浮点数笔记

apl

2025 年 3 月 26 日

目录

1 编码

1.1 传统小数的二进制表示

在二进制中，小数点左侧每一位的权重为 2^i，右侧为 $\frac{1}{2^i}$（位置计数法），一个二进制数 b 可以表示为 $b = \sum_{i=-n}^{m} 2^i \times b_i$。例如：$5\frac{3}{4} = 101.11_2$ $2\frac{7}{8} = 010.111_2$

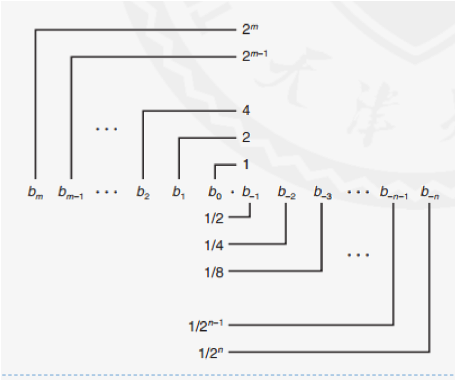

图 1 小数的二进制表示

并且，除以 2 可通过逻辑右移实现，乘以 2 可通过左移实现。像 $0.11111\cdots_2$ 这样的数字十分接近但小于 1，可表示为 $1.0-\varepsilon$。同时，并非所有有理数都能精确地用有限位二进制小数表示，例如 $\frac{1}{3}=0.0101010101[01]\cdots_2$，只能精确表示 $\frac{x}{2^k}$ 这种形式的数字。

1.2 编码浮点数

1.2.1 浮点数的表示

浮点数的表示形式为 $(-1)^s\times M\times 2^E$，其中：

- 符号位 s 决定了数字的正负。
- 尾数 M 通常在 $[1.0,2.0)$ 或 $[0.0,1.0)$。
- 阶码 E 是浮点数的权重，为 2 的 E 次幂。

图 2 浮点数编码

编码方式为：最高位是符号位 s，exp 编码后得到 E（exp $\neq E$），frac 编码后得到 M（frac $\neq M$）。

1.2.2 几种精度的浮点数

常见的浮点数精度有：

精度类型	总位数	符号位	exp 位数	frac 位数
单精度	32 位	1 位	8 位	23 位
双精度	64 位	1 位	11 位	52 位
扩展精度（仅 Intel 支持）	80 位	1 位	15 位	63 或 64 位

表 1: 不同精度浮点数的位宽分布

1.3 浮点数 IEEE 754 标准

1.3.1 规格化数

当 $\exp \neq 000\cdots0$ 且 $\exp \neq 111\cdots1$ 时：

- 尾数编码为包含一个隐式前置的 1，即 $M = 1.x_1x_2\cdots$，其中 $x_1x_2\cdots$ 为 frac 域的各位的编码。
- 阶码为一个有偏置的指数，$E = \text{Exp} - \text{Bias}$，Exp 为 exp 域无符号数编码值，$\text{bias} = 2^{k-1} - 1$，$k$ 是 exp 的位宽。
- 例如，对于单精度浮点数，$\text{bias} = 127$（$\text{Exp} : 1\cdots254$，$E : -126\cdots127$）；双精度浮点数，$\text{bias} = 1023$（$\text{Exp} : 1\cdots2046$，$E : -1022\cdots1023$）。隐式前置的整数 1 始终存在，因此在 frac 中不需要包含。

以 float $f = 15213.0$ 为例：

$$
\begin{aligned}
f = 15213_{10} &= 11101101101010000000_2 \\
&= 1.1101101101101101_2 \times 2^{13} \\
\text{尾数} \quad M &= 1.\mathbf{1101101101101101}_2 \\
\text{frac} &= \mathbf{1101101101101}0000000000_2 \\
\text{阶码} \quad E &= 13, \\
\text{Bias} &= 2^{k-1} - 1 = 127 \quad (k = 8), \\
\text{Exp} &= 140 = 10001100_2
\end{aligned}
$$

结果为 0 10001100 11011011011010000000000。

1.3.2 非规格化数

当 $\exp = 000\cdots0$ 时：

- 阶码 $E = -\text{Bias} + 1$（而不是 $E = 0 - \text{Bias}$）。
- 尾数编码为包含一个隐式前置的 0，即 $M = 0.x_1x_2\cdots$。
- 当 $\exp = 000\cdots0$，且 $\text{frac} = 000\cdots0$ 时，表示 0，要注意 $+0$ 和 -0 的区别。
- 当 $\exp = 000\cdots0$，且 $\text{frac} \neq 000\cdots0$ 时，表示非常接近于 0.0 的数字，这些数字是等间距的。

1.3.3 特殊值

当 $\exp = 111\cdots1$ 时：

- 当 $\exp = 111\cdots1$ 且 $\text{frac} = 000\cdots0$ 时，表示无穷 ∞，意味着运算出现了溢出，有正向溢出和负向溢出，例如 $1.0/0.0 = -1.0/-0.0 = +\infty$，$1.0/-0.0 = -\infty$。

- 当 $\exp = 111\cdots1$ 且 $\text{frac} \neq 000\cdots0$ 时，表示不是一个数字（NaN），表示数值无法确定，例如 $-1, \infty - \infty, \infty \times 0$。

1.3.4 IEEE 编码的特殊属性

1. 浮点数 0 和整数 0 编码相同，所有位都为 0
2. 几乎可以用无符号整数比较的方法实现浮点数的比较运算，但首先要比较符号位，同时要考虑 $-0 = 0$ 和 NaN 的问题 (NaN 比其他值都大)

1.3.5 总结

单精度浮点数可分为以下几类：

1. 规格化数：$\exp \neq 0000000$ 且 $\exp \neq 11111111$。
2. 非规格化数：$\exp = 00000000$，$\text{frac} \neq 0000000$。
3. 无穷：$\exp = 11111111$，$\text{frac} = 00000000$。
4. NaN：$\exp = 11111111$，$\text{frac} \neq 00000000$。

1.4 举例：一个微型的浮点数编码系统

以 8 位浮点数编码为例，最高位为符号位，接下来是 4 位 exp，偏置 Bias 为 7，最后 3 位是 frac，与 IEEE 规范具有相同的形式，有规格化数、非规格化数，以及 0、NaN 和无穷的编码。

s	exp	frac	E	值
0	0000	000	-6	0
0	0000	001	-6	$\frac{1}{8} \times \frac{1}{64} = \frac{1}{512}$（最接近 0）
⋮	⋮	⋮	⋮	⋮
0	1111	000	n/a	inf

表 2: 8 位浮点数编码示例

1.5 需要关注的数字

不同类型的浮点数有一些特殊的数字：

2 运算

2.1 几种舍入模式

常见的舍入模式有：

描述	exp	frac	单精度值（十进制）	双精度值（十进制）
零	$00\cdots00$	$0\cdots00$	0.0	0.0
最小非规格化数	$00\cdots00$	$0\cdots01$	$2^{-23}\times2^{-126}$	$2^{-52}\times2^{-1022}$
最大非规格化数	$00\cdots00$	$1\cdots11$	$(1-\varepsilon)\times2^{-126}$	$(1-\varepsilon)\times2^{-1022}$
最小规格化数	$00\cdots01$	$0\cdots00$	1×2^{-126}	1×2^{-1022}
一	$01\cdots11$	$0\cdots00$	1.0	1.0
最大规格化数	$11\cdots10$	$1\cdots11$	$(2-\varepsilon)\times2^{127}$	$(2-\varepsilon)\times2^{1023}$

表 3: 不同精度浮点数的特殊值

1. 向下舍入：舍入结果接近但不会大于实际结果。
2. 向上舍入：舍入结果接近但不会小于实际结果。
3. 向 0 舍入：舍入结果向 0 的方向靠近，如果为正数，舍入结果不大于实际结果；如果为负数，舍入结果不小于实际结果。
4. 向偶数舍入：浮点数运算默认的舍入模式，其他的舍入模式都会统计偏差，一组正数的总和将始终被高估或低估。

向偶数舍入适用于舍入至小数点后任何位置，当数字正好处在四舍五入的中间时，向最低位为偶数的方向舍入。例如：

值	结果	说明
1.2349999	1.23	比中间值小，四舍
1.2350001	1.24	比中间值大，五入
1.2350000	1.24	中间，向上舍入（偶数方向）
1.2450000	1.24	中间，向下舍入（偶数方向）

表 4: 向偶数舍入示例

在二进制数中，偶数方向意味着舍入后最后一位为 0，中间意味着待舍入的部分为 $100\cdots_2$。

2.2 浮点数运算

浮点数运算的基本思想是先计算出精确的值，然后将结果调整至目标的精度。如果阶码值过大，可能会导致溢出，可能会进行舍入以满足尾数的位宽。例如：

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

2.2.1 浮点数乘法

对于浮点数 $(-1)^{s_1}M_1 2^{E_1}\times(-1)^{s_2}M_2 2^{E_2}$：

- 精确结果：$M_1 \times M_2$，阶码 $E = E_1 + E_2$，符号位 $s = s_1^{s_2}$。
- 修正：如果 $M \geq 2$，右移 M，并增大 E 的值；如果 E 超出范围，发生溢出；对 M 进行舍入以满足 frac 的位宽精度要求。在实际实现中，尾数相乘的细节较为繁琐。

2.2.2 浮点数加法

对于浮点数 $(-1)^{s_1} M_1 2^{E_1} + (-1)^{s_2} M_2 2^{E_2}$（假设 $E_1 > E_2$）：

- 精确结果：$(-1)^s M_2^E$，其中符号位 s 和尾数 M 是有符号数对齐后相加的结果，阶码 $E = E_2$。
- 修正：如果 $M \geq 2$，右移 M，并增大 E 的值；如果 $M < 1$，左移 M k 位，然后 E 减去 k；如果 E 超出范围，发生溢出；对 M 进行舍入以满足 frac 的位宽精度要求。

2.3 C 语言中的浮点数

C 语言标准确保支持两种精度的浮点数，即 float（单精度）和 double（双精度）。在进行类型转换时：

- double/float 转 int：截断尾数部分，向 0 舍入。标准中未定义越界和 NaN 的情况，通常设置为 T_{Min} 和 T_{Max}。
- int 转 double：只要 int 的位宽小于等于 53 位，就能精确转换。
- int 转 float：会根据舍入模式进行舍入。

以下是一些 C 语言中浮点数操作的示例（假设 d 和 f 分别是 float 和 double 且不是 NaN 和无穷）：

表达式	结果
`x == (int)(float) x`	否：有效数字为 24 位
`x == (int)(double) x`	是：有效数字为 53 位
`f == (float)(double) f`	是：提高精度
`d == ( float ) d`	否：丢失精度
`f == -(-f)`	是：仅改变符号位
`2/3 == 2/3.0`	否：$2/3 == 0$
`if(d < 0.0) ((d*2) < 0.0)`	是
`d * d >= 0.0`	是
`(d + f) - d == f`	否：不满足结合律

表 5: C 语言中浮点数操作示例结果

2.4 精确度

在浮点数运算中，存在精度和准确度的问题。例如 $0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1 \neq 1$，但 $0.2+0.2+0.2+0.2+0.2 = 1$。在使用浮点数进行比较时，如`if (d1 - d2 == 0)`，需要谨慎处理，因为浮点数的精度限制可能导致结果不准确。在进行浮点数运算时，要充分考虑精度损失对计算结果的影响。