第二章 信息的表示和处理Ⅱ：浮点数

本章重点

- **IEEE 浮点数标准**
  - 理解规格化数和非规格化数，无穷的表示，NAN等知识
  - 浮点数在数轴上的分布
  - 可以熟练地完成实数与浮点数之间的相互转换
- **浮点数的舍入原则**
  - 浮点数的四种舍入原则
- **浮点数的乘法、加法**
- **经典例题**

本章目录

- 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

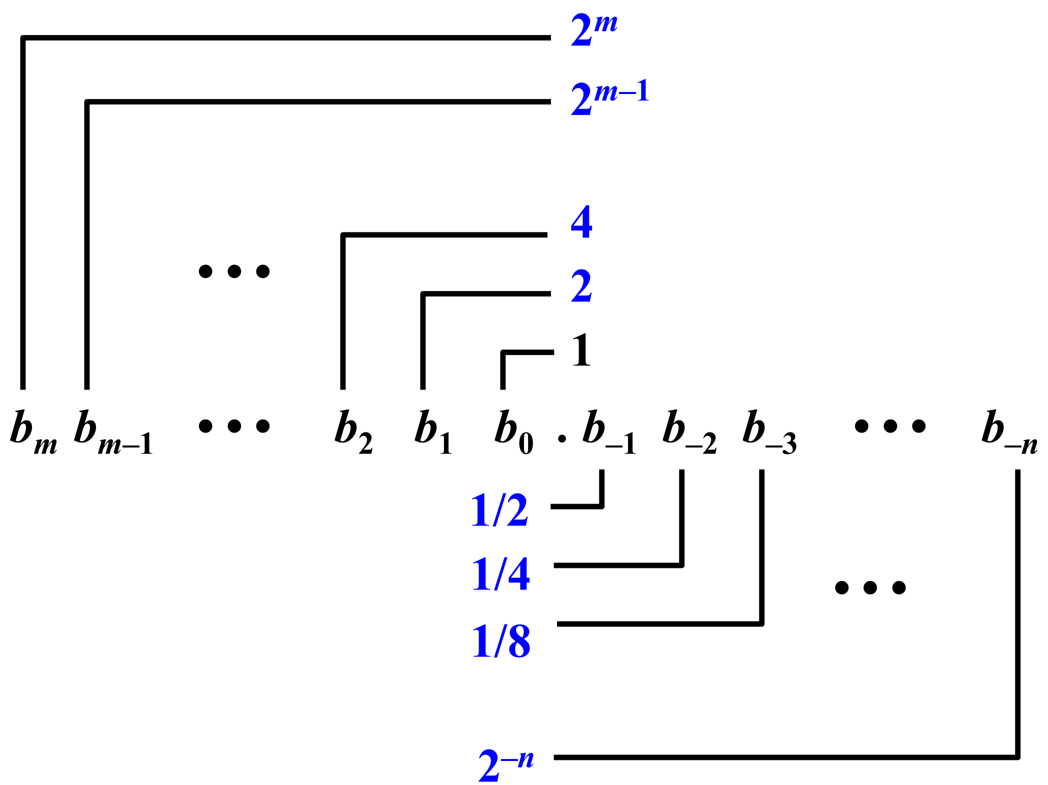
有理数编码

- **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
  - 无穷大/小怎么办？

- **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
  - $0.111111\ldots_2$
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - 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，具有快速、易于实现、精度损失小等特点
  - 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

（回顾计组）IEEE 754浮点数：单精度为例

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
s	8位指数（无符号数）								23位尾数（无符号数）						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
23位尾数（无符号数）															

指数	尾数	表示对象	换算方法
0	0	0	**规定** （符号位不同，存在+0.0和-0.0）
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S * (\text{尾数}_2) * 2^{(0-126)}$ （S代表符号位，1为负数，0为正数）
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数}-127)}$
255	0	正负无穷 (inf)	**规定**
255	非零	NaN	**规定**

（回顾计组）IEEE 754浮点数：真值转二进制

- **例题**
  - 将十进制 -0.75 转为单精度 IEEE 754格式二进制
- **解**

根据十进制小数转二进制小数算法：$-0.75_{10} = -0.11_2$

$-0.11 = -1.1 * 2^{-1}$，说明可以隐藏1，属于正负浮点数表示

符号位：1；

（IEEE754）指数部分：（-1+127）$_{10}$=（126）$_{10}$=（01111110）$_2$；

尾数部分：0.1_2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

十六进制：BF400000

（回顾计组）IEEE 754浮点数：二进制转真值

- **例题**
  - 将二进制IEEE754浮点数表示转换为十进制浮点数（空白为0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **解**

符号位为1，（IEEE754）指数字段为129，尾数字段为2^{-2} = 0.25。浮点数为：

$$(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)} = (-1)^1 * (1 + 0.25) * 2^{(129 - 127)}$$
$$= -1 * 1.25 * 2^2$$
$$= -5.0$$

浮点数的表示

- **表示有理数的形式:**

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

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

- **浮点数编码**
  - 最高有效位(MSB)s，作为符号位*s*
  - 编码字段exp， 作为阶码*E* (和E不一定相等)
  - 编码字段frac， 作为尾数 *M* (和M不一定相等)

s	exp	frac

精度选项

- 单精度: 32 bits

s	exp	frac
1	8-bits	23-bits

- 双精度: 64 bits

s	exp	frac
1	11-bits	52-bits

- 扩展精度: 80 bits (Intel)

s	exp	frac
1	15-bits	63 or 64-bits

（回顾计组）IEEE 754浮点数：单精度为例

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
s	8位指数（无符号数）								23位尾数（无符号数）						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
23位尾数（无符号数）															

指数	尾数	表示对象	换算方法
0	0	0	**规定** （符号位不同，存在+0.0和-0.0）
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S$ * **(尾数$_2$)** * $2^{(0-126)}$ （S代表符号位，1为负数，0为正数）
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S$ * (1 + **尾数$_2$**) * $2^{(\text{指数}-127)}$
255	0	正负无穷 (inf)	**规定**
255	非零	NaN	**规定**

浮点数的表示

单精度浮点数值的分类

1. 规格化的

s	≠0 && ≠255	f

2. 非规格化的

s	0000 0000	f

3a. 无穷大

s	1111 1111	000 0000 0000 0000 0000 0000

3b. NaN(Not a Number)

s	1111 1111	≠0

规格化数

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

- **条件: exp ≠ 000…0 且 exp ≠ 111…1**
- **阶码(Exponent) 采用偏置值编码: *Exp* = *E* + *Bias***
  - *Exp*: exp 字段的无符号数值
  - 偏置*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* = $1.xxx…x_2$**
  - xxx…x: 是 frac字段的数值编码
  - frac=000…0 (M = 1.0)时，为最小值
  - frac=111…1 (M = 2.0 – ε)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

$v = (-1)^s M\ 2^E$
$Exp = E + Bias$

- 数值: `float F = 15213.0`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$
- 尾数(Significand)
  M = $1.\underline{1101101101101}_2$
  `frac` = $\underline{1101101101101}0000000000_2$
- 阶码(Exponent)
  E = 13
  $Bias$ = 127
  Exp = 13+127 = 10001100_2
- 编码结果:

s	exp	frac
0	10001100	11011011011010000000000

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 = +∞, 1.0/−0.0 = −∞

- 情况2: `exp` = 111…1, `frac` ≠ 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：sqrt(–1), $\infty - \infty$, $\infty \times 0$

非规格化数

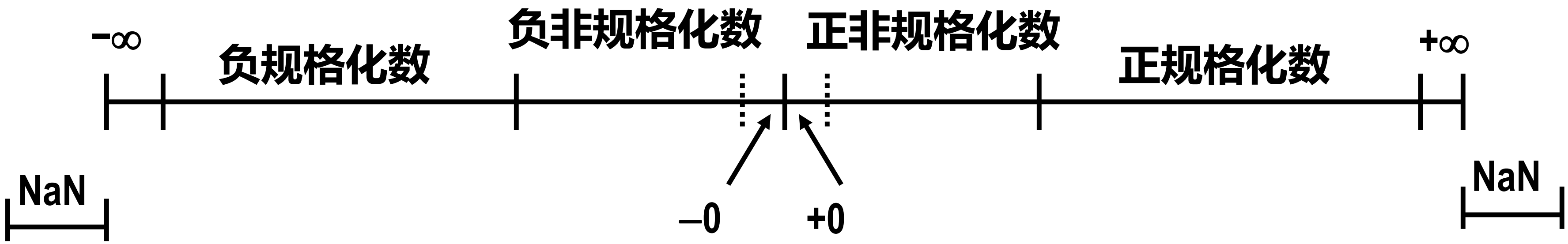
$v = (-1)^s M 2^E$
$E = 1 - Bias$

- **条件: exp = 000…0**
  - 阶码(Exponent) 值: $E = 1 - \text{Bias}$ =-126/-1022**(而不是$E = 0 - Bias$)**
  - 尾数(Significand)编码隐含先导数值0: $M = 0.xxx…x_2$
    - **xxx…x**:是 frac字段的数码

- **情况1: `exp` = 000…0, `frac` = 000…0**
  - 表示值0
  - 注意有不同的数值 +0 和 –0

- **情况2: `exp` = 000…0, `frac` ≠ 000…0**
  - 最接近0.0的那些数
  - 间隔均匀

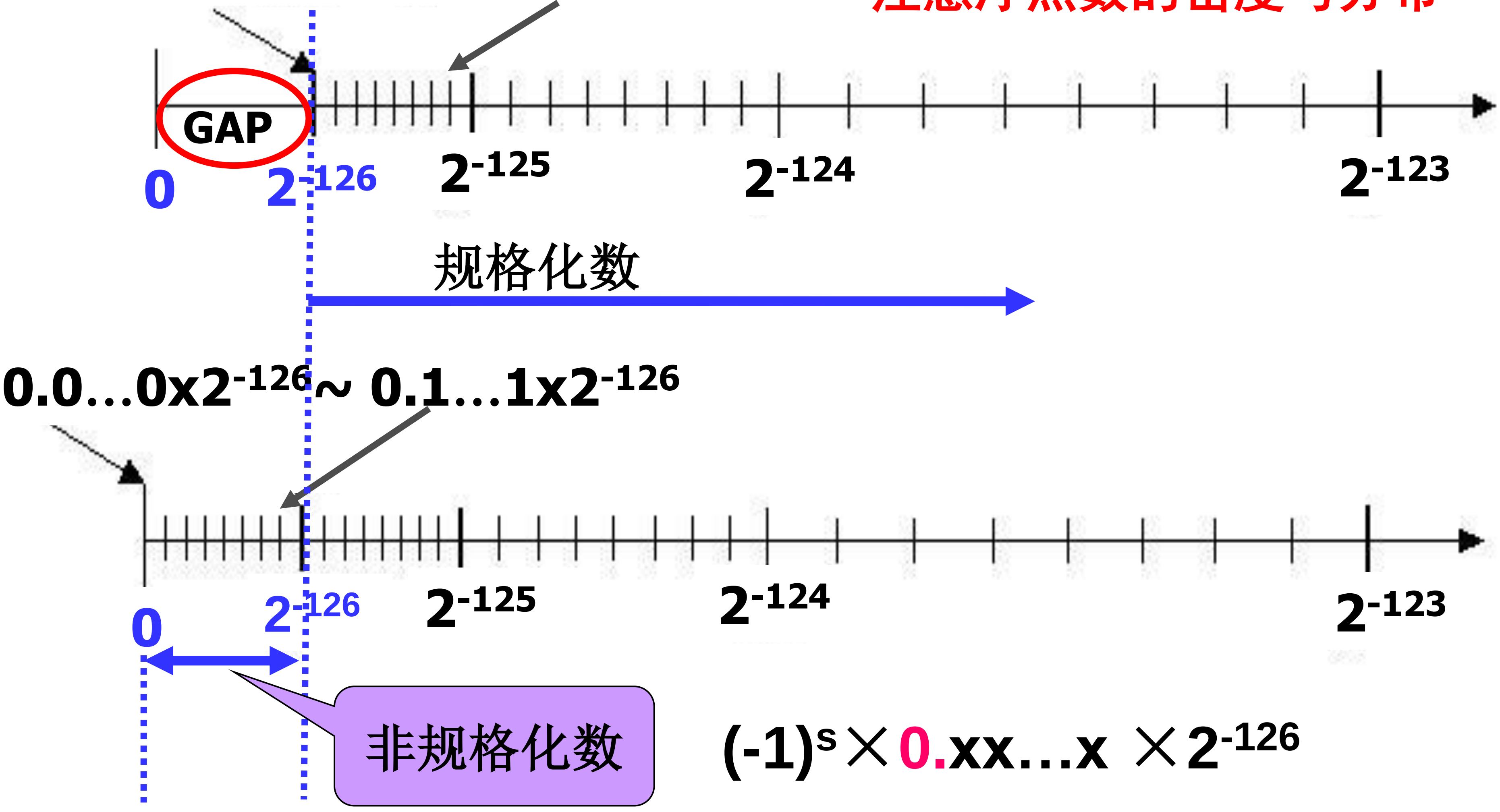
浮点编码总结



非规格化数据

$[1.0\ldots0\times2^{-126} \sim 1.1\ldots1\times2^{-126})$

注意浮点数的密度与分布



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

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

s	exp	frac
1	4-bits	3-bits

- **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、无穷的表示

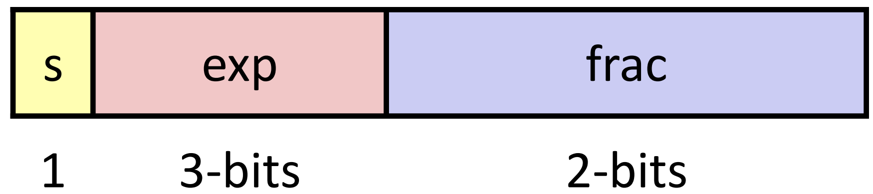
动态范围(仅正数)

$v = (-1)^s M\ 2^E$
规格化: $E = Exp - Bias$
非规格化: $E = 1 - Bias$

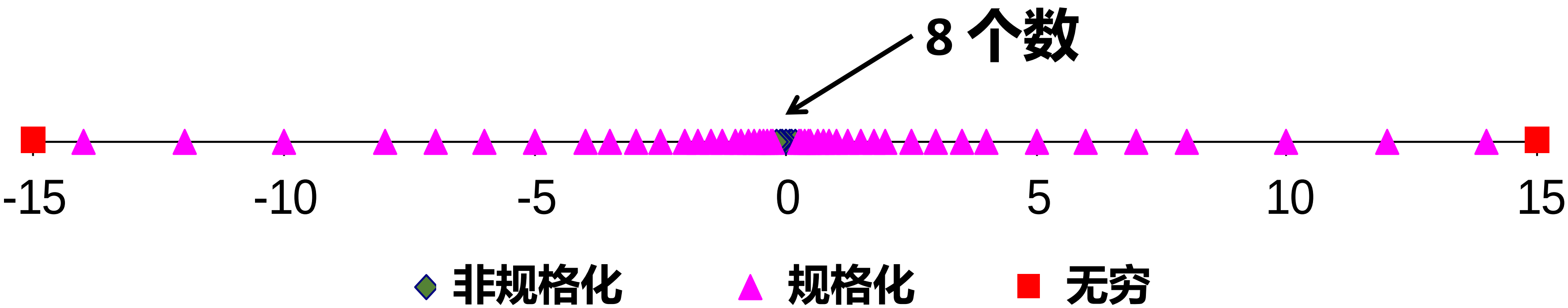
	s exp frac	E	Value	
非规格化数	0 0000 000	-6	0	
	0 0000 001	-6	1/8*1/64 = 1/512	最接近0
	0 0000 010	-6	2/8*1/64 = 2/512	
	…			
	0 0000 110	-6	6/8*1/64 = 6/512	
	0 0000 111	-6	7/8*1/64 = 7/512	最大非规格化数
规格化数	0 0001 000	-6	8/8*1/64 = 8/512	最小规格化数
	0 0001 001	-6	9/8*1/64 = 9/512	
	…			
	0 0110 110	-1	14/8*1/2 = 14/16	
	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	

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= $2^{3-1}-1 = 3$

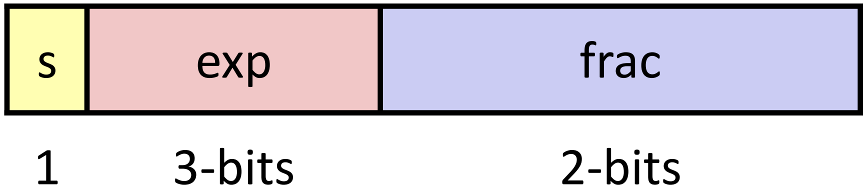


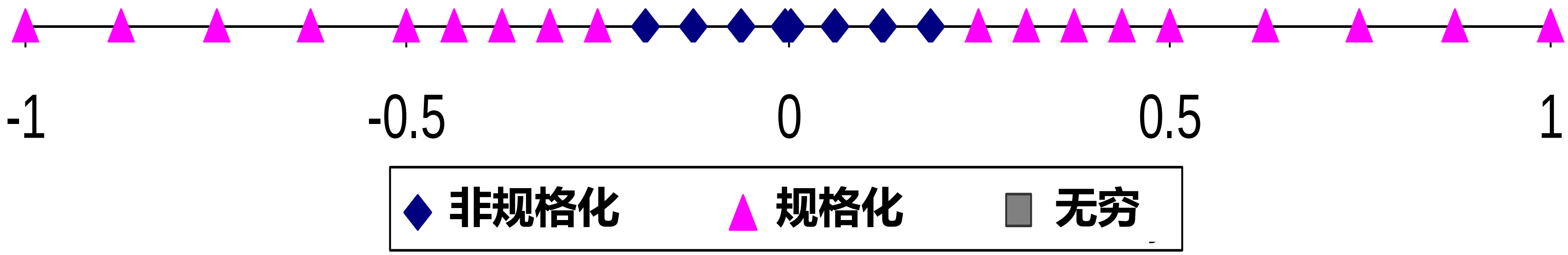
- **注意：数值在趋近于0时变密集**



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
- 浮点数比较：用“==” “>” “<” “>=” “<=“
  先比较符号位
  - 但必须注意 –0 = 0
  - NaN（Not a Number）的不确定性
    - 将比其他任何值都大
    - 比较将产生什么结果？**false**
  - 其他方面均OK
    - 规格化值 vs. 非规格化值
    - 规格化值 vs. 无穷
- 精度问题呢？ Float与Double比较呢？
- 怎么解决？讨论......................

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

- $x +^f y = \mathrm{Round}(x + y)$
- $x \times^f y = \mathrm{Round}(x \times y)$

（注：Round(x)表示对x舍入后的值）

- **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 (−∞)	\$1	\$1	\$1	\$2	–\$2
向上舍入 (+∞)	\$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
  - 其它时候：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7.895小：向下舍入)
7.8950001	7.90	(比中间值7.895大：向上舍入)
7.8950000	7.90	(等于中间值7.895且0为偶数，向上舍入)
7.8850000	7.88	(等于中间值7.885且8为偶数，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 **1**00…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	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

浮点乘法

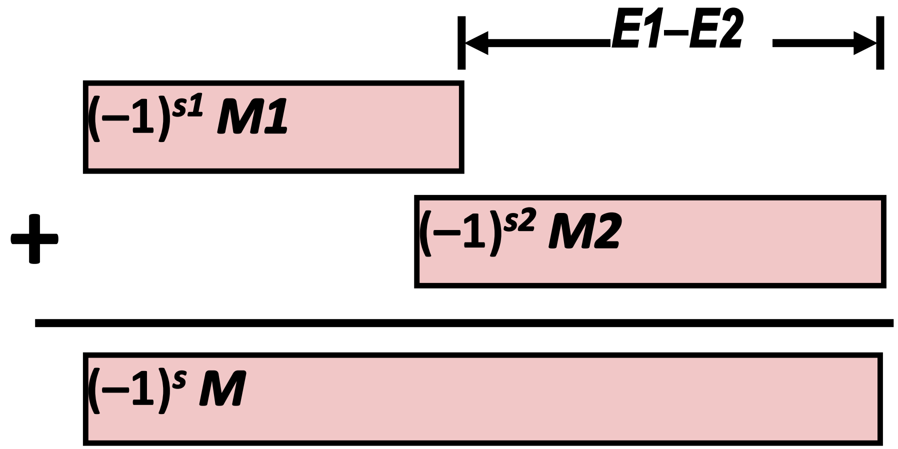
- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- **精确结果:** $(-1)^{s}\ M\ \ 2^{E}$
  - 符号(Sign) *s*: $s1 \wedge s2$
  - 尾数(Significand) *M*: $M1 \times M2$
  - 阶码(Exponent) *E*: $E1 + E2$

- **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

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
  - 假设 $E1 > E2$
  - 先对齐阶码，再相加尾数
- **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***
  - 结合性(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? ***Yes***
  - 每个元素都有逆元? ***Almost***
    - 除了无穷和NaN
- **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= 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 = 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

见教材86页，重要

当在 int、float 和 double 格式之间进行强制类型转换时，程序改变数值和位模式的原则如下(假设 int 是 32 位的)：

- 从 int 转换成 float，数字不会溢出，但是可能被舍入。
- 从 int 或 float 转换成 double，因为 double 有更大的范围(也就是可表示值的范围)，也有更高的精度(也就是有效位数)，所以能够保留精确的数值。
- 从 double 转换成 float，因为范围要小一些，所以值可能溢出成 $+\infty$ 或 $-\infty$。另外，由于精确度较小，它还可能被舍入。
- 从 float 或者 double 转换成 int，值将会向零舍入。例如，1.999 将被转换成 1，而 -1.999 将被转换成 -1。进一步来说，值可能会溢出。C 语言标准没有对这种情况指定固定的结果。与 Intel 兼容的微处理器指定位模式[10…00](字长为 w 时的 $TMin_w$)为整数不确定(integer indefinite)值。一个从浮点数到整数的转换，如果不能为该浮点数找到一个合理的整数近似值，就会产生这样一个值。因此，表达式 (int)+1e10 会得到-21483648，即从一个正值变成了一个负值。

见考研书，重要

6．C 语言中的浮点数类型及类型转换

C 语言中的 float 和 double 类型分别对应于 IEEE 754 单精度浮点数和双精度浮点数。long double 类型对应于扩展双精度浮点数，但 long double 的长度和格式随编译器和处理器类型的不同而有所不同。在 C 程序中等式的赋值和判断中会出现强制类型转换，以 char→int→long→double 和 float→double 最为常见，从前到后范围和精度都从小到大，转换过程没有损失。

1）从 int 转换为 float 时，虽然不会发生溢出，但 int 可以保留 32 位，float 保留 24 位，可能有数据舍入，若从 int 转换为 double 则不会出现。

2）从 int 或 float 转换为 double 时，因为 double 的有效位数更多，因此能保留精确值。

3）从 double 转换为 float 时，因为 float 表示范围更小，因此可能发生溢出。此外，由于有效位数变少，因此可能被舍入。

4）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（仅保留整数部分），影响精度。另外，由于 int 的表示范围更小，因此可能发生溢出。

在不同数据类型之间转换时，往往隐藏着一些不容易察觉的错误，编程时要非常小心。

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` ⇒ `((d*2) < 0.0)`
- `d > f` ⇒ `-f > -d`
- `d * d >= 0.0`
- `x * x >= 0`
- `(d+f)-d == f`

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

假定d 和 f都不是NaN

浮点数习题答案

- x == (int)(float) x — No: float只有24 位尾数
- x == (int)(double) x — Yes: 53位尾数
- f == (float)(double) f — Yes: 增加精度
- d ==(double) (float) d — No: 溢出或损失精度
- f == -(-f); — Yes: 仅仅改变符号位
- 2/3 == 2/3.0 — No: 2/3 == 0
- d < 0.0⇒ ((d*2) < 0.0) — Yes!
- d > f ⇒ -f < -d — Yes
- d *d >= 0.0 — Yes!
- x *x >= 0 — No! 例如50000*50000
- (d+f)-d == f — No: 不具备结合性，可能大数吃小数

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×0.1

  程序用24位数来近似表示十进制0.1：

  x=0.0001 1001 1001 1001 1001 100（二进制）

浮点的悲剧

程序用24位数来近似表示十进制0.1：

x = 0.00011001100110011001100（二进制）

- 单次误差：0.1-x = 0.0000000000000000000000[1100][1100]...$_2$

  = 2^{-20} ×0.1 = 9.54×10^{-8}

- 程序运行100小时后，累计的误差：

  100×3600× 10×9.54×10^{-8} =0.34344秒

  注意：这里的乘以10表示1秒钟有10个0.1秒，产生了10次误差

- 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！！！

思考题：

1. IEEE754比整数部分10位+小数部分20位的表示方法有什么优点？ 缺点呢?
2. float非无穷的最大值，最小值？
3. float数 1，65536，0.4，-1，0的内存表示
4. 一个数的Float形式是唯一的吗？（除了0）
5. 每一个IEEE754编码对应的数是唯一的吗？
6. 简述Float数据的浮点数密度分布？
7. C语言中除以0一定报错溢出吗？（整数报错，浮点无穷大 X/0>Y 可以）
8. int与float都占32个二进制位，Float与INT相比谁的个数多？各自是多少个？多多少？ （+-0、nan）
9. 怎么判断和定义浮点数的无穷大以及NaN？

【例 2.2】将十进制数 123.6875 转换成二进制数。

解：

整数部分：

	除基	取余	
2	123	1	最低位
2	61	1	
2	30	0	
2	15	1	
2	7	1	
2	3	1	
2	1	1	最高位
	0		

整数部分结束条件：商为0

因此整数部分 $123=(1111011)_2$。

乘基取整法（小数部分的转换）：小数部分乘基取整，最先取得的整数为数的最高位，最后取得的整数为数的最低位（即乘基取整，先整为高，后整为低），乘积为 1.0（或满足精度要求）时结束。

小数部分结束条件：乘积为1或者满足精度就结束

小数部分：

乘基	取整	
0.6875 × 2 = 1.3750	1	最高位
0.3750 × 2 = 0.7500	0	
0.7500 × 2 = 1.5000	1	
0.5000 × 2 = 1.0000	1	最低位

最后从高位到低位，写出二进制浮点数的整数和小数部分

因此小数部分 $0.6875=(0.1011)_2$，所以 $123.6875=(1111011.1011)_2$。

注意：在计算机中，小数和整数不一样，整数可以连续表示，但小数是离散的，所以并不是每个十进制小数都可以准确地用二进制表示。例如 0.3，无论经过多少次乘二取整转换都无法得到精确的结果。但任意一个二进制小数都可以用十进制小数表示，希望读者引起重视。

注意：关于十进制数转换为任意进制数为何采用除基取余法和乘基取整法，以及所取之数放置位置的原理，请结合 r 进制数的数值表示公式思考，而不应死记硬背。

请说明float 类型编码格式，并按步骤计算 -0.1的各部分内容，写出 -0.1在内存从低地址到高地址的存储字节内容。

- 单精度: 32 bits，1个符号位，8位指数（127移码），23位尾数（先导为1的规格化）。 -0.1 编码步骤如下：
- (1)首先进行十进制到二进制的转化：采用乘以2取整法

  $-0.0001100110011[0011]..._2$
- (2)表示为科学记数法，先导为1，则

  $-1.10011001100110011001100110011001100 \times 2^{-4}$
- (3)指数部分的+127移码为

  -4+127=123，其二进制形式为 01111011
- (4)尾数部分23位的编码为 1001 1001 1001 1001 1001 1001100

  舍入 1001 1001 1001 1001 1001 101
- (5) IEEE 754编码为：符号位1 （16进制）

  1 01111011 1001 1001 1001 1001 1001 101 = BD CC CC CD
- (6) 内存中倒序：小端模式 CD CC CC BD

经典例题

1、C语言中float类型的数据0.1的机器数表示，错误的是（ ）

A．规格化数 B.不能精确表示 C.与0.2有1个二进制位不同 D. 唯一的

答案：C 考点：浮点数的表示

本章主要以选择、填空或分析题考查

可以下面网站验证一下，究竟有几位是不同的：
https://www.h-schmidt.net/FloatConverter/IEEE754.html

Enjoy!