

第2章 信息的表示和处理Ⅱ：浮点数

教师：史先俊
计算机科学与技术学院
哈尔滨工业大学

主要内容

- 二进制小数
- IEEE 浮点数标准: IEEE 754
- 舍入模式
- 浮点数运算
- C语言的浮点数

推荐阅读:Ch2.4

有理数编码

■ 浮点表示很有用

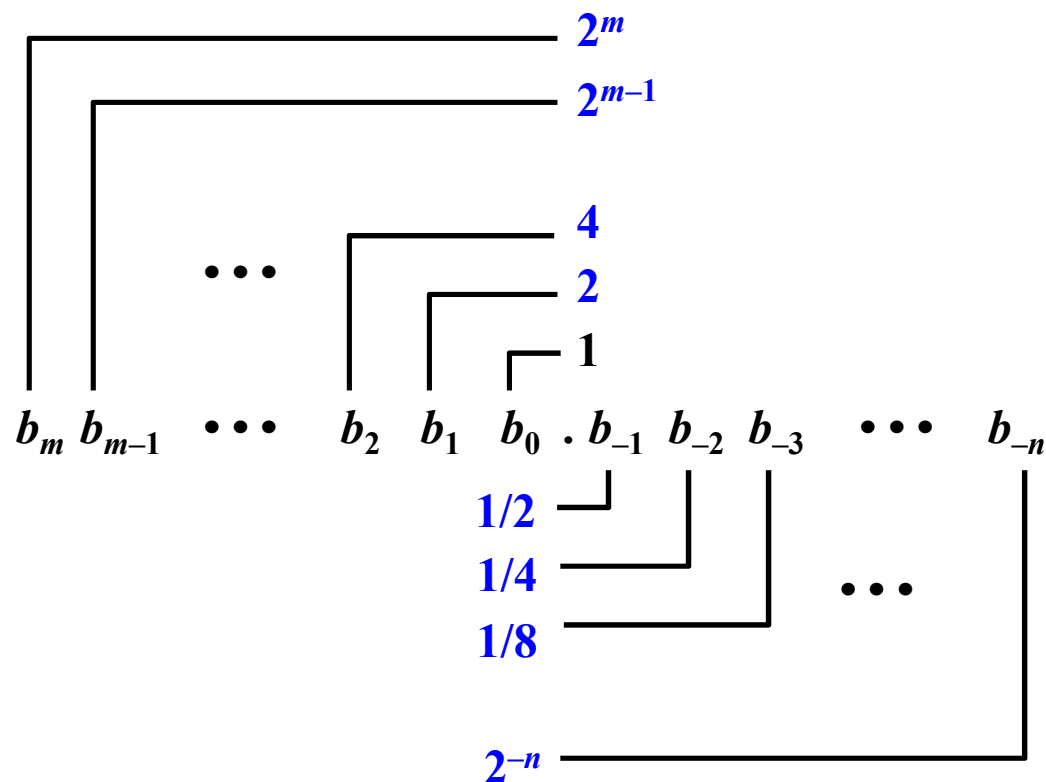
- 对形如 $V = x \times 2^y$ 的有理数进行编码
- 非常大的数 ($|V| \gg 0$) 或非常接近0的数 ($|V| \ll 1$)
- 实数的近似值

■ 从程序员角度看

- 无趣
- 晦涩难懂

二进制小数

- “小数点” 右边的位代表小数部分



- 表示的有理数: $\sum_{i=-n}^m b_i \times 2^i$

二进制小数: 例子

数值	二进制小数
$5 \frac{3}{4}$	101.11_2
$2 \frac{7}{8}$	10.111_2
$1 \frac{7}{16}$	1.0111_2

■ 观察

- 除以2 → 右移 (无符号数)
- 乘以2 → 左移
- $0.111111..._2$
 - $1/2 + 1/4 + 1/8 + ... + 1/2^i + ... \rightarrow 1.0$
 - 是最接近1.0的小数
 - 表示为 $1.0 - \varepsilon$

二进制数的问题

■ 局限性 1——近似表示

- 只能精确表示形如 $x/2^k$ 的数值
- 其他有理数的二进制表示存在重复段

- 数值 二进制表示

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

二进制数的问题

■ 在计算机内的实现问题

- 长度有限的 w 位
- 只能在 w 位内设置一个二进制小数点
- 限制了数的范围(非常小? 非常大?)

■ 定点数

- 小数点隐含在 w 位编码的某一个固定位置上
 - 例如MSB做符号位, 隐含后面是小数点, 表示小于1.0的纯小数
 - 123.456怎么办? ? ?

浮点数

- 二进制小数
- **IEEE 浮点数标准: IEEE 754**
- 浮点数示例与性质
- 舍入、加法与乘法
- C语言的浮点数
- 小结

IEEE 浮点数

■ IEEE 标准 754

- William Kahan 从1976年开始为Intel 设计(1989获图灵奖)
- 1985年成为浮点运算的统一标准，快速, 易于实现、精度损失小
- 优雅、易理解
- 所有主流的CPU都支持
- 之前有很多不同格式、不太关注精确性



浮点数的表示

■ 表示有理数的形式:

$$(-1)^s M 2^E$$

- 符号(sign) s , 决定数的符号, 是正数($s=0$)或负数($s=1$)
- 尾数(Significand) M , 二进制小数, 数值范围: $[1.0, 2.0)$
- 阶码(Exponent) E , 用 2^E 将数值加权

■ 浮点数编码

- 最高有效位(MSB) s , 作为符号位 s
- exp 字段 编码 E (和 E 不一定相等)
- frac 字段编码尾数 M (和 M 不一定相等)



精度选项

■ 单精度: 32 bits



■ 双精度: 64 bits



■ 扩展精度: 80 bits (Intel)



阶码

			IEEE754	IEEE754	
真值	补码	移码+128	移码+127	机器数	
127	01111111	11111111	11111110	254	
126	01111110	11111110	11111101	253	
.....	
2	00000010	10000010	10000001	129	
1	00000001	10000001	10000000	128	
0	00000000	10000000	'01111111	127	
-1	11111111	'01111111	' 01111110	126	
-2	11111110	' 01111110	'01111101	125	
.....	
-126	10000010	'00000010	' 00000001	1	
-127	10000001	' 00000001	'00000000	0	无此表示
-128	10000000	'00000000	11111111	-1	无此表示

规格化数

$$v = (-1)^s M 2^E$$

- 条件: $\text{exp} \neq 000\dots 0$ 且 $\text{exp} \neq 111\dots 1$
- 阶码(Exponent) 采用偏置值编码: $E = \text{Exp} - \text{Bias}$
 - Exp : exp 字段的无符号数值
 - 偏置 $\text{Bias} = 2^{k-1} - 1$, k 为阶码的位数
 - 单精度: 127 (Exp : 1...254, E : -126...127)
 - 双精度: 1023 (Exp : 1...2046, E : -1022...1023)
- 尾数(Significand) 编码隐含先导数值1: $M = 1.\text{xxx}\dots\text{x}_2$
 - $\text{xxx}\dots\text{x}$: 是 frac 字段的数码
 - $\text{frac}=000\dots 0$ ($M = 1.0$)时, 为最小值
 - $\text{frac}=111\dots 1$ ($M = 2.0 - \epsilon$)时, 为最大值
 - 额外增加了一位的精度 (隐含值1)

规格化编码示例

$$v = (-1)^s M 2^E$$

$$E = \text{Exp} - \text{Bias}$$

■ 数值: float $F = 15213.0$

$$15213_{10} = 11101101101101_2$$

$$= 1.1101101101101_2 \times 2^{13}$$

■ 尾数(Significand)

$$M = 1.\underline{1101101101101}_2$$

$$\text{frac} = \underline{1101101101101}0000000000_2$$

■ 阶码(Exponent)

$$E = 13$$

$$\text{Bias} = 127$$

$$\text{Exp} = 140 = 10001100_2$$

■ 编码结果:

0	10001100	110110110110100000000000
s	exp	frac

非规格化数

$$v = (-1)^s M 2^E$$

$$E = 1 - Bias$$

■ 条件: $\text{exp} = 000\dots 0$

- 阶码(Exponent) 值: $E = 1 - Bias = -126/-1022$ (instead of $E = 0 - Bias$)
- 尾数(Significand)编码隐含先导数值0: $M = 0.\text{xxx}\dots\text{x}_2$
 - $\text{xxx}\dots\text{x}$: 是 frac字段的数码

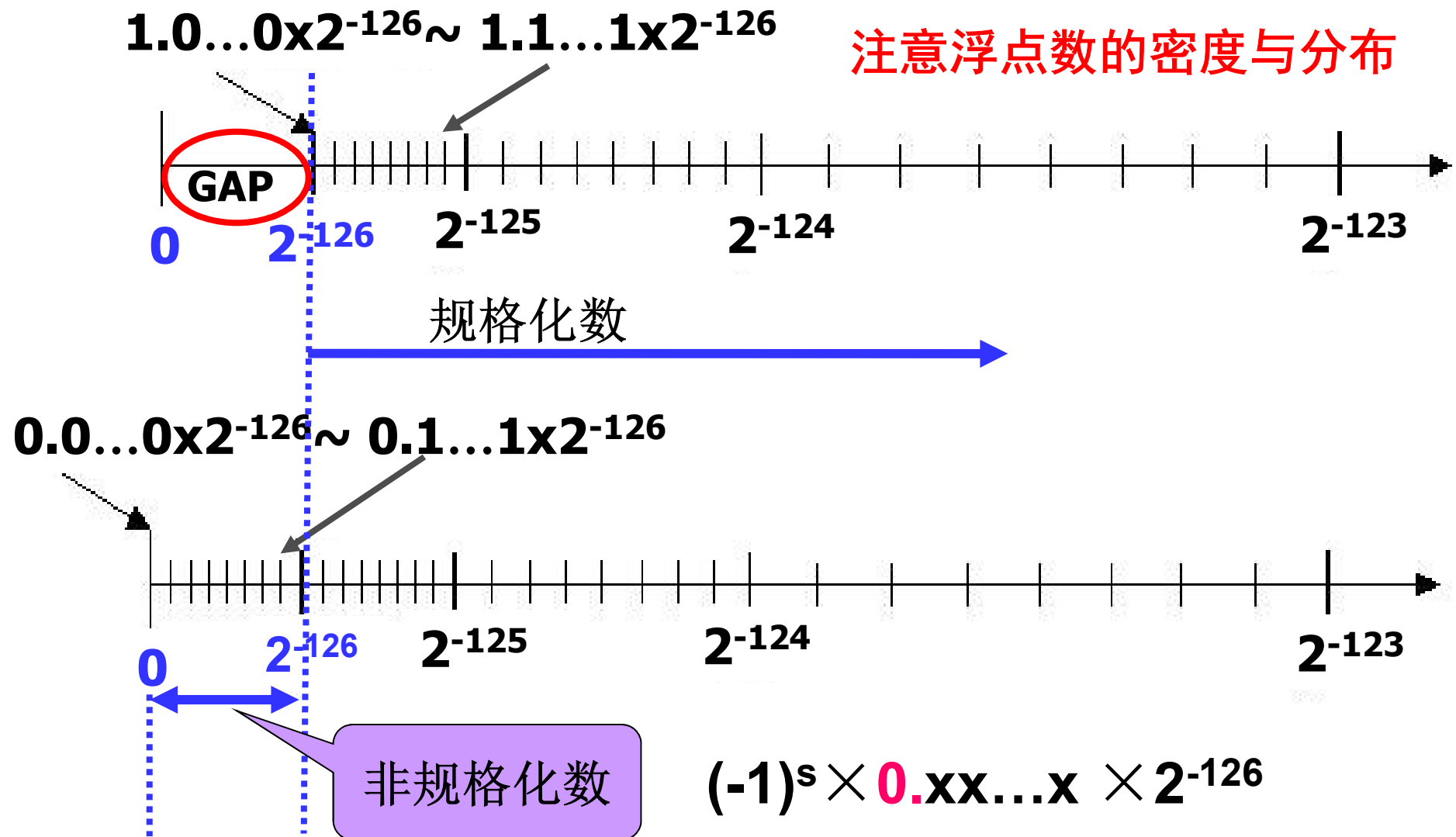
■ 情况1: $\text{exp} = 000\dots 0, \text{frac} = 000\dots 0$

- 表示值0
- 注意有不同的数值 +0 和 -0 (why?)

■ 情况2: $\text{exp} = 000\dots 0, \text{frac} \neq 000\dots 0$

- 最接近0.0的那些数
- 间隔均匀

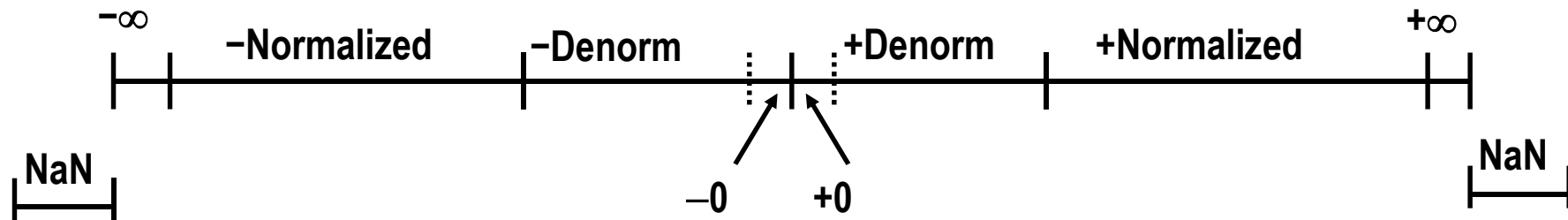
非规格化数据



特殊值

- 条件: **exp = 111...1**
- 情况1: **exp = 111...1, frac = 000...0**
 - 表示 无穷(infinity) ∞
 - 溢出的运算
 - 正无穷、负无穷
 - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
- 情况2: **exp = 111...1, frac \neq 000...0**
 - 表示: 不是一个数 Not-a-Number (NaN)
 - 表示没有数值结果 (实数或无穷), 例如:
 $\text{sqrt}(-1)$, $\infty - \infty$, $\infty \times 0$

浮点编码总结



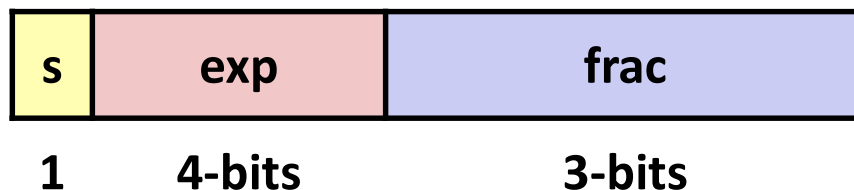
IEEE754 规格化浮点数表示范围

格式	最小值	最大值
单精度规格化 $(-1)^s \times 1.m \times 2^{e-127}$	$E_{\min}=1, M=0,$ $1.0 \times 2^{1-127} = 2^{-126}$	$E_{\max}=254, f=1.1111\cdots 1 \times 2^{254-127}$ $= 2^{127} \times (2-2^{-23})$ 约 $+3.4 \times 10^{38}$
单精度非规格化 $(-1)^s \times 0.m \times 2^{-126}$	$E=0, M=2^{-23},$ $2^{-23} \times 2^{-126} = 2^{-149}$	$E=0, f=0.1111\cdots 1 \times 2^{-126}$ $= 2^{-126} \times (1-2^{-23})$
双精度规格化 $(-1)^s \times 1.m \times 2^{e-1023}$	$E_{\min}=1, M=0,$ $1.0 \times 2^{1-1023} = 2^{-1022}$	$E_{\max}=2046,$ $f=1.1111\cdots 1 \times 2^{2046-1023}$ $= 2^{1023} \times (2-2^{-52})$ 约 $+1.8 \times 10^{308}$
双精度非规格化 $(-1)^s \times 0.m \times 2^{-1022}$	$E=0, M=2^{-52},$ $2^{-52} \times 2^{-1022} = 2^{-1079}$	$E=0, f=0.11111\cdots 1 \times 2^{-1022}$ $= 2^{-1022} \times (1-2^{-52})$

浮点数

- 二进制小数
- IEEE 浮点数标准: IEEE 754
- **浮点数示例与性质**
- 舍入、加法与乘法
- C语言的浮点数
- 小结

小浮点数例子——1字节浮点数



■ 8位浮点编码

- 符号位：最高有效位
- 阶码(Exponent)4位，偏置为7
- 小数(frac) 3位

■ 和IEEE 相同的格式

- 规格化、非规格化
- 0、NaN、无穷的表达

动态范围(仅正数)

	s exp frac	E	Value	
非规格化数	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	$8/8 * 1/64 = 8/512$	
规格化数	0 0001 001	-6	$9/8 * 1/64 = 9/512$	
	...			
	0 0110 110	-1	$14/8 * 1/2 = 14/16$	
	0 0110 111	-1	$15/8 * 1/2 = 15/16$	closest to 1 below
	0 0111 000	0	$8/8 * 1 = 1$	
	0 0111 001	0	$9/8 * 1 = 9/8$	closest to 1 above
	0 0111 010	0	$10/8 * 1 = 10/8$	
	...			
	0 1110 110	7	$14/8 * 128 = 224$	
	0 1110 111	7	$15/8 * 128 = 240$	最大规格化数
	0 1111 000	n/a	inf	

$$v = (-1)^s M 2^E$$

$n: E = Exp - Bias$

$d: E = 1 - Bias$

最接近0

最大非规格化数

最小规格化数

closest to 1 below

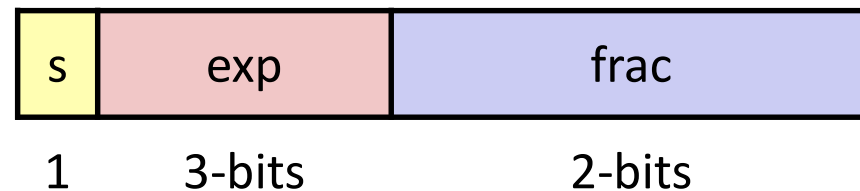
closest to 1 above

最大规格化数

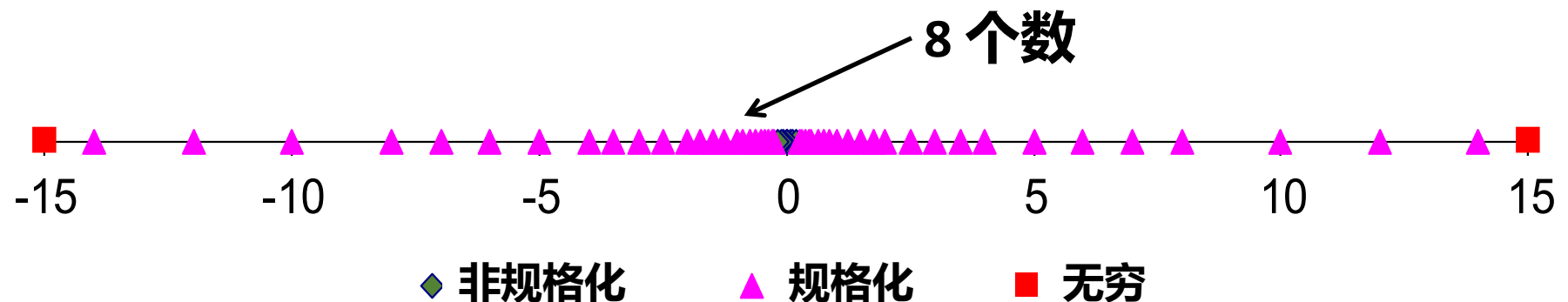
数值分布

■ 6-bit类 IEEE格式浮点数

- e : 阶码(Exponent) 位数3
- f : 小数位数 2
- 偏置bias= $2^{3-1}-1 = 3$



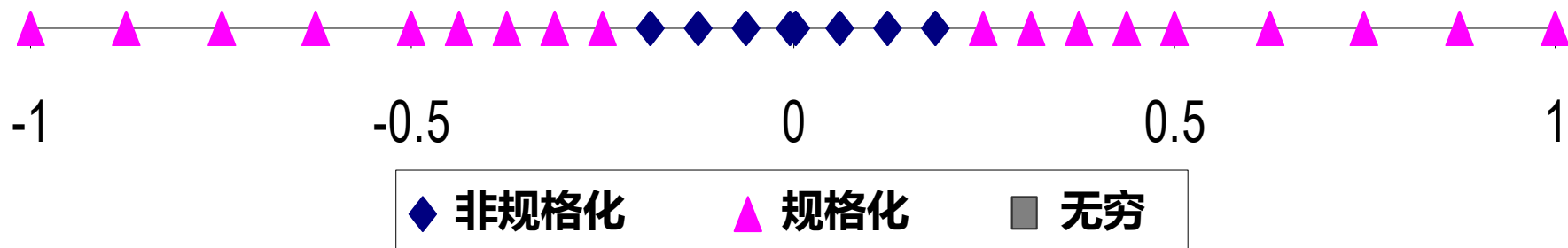
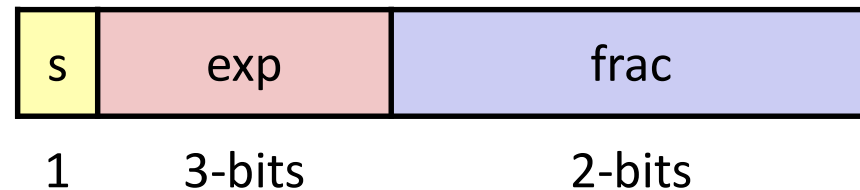
■ 注意：数值在趋近于0时变密集



数值分布(放大观察)

■ 6-bit类 IEEE格式

- e : 阶码(Exponent) 位数3
- f : 小数位数 2
- 偏置bias= $2^{3-1}-1 = 3$



IEEE编码的特殊性质

- 浮点0与整数0编码相同：所有bit均为0
- 几乎可以用与无符号整数相同的方式进行浮点数的比较
 - 先比较符号位
 - 必须考虑 $-0 = 0$
 - NaN的不确定性
 - 将比其他任何值都大
 - 比较将产生什么结果？
 - 其他方面均OK
 - 规格化值 vs. 非规格化值
 - 规格化值 vs. 无穷

浮点数

- 二进制小数
- IEEE 浮点数标准: IEEE 754
- 浮点数示例与性质
- 舍入、加法与乘法
- C语言的浮点数
- 小结

浮点数运算: 基本思想

- $x +_f y = \text{Round}(x + y)$

- $x \times_f y = \text{Round}(x \times y)$

- 基本思想

- 首先, 计算精确结果
- 然后, 变换到指定格式
 - 可能溢出: 阶码(Exponent) 太大
 - 小数部分可能需要舍入

舍入

■ 舍入模式(以美元舍入说明)

■	\$1.40	\$1.60	\$1.50	\$2.50	-\$1.50
■ 向0舍入	\$1	\$1	\$1	\$2	-\$1
■ 向下舍入 ($-\infty$)	\$1	\$1	\$1	\$2	-\$2
■ 向上 ($+\infty$)	\$2	\$2	\$2	\$3	-\$1
■ 向偶数舍入(默认)	\$1	\$2	\$2	\$2	-\$2

细究“向偶数舍入”

■ 默认的舍入模式

- 很难找到更好的方法
- 其他方法都有统计偏差
 - 对正整数集合求和时，和将始终被低估或高估（负偏差、正偏差）

■ 向偶数舍入

- 当恰好在两个可能的数值正中间时（中间值）：
舍入后，最低有效位的数码为偶数
- 其他时候：向最近的数值舍入
 - 比中间值小向下舍入，比中间值大向上舍入

■ 以10进制数向最近的百分位舍入为例：

7.8949999	7.89	(比中间值小：向下舍入)
7.8950001	7.90	(比中间值大：向上舍入)
7.8950000	7.90	(中间值—向上舍入)
7.8850000	7.88	(中间值—向下舍入)

二进制数的舍入

■ 二进制小数的舍入

- “偶数”：最低有效位值为0
- “中间值”：舍入位置右侧的位都是0，即形如：XXX 100...₂

■ 例子

- 舍入到最近的1/4 (小数点右边第2位)

数值	二进制	舍入后	舍入动作	舍入后的值
2 3/32	10.00 <u>01</u> ₂	10.00 ₂	(<1/2—down)	2
2 3/16	10.00 <u>11</u> ₂	10.01 ₂	(>1/2—up)	2 1/4
2 7/8	10.11 <u>10</u> ₂	11.00 ₂	(1/2—up)	3
2 5/8	10.10 <u>10</u> ₂	10.10 ₂	(1/2—down)	2 1/2

浮点乘法

- $(-1)^{s1} M1 2^{E1} \times (-1)^{s2} M2 2^{E2}$
- 精确结果: $(-1)^s M 2^E$
 - 符号(Sign) s : $s1 \wedge s2$
 - 尾数(Significand) M : $M1 \times M2$
 - 阶码(Exponent) E : $E1 + E2$
- 修正
 - 如 $M \geq 2$, 将 M 右移(1位), E 加1
 - 如 E 超出范围, 则溢出
 - 将 M 舍入, 以符合小数部分的精度要求
- 实现
 - 主要问题: 实现尾数(Significand)的乘

浮点数加法

$$\blacksquare (-1)^{s1} M1 2^{E1} + (-1)^{s2} M2 2^{E2}$$

▪ 假设 $E1 > E2$

$$\blacksquare \text{准确结果: } (-1)^s M 2^E$$

▪ 符号 s , 尾数 M :

▪ 有符号数对齐、相加的结果

▪ 阶码(Exponent) E : $E1$

修正

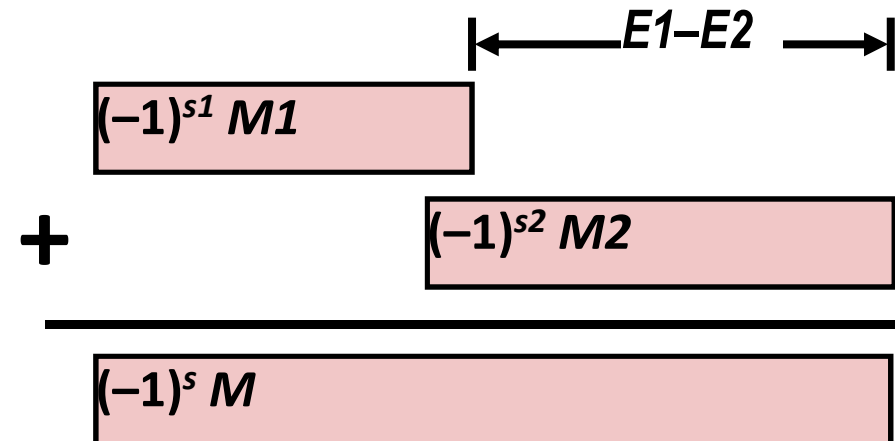
▪ $M \geq 2$: 将 M 右移(1位), E 加 1

▪ $M < 1$: 将 M 左移 k 位, E 减 k

▪ E 超范围: 溢出

▪ 将 M 舍入, 以符合小数部分的精度要求

二进制小数点对齐



浮点数加法的数学性质

■ 与阿贝尔群比较

加法运算下:

- 是否封闭

Yes

- But may generate infinity or NaN

- 交换性(Commutative)?

Yes

- 分配性(Associative)?

No

- 溢出和舍入的不确定性

- $(3.14+1e10)-1e10 = 0$, $3.14+(1e10-1e10) = 3.14$

- 0 是加法的单位元?

- 每个元素都有逆元?

Yes

- 除了无穷和NaN

Almost

■ 单调性(Monotonicity)

- $a \geq b \Rightarrow a+c \geq b+c$?

Almost

- 除了无穷和NaN

浮点数乘法的数学性质

■ 与交换环相比

- 乘法下封闭性? **Yes**
 - 但可能产生无穷或NaN
- 乘法的交换性? **Yes**
- 乘法的结合性? **No**
 - 可能溢出、舍入不精确
 - 例: $(1e20 * 1e20) * 1e-20 = \text{inf}$, $1e20 * (1e20 * 1e-20) = 1e20$
- 1 是乘法的单位元? **Yes**
- 乘法对加法的分配性? **No**
 - 可能溢出、舍入不精确
 - $1e20 * (1e20 - 1e20) = 0.0$, $1e20 * 1e20 - 1e20 * 1e20 = \text{NaN}$

■ 单调性

- $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$? **Almost**
 - 除了无穷和 NaN

浮点数

- 二进制小数
- IEEE 浮点数标准: IEEE 754
- 浮点数示例与性质
- 舍入、加法与乘法
- **C语言的浮点数**
- 小结

C语言的浮点数

■ 两种精度

- `float` 单精度
- `double` 双精度

■ 类型转换

- `int, float, double` 间转换, 将改变位模式
- `double/float → int`
 - 截掉小数部分
 - 类似向0舍入
 - 当数值超范围或NaN时无定义: 通常设置为 TMin
- `int → double`
 - 精确转换, 只要 `int` 的位宽 ≤ 53 bit, 即可精确转换
- `int → float`
 - 将根据舍入模式进行舍入

浮点数习题

■ 针对下列C表达式:

- 证明对所有参数值都成立
- 或什么条件下不成立

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

假定d 和 f都不是NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (double)(float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0` \Rightarrow `((d*2) < 0.0)`
- `d > f` \Rightarrow `-f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

浮点数习题答案

- $x == (\text{int})(\text{float})\ x$ No: 24 位尾数
- $x == (\text{int})(\text{double})\ x$ Yes: 53位尾数
- $f == (\text{float})(\text{double})\ f$ Yes: 增加精度
- $d == (\text{float})\ d$ No: 损失精度
- $f == -(-f);$ Yes: 仅仅改变符号位
- $2/3 == 2/3.0$ No: $2/3 == 0$
- $d < 0.0 \Rightarrow ((d*2) < 0.0)$ Yes!
- $d > f \Rightarrow -f < -d$ Yes
- $d * d \geq 0.0$ Yes!
- $(d+f)-d == f$ No: 不具备结合性

浮点的悲剧

- 1991年2月25日
- 美国爱国者导弹拦截伊拉克飞毛腿导弹失败
- 后果：飞毛腿导弹炸死**28**名士兵
- 爱国者导弹的内置时钟计数器**N**每**0.1**秒记一次数。
- 时间计算

$$T = N \times 0.1$$

程序用**24**位数来近似表示**0.1**:

x=0.0001 1001 1001 1001 1001 100

浮点的悲剧

- $0.1-x = 0.\text{000000000000000000000000}[1100][1100]..._2$
- $0.1-x = 2^{-20} \times 0.1 = 9.54 \times 10^{-8}$
- 程序运行100 后，累计的误差：
 $100 \times 3600 \times 10 \times 9.54 \times 10^{-8} = 0.34344$ 秒
- 软件升级不完全，第一次读取了精确时间，而另一次读取了有误差的时间，结果悲剧....
- 飞毛腿速度：2000 m/s
- 飞毛腿位置的估计误差：686 m

天价“溢出”

- 代价5亿美元的溢出



天价“溢出”

- 主角：阿丽亚娜5(Ariane5)型火箭的首次发射
- 时间：1996.6.4
- 剧情：发射后仅37秒，偏离路径，解体爆炸
- 代价：5亿美元
- 原因：溢出
 - 溢出——将64位浮点数转换成16位有符号整型数时，发生溢出。这个溢出的整型数，用于描述火箭的水平速度
 - Ariane4的水平速度绝对不会超过16位数的范围，因此用了16位整数
 - Ariane5简单复用了这部分代码
 - 问题：Ariane 5 的水平速度是Ariane 4的5倍！！！！

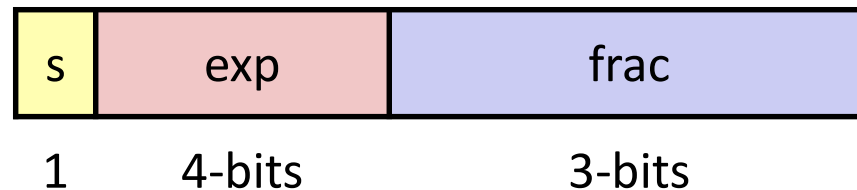
小结

- IEEE 浮点数 具有清晰的数学性质
- 表示形如 $M \times 2^E$ 的数字
- 对运算进行推理，而不用考虑其实现
 - 就像有完美的精度，然后在进行舍入
- 和实数运算不同
 - 结合性、分配性有冲突
 - 日子变得难：编译器、认真的数值应用程序员

生成浮点数

■ 步骤

- 规格化为1开头的数
- 小数部分舍入成符合的形式
- 后规格化，处理舍入的效果

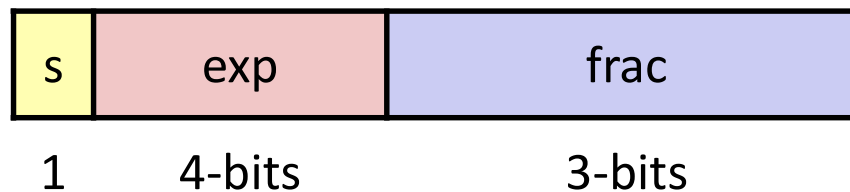


■ 例子

- 将 8-bit 无符号数转换成小浮点格式

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010
63	00111111

规格化



■ 要求

- 调整编码的所有参数，得到1开始的数，即形如1.xxxxx的数字
- 指数减作为左移

数值	二进制	小数	指数
128	10000000	1.0000000	7
15	00001101	1.1010000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

舍入

1 . BB**G****R**XXX

保护位(Guard bit): 结果的LSB

黏着位(Sticky bit): 剩余位

舍入位(Round bit): 舍入位中的第一个bit

■ 舍入的条件

- Round = 1, Sticky = 1 \rightarrow > 0.5
- Guard = 1, Round = 1, Sticky = 0 \rightarrow Round to even

数值	小数	<i>GRS</i>	<i>Incr?</i>	舍入后的值
128	1.000 0000	000	N	1.000
15	1.101 0000	100	N	1.101
17	1.000 1000	010	N	1.000
19	1.001 1000	110	Y	1.010
138	1.000 1010	011	Y	1.001
63	1.111 1100	111	Y	10.000

后规格化

■ 问题

- 舍入可能导致溢出
- 解决：单次右移 & 阶码(Exponent)增1

数值	舍入后的值	指数值	修正	结果
128	1.000	7		128
15	1.101	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000/6	64

有趣的数字

{single, double}

<i>Description</i>	<i>exp</i>	<i>frac</i>	<i>Numeric Value</i>
■ 0	00...00	00...00	0.0
■ 最小值的后非规格化数	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
<ul style="list-style-type: none"> ■ Single $\approx 1.4 \times 10^{-45}$ ■ Double $\approx 4.9 \times 10^{-324}$ 			
■ 最大的非规格化数	00...00	11...11	$(1.0 - \epsilon) \times 2^{-\{126,1022\}}$
<ul style="list-style-type: none"> ■ Single $\approx 1.18 \times 10^{-38}$ ■ Double $\approx 2.2 \times 10^{-308}$ 			
■ 最小的后规格化数	00...01	00...00	$1.0 \times 2^{-\{126,1022\}}$
<ul style="list-style-type: none"> ■ 刚刚比最大的非规格化数大 			
■ 1	01...11	00...00	1.0
■ 最大的规格化数	11...10	11...11	$(2.0 - \epsilon) \times 2^{\{127,1023\}}$
<ul style="list-style-type: none"> ■ Single $\approx 3.4 \times 10^{38}$ ■ Double $\approx 1.8 \times 10^{308}$ 			