

第6讲

浮点数表示及运算

吴海军

南京大学计算机科学与技术系





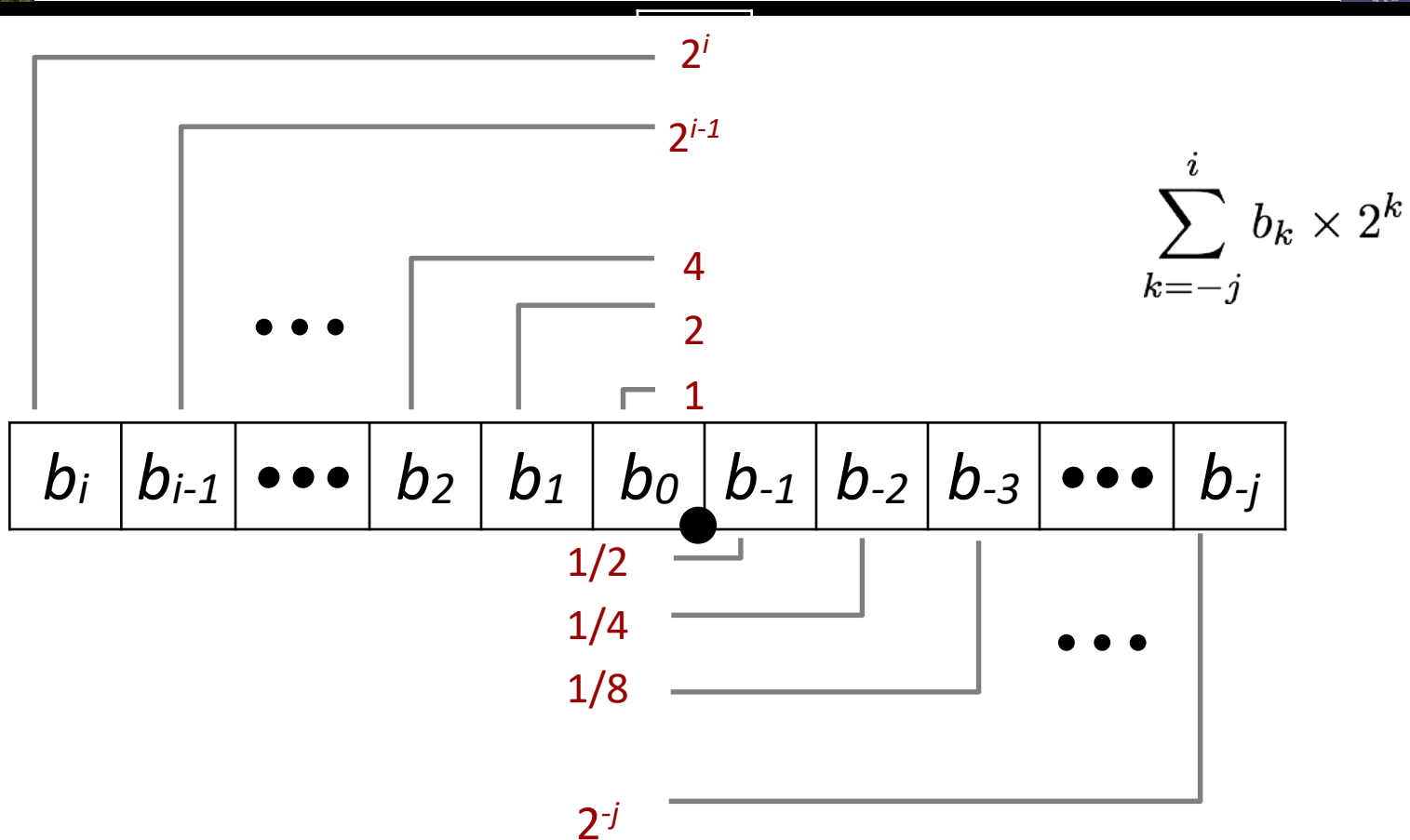
主要内容



- 二进制小数表示
- IEEE浮点数标准定义
- 浮点数特性
- 浮点数运算
- 浮点数在C语言中的使用



二进制小数表示





二进制小数实例



Value Representation

$5 \frac{3}{4}$ 101.11_2
 $2 \frac{7}{8}$ 010.111_2

Value Representation

$1/3$ $0.0101010101[01]..._2$
 $1/5$ $0.001100110011[0011]..._2$
 $1/10$ $0.0001100110011[0011]..._2$

Value Representation

13亿 $1100\ 0001\ 1011\ 0111\ 0001\ 0000\ 1000\ 0000\ 00_2$

地球的质量 $5.98 \times 10^{24} \text{ kg} = (\quad ? \quad)_2$ 需要多少位二进制数表示

受限:

只能精确表示形如 $x/2^k$

仅考虑有限长度的编码, 只能近似表示小数的值
表示很大的数比较困难



科学计数法与浮点数



二进制数科学记数法:

mantissa (尾数) \rightarrow 0.101_{two} \times 2^{-10} \leftarrow *exponent* (指数)

\nwarrow *binary point* \nearrow 基为2

只要对尾数和指数分别编码，就可表示一个浮点数。



浮点数标准-IEEE 754



1980之前，各个计算机厂商的机器内部的浮点数表示格式不统一，而且相互不兼容。

1970年代后期，IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE 754的制定

现在所有计算机都采用 **IEEE 754** 来表示浮点数

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.



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



Prof. William Kahan



浮点数表示形式



- IEEE754浮点标准形式 $V = (-1)^s M 2^E$
 - 符号位S(Sign): 决定这个数是正数($s=0$)还是负数($s=1$)。
 - 尾数M(significand): 是一个二进制小数, 范围在 $[1.0, 2.0)$
 - 阶码E(exponent): 对浮点数加权重, 权重是2的E次幂。
- 编码表示, 分3个字段:
 - 1位符号字段s
 - k位阶码字段exp: 阶码E的编码
 - n位小数字段frac: 尾数M的编码





浮点数的精度



- 单精度：32位



- 双精度：64位



- 高精度：80 位 (Intel only)





规格化数值



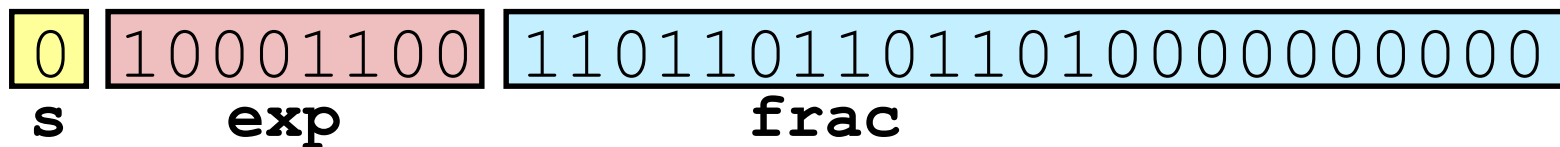
- 要求：阶码编码 exp 既不全0又不全1。
- 阶码编码 exp 表示阶码 E 的偏置(biased)形式： $\text{exp} = E + \text{bias}$
 - exp ：无符号数
 - $\text{bias} = 2^{k-1} - 1$ ， k 为阶码的位宽
 - 单精度：127 (Exp: 1...254, E: -126...127)
 - 双精度：1023 (Exp: 1...2046, E: -1022...1023)
- 小数字段 frac 表示尾数 M 的编码 $M = 1.\text{xxx}...\text{x}_2$
 - $\text{xxx}...\text{x}$ ：小数字段数值
 - 最小值：000...0 ($M = 1.0$)
 - 最大值：111...1 ($M = 2.0 - \varepsilon$)



单精度浮点数示例



- 数值: Float $F = 15213.0$;
 - $15213_{10} = 11101101101101_2$
 $= 1.1101101101101_2 \times 2^{13}$
- 小数字段Significand
 - $M = 1.\underline{1101101101101}_2$
 - $\text{frac} = \underline{1101101101101}0000000000_2$
- 阶码字段Exponent
 - $E = 13$
 - $\text{Bias} = 127$
 - $\text{Exp} = 140 = 10001100_2$
- Result:





单精度浮点数示例



求0xBEE00000H表示的IEEE 754单精度浮点数值

1011 11101 110 0000 0000 0000 0000 0000

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

符号位: 1, 则为负数。

阶码字段:

- $0111\ 1101_2 = 125$
- 偏置校正: $125 - 127 = -2$

小数字段:

$$\begin{aligned} & 1 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5} + \dots \\ & = 1 + 2^{-1} + 2^{-2} = 1 + 0.5 + 0.25 = 1.75 \end{aligned}$$

数值表示: $-1.75 \times 2^{-2} = -0.4375$



非规格化数值



- 阶码编码exp等于全0
 - 阶码编码exp: $E = 1 - Bias$
- 小数字段frac表示尾数M的编码M= 0.xxx...x₂
 - xxx...x: bits of frac
 - Exp= 000...0, frac= 000...0
 - 表示数值为0
 - 有正负0区分
 - Exp= 000...0, frac ≠ 000...0
 - 数值非常接近于0

隐含非规格化小数字段M小于1



特殊值



- 阶码编码exp等于全1
 - $\text{Exp} = 111\dots 1$, $\text{frac} = 000\dots 0$
 - 表示数值为 ∞
 - 当两个非常大的数相乘或除数为0时，表示数据溢出
 - $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
 - $\text{Exp} = 111\dots 11$, $\text{frac} \neq 000\dots 0$
 - 表示NaN，不是一个数（Not a Number）
 - 一些运算结果不能是实数或无穷，就会返回该值
 - E.g., $\text{sqrt}(-1)$, $\infty - \infty$, $\infty \times 0$



8位浮点数示(k-4,n=3)



正数表示范围

s exp frac E Value

0 0000 000 -6 0

0 0000 001 -6 $1/8 * 1/64 = 1/512$

0 0000 010 -6 $2/8 * 1/64 = 2/512$

...

0 0000 110 -6 $6/8 * 1/64 = 6/512$

0 0000 111 -6 $7/8 * 1/64 = 7/512$

0 0001 000 -6 $8/8 * 1/64 = 8/512$

0 0001 001 -6 $9/8 * 1/64 = 9/512$

...

0 0110 110 -1 $14/8 * 1/2 = 14/16$

0 0110 111 -1 $15/8 * 1/2 = 15/16$

0 0111 000 0 $8/8 * 1 = 1$

0 0111 001 0 $9/8 * 1 = 9/8$

0 0111 010 0 $10/8 * 1 = 10/8$

...

0 1110 110 7 $14/8 * 128 = 224$

0 1110 111 7 $15/8 * 128 = 240$

0 1111 000 n/a inf

非规格化数值

最小非规格化数值

数值接近0

最大非规格化数值

最小规格化数值

规格化数值

数值接近1

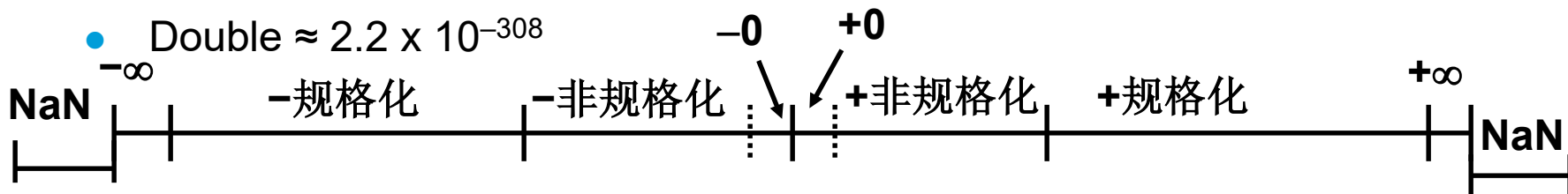
最大规格化数值



浮点数表示范围



Description	exp	frac	Numeric Value
• 0	00...00	00...00	+/- 0.0
• 最小的正规格化数值	00...01	00...00	$1.0 \times 2^{-\{126,1022\}}$
• 1	01...11	00...00	1.0
• 最大的规格化数值	11...10	11...11	$(2.0 - \epsilon) \times 2^{\{127,1023\}}$
• Single $\approx 3.4 \times 10^{38}$			
• Double $\approx 1.8 \times 10^{308}$			
• 最小的正非规格化数值	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
• Single $\approx 1.4 \times 10^{-45}$			
• Double $\approx 4.9 \times 10^{-324}$			
• 最大的非规格化数值	00...00	11...11	$(1.0 - \epsilon) \times 2^{-\{126,1022\}}$
• Single $\approx 1.18 \times 10^{-38}$			
• Double $\approx 2.2 \times 10^{-308}$			

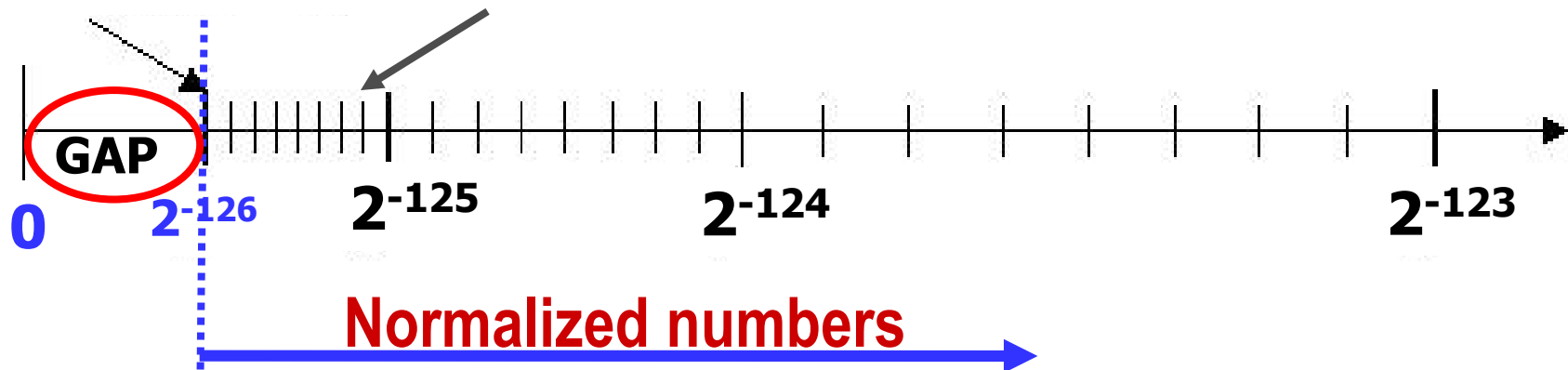




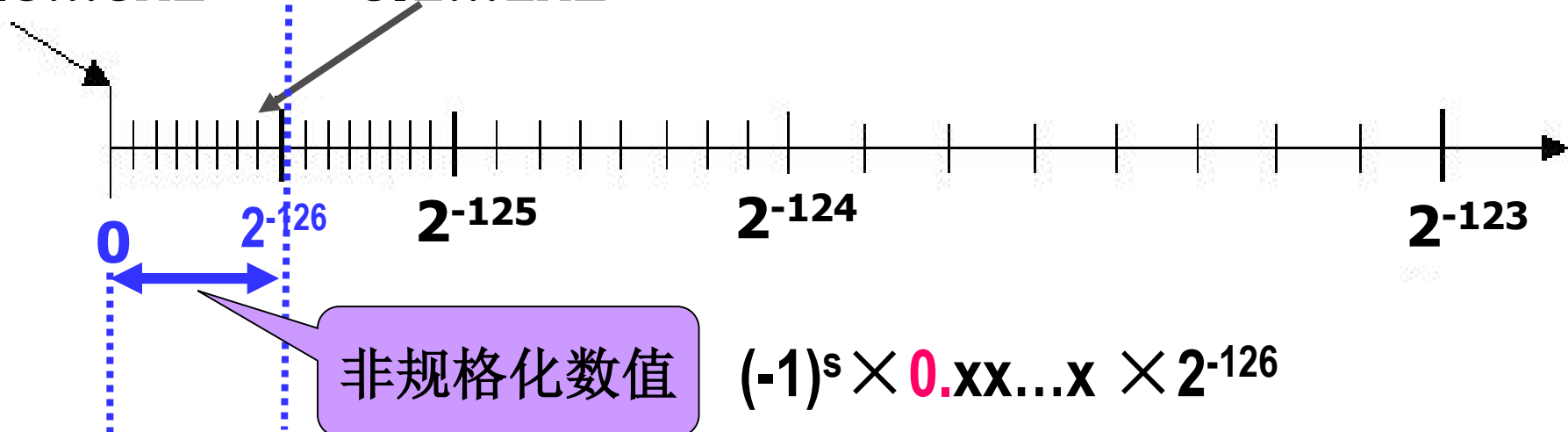
单精度非规格化数值范围



$1.0...0 \times 2^{-126} \sim 1.1...1 \times 2^{-126}$



$0.0...0 \times 2^{-126} \sim 0.1...1 \times 2^{-126}$



$$(-1)^s \times 0.xx...x \times 2^{-126}$$



浮点数运算及结果



设两个规格化浮点数分别为 $A = M_a \cdot 2^{E_a}$ $B = M_b \cdot 2^{E_b}$, 则:

$$A \pm B = (M_a \pm M_b \cdot 2^{-(E_a - E_b)}) \cdot 2^{E_a} \quad (\text{假设 } E_a \geq E_b)$$

1.5+1.5=?

$$A * B = (M_a * M_b) \cdot 2^{E_a + E_b}$$

1.5-1.0=?

$$A / B = (M_a / M_b) \cdot 2^{E_a - E_b}$$

上述运算结果可能出现以下几种情况:

SP最大指数为多少? 127!

阶码上溢: 一个正指数超过了最大允许值 $\Rightarrow +\infty / -\infty / \text{溢出}$

阶码下溢: 一个负指数超过了最小允许值 $\Rightarrow +0 / -0$ **SP最小指数为多少? -126!**

尾数溢出: 最高有效位有进位 \Rightarrow 右规 **尾数溢出, 结果不一定溢出**

非规格化尾数: 数值部分高位为0 \Rightarrow 左规

右规或对阶时, 右段有效位丢失 \Rightarrow 尾数舍入 **运算过程中添加保护位**

IEEE建议实现时为每种异常情况提供一个自陷允许位。若某异常对应的位为1, 则发生相应异常时, 就调用一个特定的异常处理程序执行。



IEEE754标准规定的五种异常情况



① 无效运算（无意义）

- 运算时有一个数是非有限数，如：

加 / 减 ∞ 、 $0 \times \infty$ 、 ∞ / ∞ 等

- 结果无效，如：

源操作数是NaN、 $0/0$ 、 $x \text{ REM } 0$ 、 $\infty \text{ REM } y$ 等

② 除以0（即：无穷大）

③ 数太大（阶码上溢）：对于SP，指阶码 $E > 1111\ 1110$ （指数大于127）

④ 数太小（阶码下溢）：对于SP，指阶码 $E < 0000\ 0001$ （指数小于-126）

⑤ 结果不精确（舍入时引起），例如1/3，1/10等不能精确表示成浮点数

上述情况硬件可以捕捉到，因此这些异常可设定让硬件处理，也可设定让软件处理。让硬件处理时，称为硬件陷阱。



整除0和浮点数除0的问题



```
#include <conio.h>
#include <stdio.h>
int main()
{
    int a=1,b=0;
    printf( "Division by zero:%d\n ", a/b);
    getchar();
    return 0;
}
```

为什么整数除0会发生异常而
浮点数除0不会?

```
int main()
{
    double x=1.0,y=-1.0,z=0.0;
    printf( "division by zero:%f %f\n ", x/z, y/z);
    getchar();
    return 0;
}
```

问题一:为什么整除int型会产生错误? 是什么错误?

二:用double型的时候结果为1. #INFOO 和 -1. #INFOO, 作何解释???



浮点数加/减运算



- 十进制科学计数法的加法例子

$$0.123 \times 10^5 + 0.560 \times 10^2$$

其计算过程为：

$$\begin{aligned} 0.123 \times 10^5 + 0.560 \times 10^2 &= 0.123 \times 10^5 + 0.000560 \times 10^5 \\ &= (0.123 + 0.000560) \times 10^5 \\ &= 0.12356 \times 10^5 = 0.124 \times 10^5 \end{aligned}$$

进行尾数加减运算前，必须“对阶”！最后还要考虑舍入
计算机内部的二进制运算也一样！

- “对阶”操作：目的是使两数阶码相等

- 小阶向大阶看齐，阶小的那个数的尾数右移，右移位数等于两个阶码差的绝对值
- IEEE 754尾数右移时，要将隐含的“1”移到小数部分，高位补0，移出的低位保留到特定的“附加位”上



浮点数加减法基本要点



(假定: X_m 、 Y_m 分别是X和Y的尾数, X_e 和 Y_e 分别是X和Y的阶码)

- (1) 求阶差: $\Delta e = Y_e - X_e$ (若 $Y_e > X_e$, 则结果的阶码为 Y_e)
- (2) 对阶: 将 X_m 右移 Δe 位, 尾数变为 $X_m * 2^{X_e - Y_e}$ (保留右移部分: 附加位)
- (3) 尾数加减: $X_m * 2^{X_e - Y_e} \pm Y_m$
- (4) 规格化:

当尾数高位为0, 则需左规: 尾数左移一次, 阶码减1, 直到MSB为1
每次阶码减1后要判断阶码是否下溢 (比最小可表示的阶码还要小)

当尾数最高位有进位, 需右规: 尾数右移一次, 阶码加1, 直到MSB为1
每次阶码加1后要判断阶码是否上溢 (比最大可表示的阶码还要大)

阶码溢出异常处理: 阶码上溢, 则结果溢出; 阶码下溢, 则结果为0

- (5) 如果尾数比规定位数长, 则需考虑舍入 (有多种舍入方式)
- (6) 若运算结果尾数是0, 则需要将阶码也置0。为什么?

尾数为0说明结果应该为0 (阶码和尾数为全0)。



浮点数加法运算举例



Example: 用二进制形式计算 $0.5 + (-0.4375) = ?$

解: $0.5 = 1.000 \times 2^{-1}$, $-0.4375 = -1.110 \times 2^{-2}$

对阶: $-1.110 \times 2^{-2} \rightarrow -0.111 \times 2^{-1}$

加减: $1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = 0.001 \times 2^{-1}$

左规: $0.001 \times 2^{-1} \rightarrow 1.000 \times 2^{-4}$

判溢出: 无

结果为: $1.000 \times 2^{-4} = 0.0001000 = 1/16 = 0.0625$

问题: 为何IEEE 754 加减运算右规时最多只需一次?

因为即使是两个最大的尾数相加, 得到的和的尾数也不会达到4, 故尾数的整数部分最多有两位, 保留一个隐含的“1”后, 最多只有一位被右移到小数部分。



舍入位



举例：十进制数，最终有效位数为 3，采用两位附加位（G、R）。

$$2.34\textcolor{blue}{0}\textcolor{red}{0} * 10^2$$

问题：若没有舍入位，采用就近舍入到偶数，则结果是什么？

$$0.02\textcolor{blue}{5}\textcolor{red}{3} * 10^2$$

结果为2.36！精度没有2.37高！

$$2.36\textcolor{blue}{5}\textcolor{red}{3} * 10^2$$

IEEE Standard: 四种舍入方式

- 就近舍入到偶数
- 朝正无穷大方向舍入
- 朝负无穷大方向舍入
- 朝0方向舍入

就近舍入到偶数：

小数位 $< 1/2$ 截去

$> 1/2$ 加1

$= 1/2$ 最近的偶数，末位为0

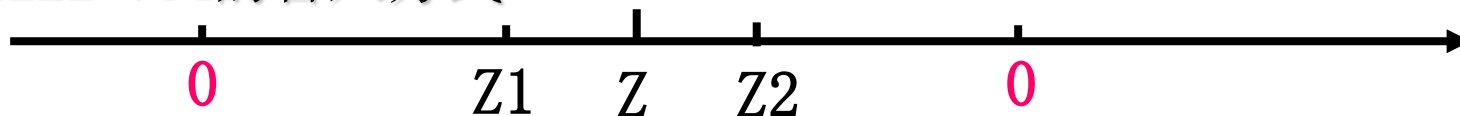
可以证明默认方式得到的平均误差最小。



IEEE 754的舍入方式的说明



IEEE 754的舍入方式



(Z1和Z2分别是结果Z的最近的可表示的左、右两个数)

(1) 就近舍入: 舍入为最近可表示的数

非中间值: 0舍1入;

中间值: 强迫结果为偶数-慢

例如: 附加位为

01: 舍

11: 入

10: (强迫结果为偶数)

例: $1.1101\textcolor{red}{11} \rightarrow 1.1110$; $1.1101\textcolor{red}{01} \rightarrow 1.1101$;

$1.1101\textcolor{red}{10} \rightarrow 1.1110$; $1.1111\textcolor{red}{10} \rightarrow 10.0000$;

(2) 朝 $+\infty$ 方向舍入: 舍入为Z2(正向舍入)

(3) 朝 $-\infty$ 方向舍入: 舍入为Z1(负向舍入)

(4) 朝0方向舍入: 截去。正数: 取Z1; 负数: 取Z2



浮点数舍入举例



例：将同一实数分别赋值给单精度和双精度类型变量，然后打印输出。

```
#include <stdio.h>
main()
{
    float a;
    double b;
    a = 123456.789e4;
    b = 123456.789e4;
    printf( "%f/n%f/n" , a, b);
}
```

运行结果如下：

```
1234567936.000000
1234567890.000000
```

为什么float情况下输出的结果会比原来的大？这到底有没有根本性原因还是随机发生的？为什么会出现这样的情况？

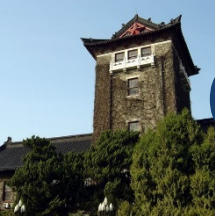
问题：为什么同一个实数赋值给float型变量和double型变量，输出结果会有所不同呢？



C语言中的浮点数类型



- C语言中有`float`和`double`类型，分别对应IEEE 754单精度浮点数格式和双精度浮点数格式
- `long double`类型的长度和格式随编译器和处理器类型的不同而有所不同，IA-32中是80位扩展精度格式
- 从`int`转换为`float`时，不会发生溢出，但可能有数据被舍入
- 从`int`或 `float`转换为`double`时，因为`double`的有效位数更多，故能保留精确值
- 从`double`转换为`float`和`int`时，可能发生溢出，此外，由于有效位数变少，故可能被舍入
- 从`float` 或`double`转换为`int`时，因为`int`没有小数部分，所以数据可能会向0方向被截断



Questions about IEEE 754



- What's the range of representable values?

The largest number for single: $+1.11...1 \times 2^{127}$

约 $+3.4 \times 10^{38}$

How about double?

约 $+1.8 \times 10^{308}$

- What about following type converting: not always true!

```
if ( i == (int) ((float) i) ) {  
    printf ("true");  
}
```

How about
double?

True!

```
if ( f == (float) ((int) f) ) {  
    printf ("true");  
}
```

How about
double?

Not always
true!

- How about FP add associative? FALSE!

$x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, $z = 1.0$

$(x+y)+z = (-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0 = 1.0$

$x+(y+z) = -1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0) = 0.0$



浮点运算举例



- 对于以下给定的关系表达式，判断是否永真。

```
int x ;  
float f ;  
double d ;
```

Assume neither
d nor f is NaN

- $x == (\text{int})(\text{float}) x$ 否
- $x == (\text{int})(\text{double}) x$ 是
- $f == (\text{float})(\text{double}) f$ 是
- $d == (\text{float}) d$ 否
- $f == -(-f);$ 是
- $2/3 == 2/3.0$ 否
- $d < 0.0 \Rightarrow ((d*2) < 0.0)$ 是
- $d > f \Rightarrow -f > -d$ 是
- $d * d \geq 0.0$ 是
- $(d+f)-d == f$ 否

自己写程序测试一下!



浮点运算举例



- 1996年6月4日，Ariana 5火箭初次航行，在发射仅仅37秒钟后，偏离了飞行路线，然后解体爆炸，火箭上载有价值5亿美元的通信卫星。
- 原因是**在将一个64位浮点数转换为16位带符号整数时，产生了溢出异常**。溢出的值是火箭的水平速率，这比原来的Ariana 4火箭所能达到的速率高出了5倍。在设计Ariana 4火箭软件时，设计者确认水平速率决不会超出一个16位的整数，但在设计Ariana 5时，他们没有重新检查这部分，而是直接使用了原来的设计。
- 在不同数据类型之间转换时，往往隐藏着一些不容易被察觉的错误，这种错误有时会带来重大损失，因此，编程时要非常小心。



浮点运算举例



- 1991年2月25日，海湾战争中，美国在沙特阿拉伯达摩地区设置的爱国者导弹拦截伊拉克的飞毛腿导弹失败，致使飞毛腿导弹击中了沙特阿拉伯载赫蓝的一个美军军营，杀死了美国陆军第十四军需分队的28名士兵。其原因是由爱国者导弹系统时钟内的一个软件错误造成的，引起这个软件错误的原因是浮点数的精度问题。
- 爱国者导弹系统中有一个内置时钟，用计数器实现，每隔0.1秒计数一次。程序用0.1的一个24位定点二进制小数x来乘以计数值作为以秒为单位的时间。
- 0.1的二进制表示是一个无限循环序列：0.00011[0011]...， $x=0.000\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\text{B}$ 。显然，x是0.1的近似表示， $0.1-x$

$$= 0.000\ 1100\ 1100\ 1100\ 1100\ 1100\ [1100]\cdots -$$

$$0.000\ 1100\ 1100\ 1100\ 1100\ 1100\text{B}, \text{ 即为:}$$

$$= 0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100\ [1100]\cdots\text{B}$$

$$= 2^{-20} \times 0.1 \approx 9.54 \times 10^{-8}$$

2020/6/29



浮点运算举例



已知在爱国者导弹准备拦截飞毛腿导弹之前，已经连续工作了100小时，飞毛腿的速度大约为2000米/秒，则由于时钟计算误差而导致的距离误差是多少？

100小时相当于计数了 $100 \times 60 \times 60 \times 10 = 36 \times 10^5$ 次，因而导弹的时钟已经偏差了 $9.54 \times 10^{-8} \times 36 \times 10^5 \approx 0.343$ 秒

因此，距离误差是 $2000 \times 0.343 \text{秒} \approx 687 \text{米}$

小故事：实际上，以色列方面已经发现了这个问题并于1991年2月11日知会了美国陆军及爱国者计划办公室（软件制造商）。以色列方面建议重新启动爱国者系统的电脑作为暂时解决方案，可是美国陆军方面却不知道每次需要间隔多少时间重新启动系统一次。1991年2月16日，制造商向美国陆军提供了更新软件，但这个软件最终却在飞毛腿导弹击中军营后的一天才运抵部队。



浮点运算举例



- 若x用float型表示，则x的机器数是什么？0.1与x的偏差是多少？系统运行100小时后的时钟偏差是多少？在飞毛腿速度为2000米/秒的情况下，预测的距离偏差为多少？
 - $0.1 = 0.0\ 0011[0011]B = +1.1\ 0011\ 0011\ 0011\ 0011\ 0011\ 00B \times 2^{-4}$ ，故x的机器数为0 011 1101 1 100 1100 1100 1100 1100 1100
 - Float型仅24位有效位数，后面的有效位全被截断，故x与0.1之间的误差为：
 $|x - 0.1| = 0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100\ [1100] \cdots B$ 。这个值等于 $2^{-24} \times 0.1 \approx 5.96 \times 10^{-9}$ 。100小时后时钟偏差 $5.96 \times 10^{-9} \times 36 \times 10^5 \approx 0.0215$ 秒。距离偏差 $0.0215 \times 2000 \approx 43$ 米。比爱国者导弹系统精确约16倍。
- 若用32位二进制定点小数 $x = 0.000\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101\ B$ 表示0.1，则误差比用float表示误差更大还是更小？
 - 当 $x = 0.000\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1101\ B$ 时，与0.1之间的误差约为：
 $|x - 0.1| = 0.000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 00\ 1100\ [1100] \cdots B$ 。这个值等于 $2^{-30} \times 0.1 \approx 9.31 \times 10^{-11}$ 。100小时后时钟偏差 $9.31 \times 10^{-11} \times 36 \times 10^5 \approx 0.000335$ 秒。预测的距离偏差仅为 $0.000335 \times 2000 \approx 0.67$ 米。



浮点运算举例



- 从上述结果可以看出：
 - 用32位定点小数表示0.1，比采用float精度高64倍
 - 用float表示在计算速度上更慢，必须先把计数值转换为IEEE 754格式浮点数，然后再对两个IEEE 754格式的数相乘，故采用float比直接将两个二进制数相乘要慢得多
- Ariana 5火箭和爱国者导弹的例子带来的启示
 - ✓ 程序员应对底层机器级数据的表示和运算有深刻理解
 - ✓ 计算机世界里，经常是“差之毫厘，失之千里”，需要细心再细心，精确再精确
 - ✓ 不能遇到小数就用浮点数表示，有些情况下（如需要将一个整数变量乘以一个确定的小数常量），可先用一个确定的定点整数与整数变量相乘，然后再通过移位运算来确定小数点