeley math professor
William Kahan.

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

5.2 浮点数据表示

What Every Computer Scientist Should Know About
Floating-Point Arithmetic。DAVID GOLDBERG。
ACM Computing Surveys, Vol 23, No 1, March 1991

IEEE 754格式

32浮点数 *float*



64浮点数 *double*



- ◆ 机器数构成：阶码E，尾数M，符号位S.
- ◆ $N = (-1)^S \times 1.M \times 2^e$ (规格化尾数，1隐藏)
- ◆ 阶码E移码表示，偏移量127/1023， $E = e + 127/1023$;尾数原码表示
- ◆ **偏移值127/128**: 综合考虑表示计算机系统中实数集合和表示特殊值。

$$e_{\min} = -127, e_{\max} = 126 \text{ (} E=128 \text{)}; e_{\min} = -126, e_{\max} = 127 \text{ (} E=127 \text{)};$$



5.2 浮点数据表示

符号位	阶码	尾数	表示的数据
0/1	255	1xxxx	NaN Not a Number *
0/1	255	非零 0xxxx	sNaN Signaling NaN **
0	255	0	+ ∞
1	255	0	- ∞
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$ (规格化)
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$ (非规格化)
0/1	0	0	+0/-0

** sNaN 是NaN的一种，例如可以用来检查变量是否被初始化

5.2 浮点数据表示

返回NaN的运算有如下三种

- 1)至少有一个参数是NaN的运算
- 2)不定式

- ◆除法运算： $0/0$ 、 ∞/∞ 、 $\infty/-\infty$ 、 $-\infty/\infty$ 、 $-\infty/-\infty$
- ◆乘法运算： $0\times\infty$ 、 $0\times-\infty$
- ◆加法运算： $\infty + (-\infty)$ 、 $(-\infty) + \infty$
- ◆减法运算： $\infty - \infty$ 、 $(-\infty) - (-\infty)$

3)产生复数结果的实数运算。例如：

- ◆对负数进行开偶次方的运算
- ◆对负数进行对数运算
- ◆对正弦或余弦到达域以外的数进行反正弦或反余弦运算

符号位	阶码	尾数	表示的数据
0/1	255	1xxxx	NaN Not a Number *
0/1	255	非零 0xxxx	sNaN Signaling NaN **
0	255	0	$+\infty$
1	255	0	$-\infty$
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$ (规格化)
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$ (非规格化)
0/1	0	0	$+0/-0$

** sNaN 是NaN的一种，例如可以用来检查变量是否被初始化

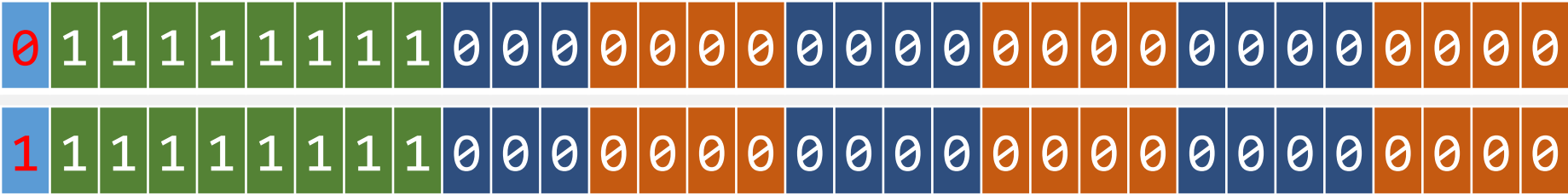
5.2 浮点数据表示

```
main()
{
    float a=1.0,b=-1.0;
    a=a/0; b=b/0;
    printf("a=%f b=%f",a,b);
    return;
}
```

符号位	阶码	尾数	表示的数据
0/1	255	1xxxx	NaN Not a Number *
0/1	255	非零 0xxxx	sNaN Signaling NaN **
0	255	0	+ ∞
1	255	0	- ∞
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$ (规格化)
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$ (非规格化)
0/1	0	0	+0/-0

** sNaN 是NaN的一种，例如可以用来检查变量是否被初始化

a=1.#INF00 b=-1.#INF00



5.2 浮点数据表示

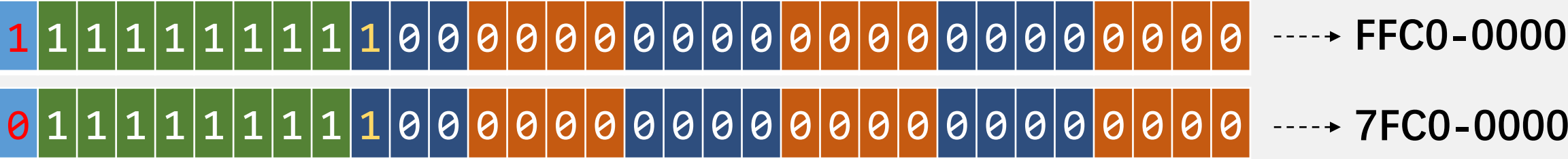
```
main()  
{  
    float a=0.0, b;  
    a=a/0; b=-sqrt(-1);  
    printf ("a=%f b=%f",a,b);  
    return;  
}
```

符号位	阶码	尾数	表示的数据
0/1	255	1xxxx	NaN Not a Number *
0/1	255	非零 0xxxx	sNaN Signaling NaN **
0	255	0	+ ∞
1	255	0	- ∞
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$ (规格化)
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$ (非规格化)
0/1	0	0	+0/-0

** sNaN 是NaN的一种，例如可以用来检查变量是否被初始化

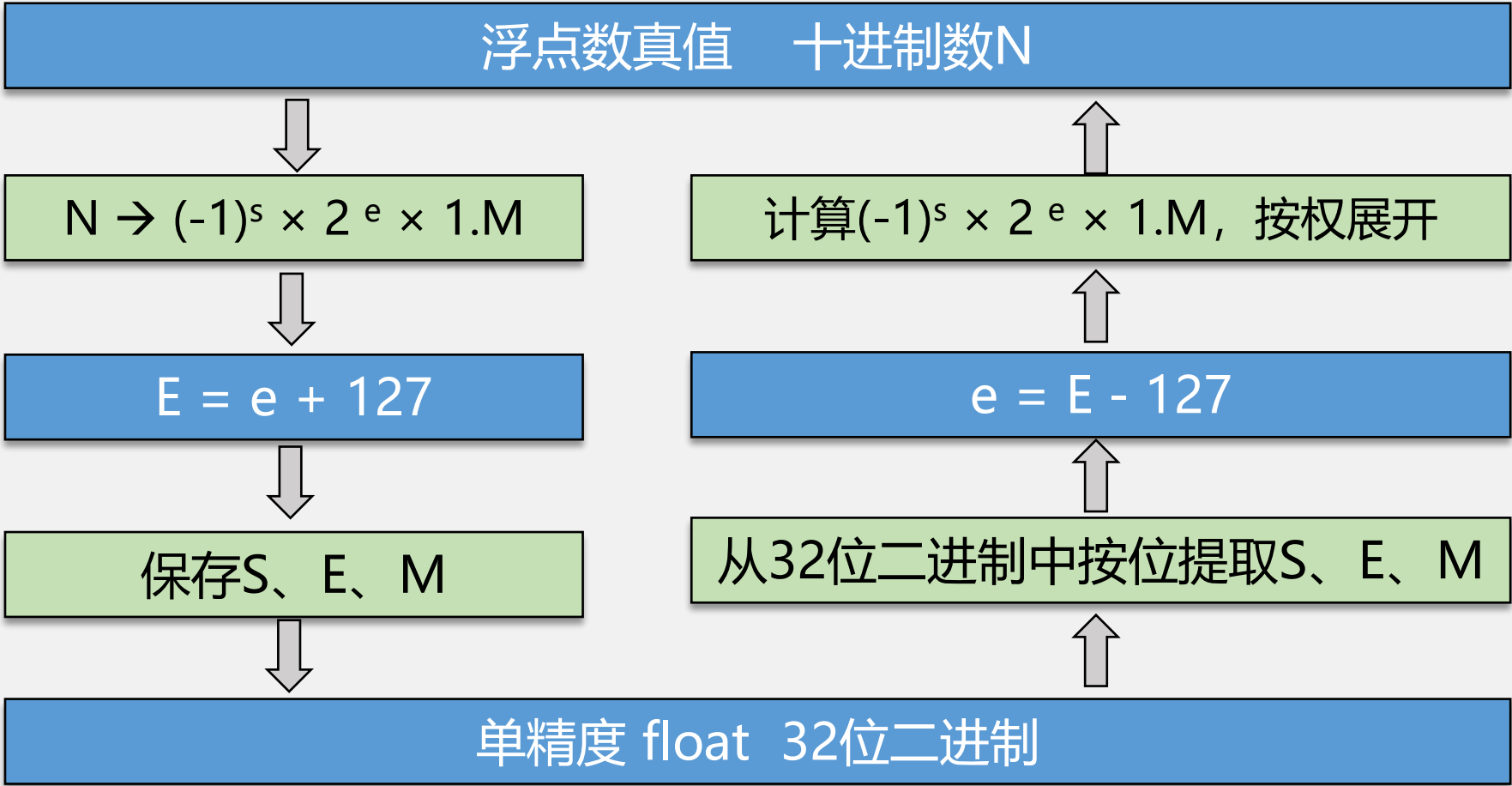
a=1.#IND00 b=1.#QNAN0

IND: Implementation Not define

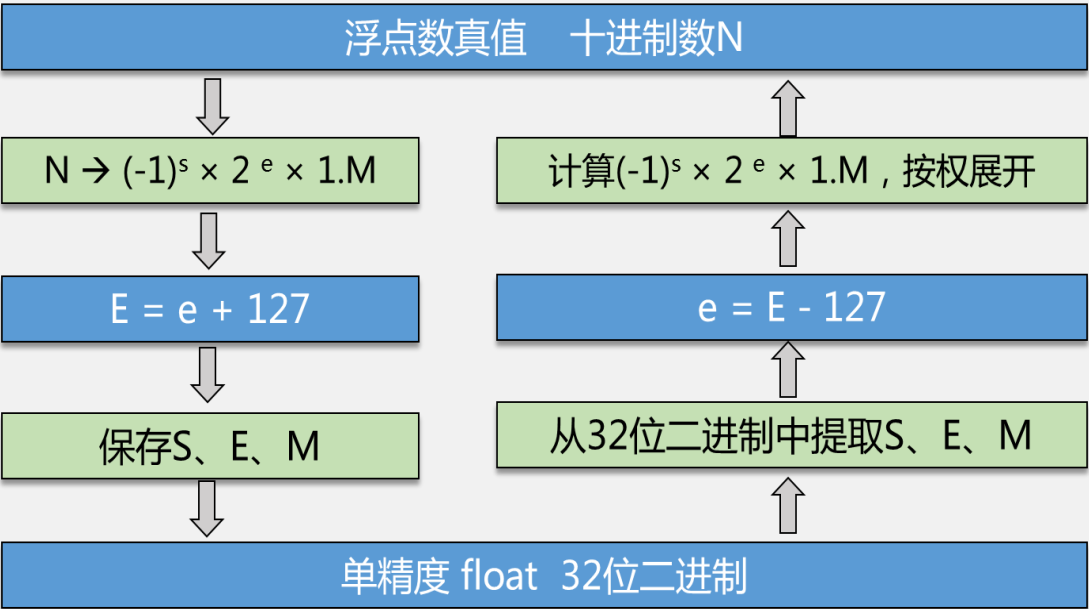


5.2 浮点数据表示

4) IEEE 754格式与浮点数真值的转换



5.2 浮点数据表示



例 3.3₁₀ → IEEE 754 float

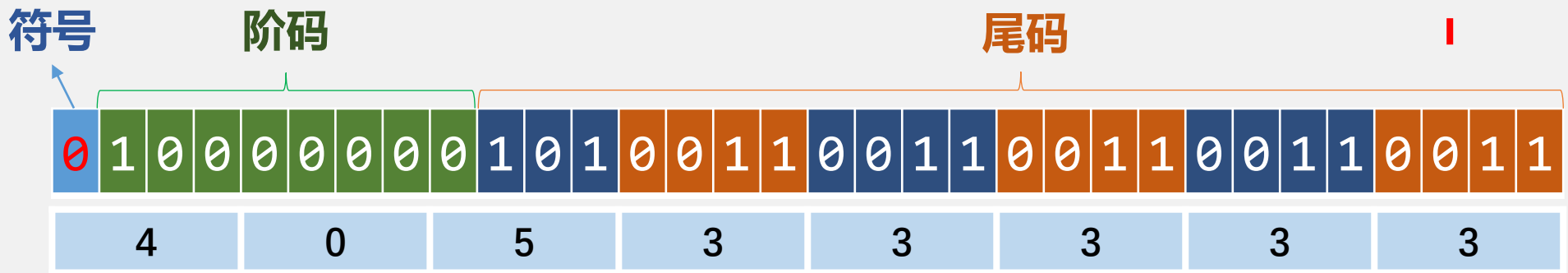
3.3₁₀ = 11.01001100110011001100110011...₂

= 1.101001100110011001100110011 × 2¹

S = 0, e = 1,

E = 01111111 + 1 = 100000000

M = 10100110011001100110011



5.2 浮点数据表示

```
main()
{
    double a,b,c;  int d;
    b=3.3;  c=1.1;
    a=b/c;
    d=b/c;
    printf("%f,%d",a,d);
    if (3.0!=a)
        printf("\nReally? 3.0!=a");
}
```

3.000000,2

Really? 3.0!=a

$$(2^{-126} + 10^{20}) - 10^{20} = ? \quad 2^{-126} + (10^{20} - 10^{20}) = ?$$

由于精确度损失问题，浮点运算不满足结合律

5.3 浮点运算(加/减)-- 一般格式的浮点数

1.浮点数加/减运算方法

$$X = 2^{E_X} M_X \quad Y = 2^{E_Y} M_Y$$

$$X \pm Y = ?$$

$$\text{如: } E_X = E_Y \quad S = 2^{E_X} (M_X \pm M_Y)$$

$$\text{如: } E_X \neq E_Y \quad ? \quad \text{使两数的阶码变成相等} \implies \text{对阶}$$

浮点运算步骤： 对阶、尾数运算、规格化、舍入、溢出判断

5.3 浮点运算(加/减)-- 一般格式的浮点数

1. 浮点数加/减运算方法

1) 对阶 (使得两者的阶码相等)

$$2 \times 10^8 + 3 \times 10^5$$
$$= 2 \times 10^8 + 3 \times 10^8 \times 10^{-3}$$

大变小 or 小变大 ? \implies 小变大! , 对阶的同时, 尾数要同步右移

$$2^8 * (0.11000) + 2^6 * (0.00111) = 2.003 \times 10^8$$

◆ 大阶变小阶:

$$2^8 * (0.11000) \rightarrow 2^6 * (\underline{11}.000) \rightarrow 2^5 * (\underline{11}0.000)$$

◆ 小阶变大阶:

$$2^6 * (0.00111) \rightarrow 2^7 * (0.000011\underline{1}) \rightarrow 2^8 * (0.000001\underline{11})$$

数部会丢

尾数字长定, 也会丢

5.3 浮点运算(加/减)-- 一般格式的浮点数

2) 运算结果规格化

什么意思?

(1) 为什么需要进行规格化 - 浮点数可表达的多样性

$$2^7 * (00.11000) = 2^8 * (00.01100)$$

$$2^7 * (10.11000) = 2^8 * (11.01100)$$

$$2^7 * (01.11000) = 2^8 * (00.11100)$$

尾数不为0

绝对值大于1/2

5.3 浮点运算(加/减)-- 一般格式的浮点数

2) 运算结果规格化

(2)规格化的标准是什么?

00.1 或 11.0

尾数不为零时, 其绝对值大于等于1/2

◆真值规格化数 0.1XXXX -0.1XXXX

◆原码规格化数 0.1XXXX 1.1XXXX

◆补码规格化数 00.1XXXX 11.0XXXX

◆补码非规格化数 00.0XXXX 11.1XXXX

01.XXXXX 10.XXXXX

5.3 浮点运算(加/减)-- 一般格式的浮点数

2) 运算结果规格化

(3) 如何规格化? - 使非规格化尾数变成规格化尾数, 同步变化阶码

补码规格化数 00.1XXXX 11.0XXXX

00.0XXXX 11.1XXXX

01.XXXXX 10.XXXXX

左移规格化

左移几位?

右移规格化

右移1位

只会变为1.0 or 00.1

在规格化
化也多!!!

① 什么是规格化数

② 符号位?

③ 12位?

④ 实际意义
不符

3) 舍入处理

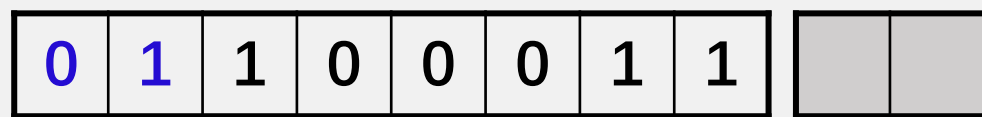
0	1	1	0	0	0	1	1	
0	0	1	1	0	0	0	1	1

- ◆ 右移动规格化后低位部分丢失了数据位，从而产生误差
- ◆ 有多种处理方法：0舍1入，~~截去法~~在末位恒值1
- ◆ 不同舍入方式对精度产生不同的影响！如果减少影响？

5.3 浮点运算(加/减)-- 一般格式的浮点数

3) 舍入处理

增加浮点运算附加位



Guard
(保护位)

Round
(舍入位)

IEEE 754

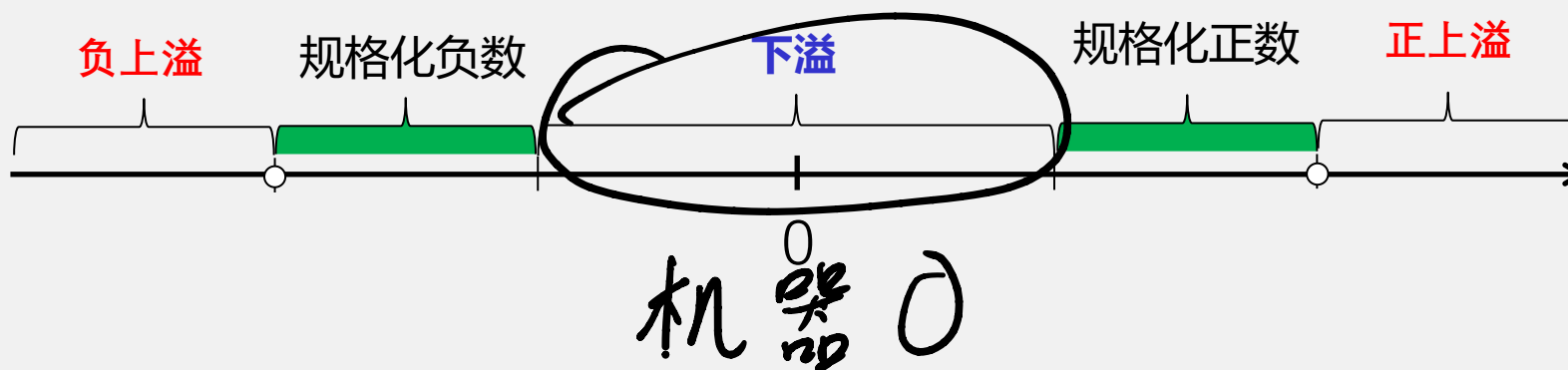
有明确移出
的又移回来了

增加浮点运算附加位 意义何在?

5.3 浮点运算(加/减)-- 一般格式的浮点数

4) 溢出的标准及处理

- ◆ 通过阶码是否溢出判断浮点数的溢出情况(双符号位检测)
- ◆ 阶码符号位为 01 \rightarrow 浮点数上溢 $|X| \rightarrow \infty$
- ◆ 阶码符号位为 10 \rightarrow 浮点数下溢 $|X| \rightarrow 0$



5.3 浮点运算(加/减) -- 一般格式的浮点数

为74格式

$1.xxxx \rightarrow xx.xx$

2. 浮点数加/减运算举例

例1 两浮点数 $x = 2^{101} \times 0.11011011$, $y = 2^{111} \times (0.10101100)$ 。假设尾数在计算机中以补码表示, 尾数位共12位, 采用双符号位, 阶码以补码表示, 共5位, 也采用双符号位, 求 $x + y$ 。

解: 将 x, y 转换成浮点数据格式

$$[x]_{\text{浮}} = 00\ 101, 00.11011011$$

$$[Y]_{\text{浮}} = 00\ 111, 11.01010100$$

1) 对阶

$$[E_x - E_y]_{\text{补}} = [E_x]_{\text{补}} + [-E_y]_{\text{补}} = 00101 + 11001 = 11110 = -2 < 0$$

小阶对大阶, x 阶码加2, x 尾数右移2位

5.3 浮点运算(加/减)-- 一般格式的浮点数

$$[X]_{\text{浮}} = 00\ 111, 00.00110110\textcolor{red}{11} \quad \text{保留位}$$

$$[Y]_{\text{浮}} = 00\ 111, 11.01010100\ 00$$

2) 尾数求和

$$[X+Y]_{\text{浮}} = 00\ 111, 11.10001010\textcolor{red}{11} \quad \text{保留位参与运算}$$

3) 结果规格化

$$[X+Y]_{\text{浮}} = 00\ 110, 11.00010101\textcolor{red}{1} \quad \text{非规数, 左归一位, 阶码减一}$$

4) 舍入处理

$$[X+Y]_{\text{浮}} = 00\ 110, 11.00010110 \quad (0\text{舍}1\text{入法})$$

$$[X+Y]_{\text{浮}} = 00\ 110, 11.00010101 \quad (\text{截去法})$$

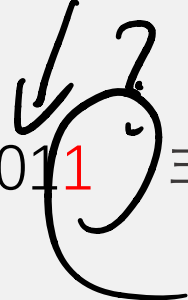
5) 溢出判断

$$[X+Y]_{\text{浮}} = 2^{110} \times (-0.11101011) \quad \text{无溢出}$$

前面有说12位,

实际上'11'
不作保留位

符号



5.3 浮点运算(加/减)-- 一般格式的浮点数

例2, 已知 $X=2^{11} \times 0.11111111$, $Y=2^{11} \times 0.10000001$, 求 $X+Y$

解: $[X]_{\text{浮}}=00111, 00.1111\ 1111$, $[Y]_{\text{浮}}=00111, 00.1000\ 0001$

1)尾数求和

$$[X+Y]_{\text{浮}} = 00111, 01.1000\ 0000$$

2)结果规格化

$$[X+Y]_{\text{浮}} = \mathbf{01}\ 000, \mathbf{00.1100}\ 0000 \quad \text{右移一位规格化, 阶码} +1$$

3)舍入处理

$$[X+Y]_{\text{浮}} = \mathbf{01}\ 000, \mathbf{00.11000000}$$

4)溢出判断

阶码双符号位 01, 上溢



3.计算机中关于运算的案例分析

```
union { char c[4]; float f; int i;} t1, t2, t3;
main() {
    t1.i = 0X7F000000; t2.i = 0X7F7FFFFFFF; // Max float
    t3.f = t1.f + t1.f;
    printf("%08X    %f\n", t1.i, t1.f);
    printf("%08X    %f\n", t2.i, t2.f);
    printf("%08X    %f\n", t3.i, t3.f);
}
```

???

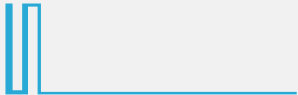
7F000000	1701411834604692300000000000000000000000000000000.000000
7F7FFFFFFF	3402823466385288600000000000000000000000000000000.000000
7F800000	1.#INF00

U

```
union { char c[4]; float f; int i;} t1,t2,t3,t4;
main() {
    t1.i = 0x00000001; t2.i = 0x00C00000; t3.i = 0x00800000; //Minus float
    t4.f = t2.f - t3.f;
    printf("%08X %.61f\n",t1.i,t1.f); printf("%08X %.61f\n",t2.i,t2.f);
    printf("%08X %.61f\n",t3.i,t3.f); printf("%08X %.61f\n",t4.i,t4.f);}
```

t2.f 0 0 0 0 0 0 0 0 1 1 0 ⇒ 1.1(尾数)

[illegible][illegible][illegible][illegible][illegible]



第二部分完