# 単精度浮動小数点数

単精度浮動小数点の表現方法、最近接偶数方向丸め、加算時の丸め誤差についてまとめる。

## 1. 単精度浮動小数点数 （float）

一般に 2 進 $p$ 桁の浮動小数点は以下のように表現される：

$$\begin{aligned}&(-1)^s \times \left(1 + \frac{m_1}{2^1} + \frac{m_2}{2^2} + \cdots + \frac{m_{p-1}}{2^{p-1}}\right) \times 2^e \\ &= (-1)^s \left(1.m_1 m_2 \cdots m_{p-1}\right)_2 \times 2^e \\ &= (\text{符号部}) \times (\text{仮数部}) \times 2^{(\text{指数部})}\end{aligned} \tag{1}$$

ただし

$$s,\, m_i \in \{0,\, 1\},\, e \in Z$$

である。

IEEE 754 での単精度浮動小数点数の形式では

- 符号ビット: 1 ビット
- 指数部の幅: 8 ビット
- 仮数部の幅: 23 ビット

の 32 bit を用いて以下のように表現している（図 1）。式 (1) の仮数部はつねに 1 から始まるので、この 1 を省略して残りの部分を 23 bit で表す。このため IEEE 754 の仮数部は実質 24 bit を表すので、10 進に換算したとき表現できる桁数は $\log_{10}(2^{24}) \approx 7.225$ 桁となり有効数字は 7 桁となる。

また、図 1 の exponent の 8 bit を 10 進数に変換したものから 127 を引いた値が式 (1) の指数部となる。すなわち

$$(\text{指数部}) = (\text{exponent}\,8\,\text{bit})_{10} - 127 \tag{2}$$

である。

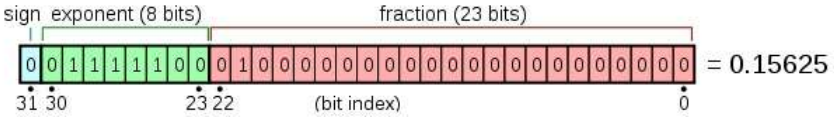


図 1　IEEE 754 での binary32 の標準化されたレイアウト

図 1 の場合を 10 進数で表すと、指数部が

$$\left(2^6 + 2^5 + 2^4 + 2^3 + 2^2\right) - 127 = 124 - 127 = -3$$

となるので

$$(-1)^0 \times \left(1 + \frac{1}{2^2}\right) \times 2^{-3} = \frac{5}{4} \times \frac{1}{8} = \frac{5}{32} = 0.15625$$

となる。

## 2. 最近接偶数方向丸め (round to nearest even)

IEEE 754 の仮数部は実質 24 bit であるので、24 bit を超える場合は 24 bit に丸めることが必要となる。この丸め方の一つが**最近接偶数方向丸め (RN : round to nearest even)**[1] である。この方法は C 言語でも用いられている。

この最近接偶数方向丸めでは

(I) 数値が最近接の 2 数の浮動小数点のちょうど真ん中の数以外

⇒ 最近接の 2 数の浮動小数点のうち、最も近い浮動小数点数に選択する。

(II) 数値が最近接の 2 数の浮動小数点のちょうど真ん中の数

⇒ 最近接の 2 数の浮動小数点のうち、仮数部の最下位が 偶数 (0) の方を選択する。

簡単のため仮数部が 3 bit、数値が 4 bit の場合を考える（¦ の右側は溢れた bit を表す）。

- $x = 1.00¦0$ の場合：最近接する 2 つの浮動小数点 $x_1, x_2$ とそのちょうど真ん中の値 $x_{\mathrm{mid}}$ は

$$\begin{aligned} x_1 &= 1.00 \\ x_2 &= 1.01 \\ x_{\mathrm{mid}} &= 1.00¦1 \end{aligned}$$

  であり、$x < x_{\mathrm{mid}}$ は **(I)** の場合に相当するので、最も近い浮動小数点 $x_1 = 1.00$ を選択し、$x = 1.00$ と丸められる。
- $x = 1.00¦1$ の場合：最近接する 2 つの浮動小数点 $x_1, x_2$ とそのちょうど真ん中の値 $x_{\mathrm{mid}}$ は

$$\begin{aligned} x_1 &= 1.00 \\ x_2 &= 1.01 \\ x_{\mathrm{mid}} &= 1.00¦1 \end{aligned}$$

  であり、$x = x_{\mathrm{mid}}$ は **(II)** の場合に相当するので、仮数部の最下位が 偶数 (0) の $x_1 = 1.00$ を選択し、$x = 1.00$ と丸められる。
- $x = 1.01¦0$ の場合：最近接する 2 つの浮動小数点 $x_1, x_2$ とそのちょうど真ん中の値 $x_{\mathrm{mid}}$ は

$$\begin{aligned} x_1 &= 1.01 \\ x_2 &= 1.10 \\ x_{\mathrm{mid}} &= 1.01¦1 \end{aligned}$$

  であり、$x < x_{\mathrm{mid}}$ は **(I)** の場合に相当するので、最も近い浮動小数点 $x_1 = 1.01$ を選択し、$x = 1.01$ と丸められる。
- $x = 1.01¦1$ の場合：最近接する 2 つの浮動小数点 $x_1, x_2$ とそのちょうど真ん中の値 $x_{\mathrm{mid}}$ は

$$\begin{aligned} x_1 &= 1.01 \\ x_2 &= 1.10 \\ x_{\mathrm{mid}} &= 1.01¦1 \end{aligned}$$

  であり、$x = x_{\mathrm{mid}}$ は **(II)** の場合に相当するので、仮数部の最下位が 偶数 (0) の $x_2 = 1.10$ を選択し、$x = 1.10$ と丸められる。

## 3. 浮動小数点の加算と丸め誤差

浮動小数点の加算 [2] は次のように行われる。

1. 10 進数を内部表現に変換する。[3, 4]
2. 仮数部の先頭に 1 を追加し、24 bit とする。
3. 桁合わせのために指数部の差を求める。
4. 小さい方の数の仮数部を指数部の差の分だけ右にシフトする。
5. 最近接偶数方向丸めに従って仮数部を丸める（この段階で丸め誤差が発生）。[1]
6. 仮数部の加算を行い、和の内部表現を得て、それを 10 進数に変換する。[3, 4]

以下では

- 16777216.0 + 1.0
- 16777216.0 + 2.0
- 16777216.0 + 3.0
- 16777216.0 + 1.1

の加算を C 言語で計算した場合の丸め誤差の発生原因について考える。

```c
int main()

    float a = 1 << 24; // 16777216
    float b = 1;       // b の値を変化させて丸め誤差の影響を調べる
    float c;

    c = a + b;

    printf("a = %f \n", a);
    printf("b = %f \n", b);
    printf("c = %f \n", c);

    return 0;
}

// results

a = 16777216.000000
b =        1.000000
c = 16777216.000000

a = 16777216.000000
b =        2.000000
c = 16777218.000000

a = 16777216.000000
b =        3.000000
c = 16777220.000000

a = 16777216.000000
b =        1.100000
c = 16777218.000000
```

3.1 $a = 16777216,\ b = 1\ \Rightarrow\ a + b = 16777216$

$a = 16777216$ と $b = 1$ の指数部の差は 24 bit となるため、右シフト後の $b$ の仮数部は

$$b \to b' = (0000\,0000\,0000\,0000\,0000\,0000 \,|\, 1000\,0000\,0000\,0000\,0000\,0000)_2$$

となる。この $b'$ に最隣接する 2 つの浮動小数点 $b_1, b_2$ とそのちょうど真ん中の値 $b_{\mathrm{mid}}$ は

$$\begin{aligned} b_1 &= (0000\,0000\,0000\,0000\,0000\,0000)_2 \\ b_2 &= (0000\,0000\,0000\,0000\,0000\,0001)_2 \\ b_{\mathrm{mid}} &= (0000\,0000\,0000\,0000\,0000\,0000 \,|\, 1)_2 \end{aligned}$$

である。この場合、$b' = b_{\mathrm{mid}}$ であるので **(II)** の場合に相当し、$b_1$ が選択され

$$b = (0000\,0000\,0000\,0000\,0000\,0000)_2$$

と丸められる。この場合には 0 が仮数部に加算されることになるので、加算の結果は 16777216 となり値が変化しない。


図 2　浮動小数点の加算と丸め誤差 (1)

したがって、以下に示すような 1.0 を $n = 20000000$ 回加えるような計算を行った場合、float 型では 16777216.0 を超えると 1.0 を加えても丸め誤差のため値が変化しないので、注意が必要である。

```c
#include <stdio.h>

int main()
{
    int n = 20000000;
    float sum1 = 0.0;
    int sum2 = 0;

    for (int j = 0; j < n; j++) {
        sum1 += 1.0;
        sum2 += 1;
    }

    printf("sum1 = %f \n", sum1);
    printf("sum2 = %d \n", sum2);

    return 0;
}

// results

sum1 = 16777216.000000
sum2 = 20000000
```

この場合には、float を double へ変更することで正しく計算できる。

3.2 $a = 16777216,\ b = 2\ \Rightarrow\ a + b = 16777218$

$a = 16777216$ と $b = 2$ の指数部の差は 23 bit となるため、右シフト後の $b$ の仮数部は

$$b \to b' = (0000\,0000\,0000\,0000\,0000\,0001\,|\,0000\,0000\,0000\,0000\,0000\,000)_2$$

となる。この $b'$ に最隣接する 2 つの浮動小数点 $b_1, b_2$ とそのちょうど真ん中の値 $b_{\mathrm{mid}}$ は

$$\begin{aligned} b_1 &= (0000\,0000\,0000\,0000\,0000\,0001)_2 \\ b_2 &= (0000\,0000\,0000\,0000\,0000\,0010)_2 \\ b_{\mathrm{mid}} &= (0000\,0000\,0000\,0000\,0000\,0001\,|\,1)_2 \end{aligned}$$

である。この場合、$b' < b_{\mathrm{mid}}$ であるので **(I)** の場合に相当し、$b_1$ が選択され

$$b = (0000\,0000\,0000\,0000\,0000\,0001)_2$$

と丸められる。この $b$ が仮数部に加算され、加算結果の内部表現は

$$a + b = (0\,100\,1011\,1\,000\,0000\,0000\,0000\,0000\,0001)_2$$

となる。これを 10 進数に変換すれば 16777218 となる。


図 3　浮動小数点の加算と丸め誤差 (2)

3.3 $a = 16777216,\ b = 3\ \Rightarrow\ a + b = 16777220$

$a = 16777216$ と $b = 3$ の指数部の差は 23 bit となるため、右シフト後の $b$ の仮数部は

$$b \to b' = (0000\,0000\,0000\,0000\,0000\,0001\,|\,1000\,0000\,0000\,0000\,0000\,000)_2$$

となる。この $b'$ に最隣接する 2 つの浮動小数点 $b_1$, $b_2$ とそのちょうど真ん中の値 $b_{\mathrm{mid}}$ は

$$b_1 = (0000\,0000\,0000\,0000\,0000\,0001)_2$$
$$b_2 = (0000\,0000\,0000\,0000\,0000\,0010)_2$$
$$b_{\mathrm{mid}} = (0000\,0000\,0000\,0000\,0000\,0001\,|\,1)_2$$

である。この場合、$b' = b_{\mathrm{mid}}$ であるので **(II)** の場合に相当し、$b_2$ が選択され

$$b = (0000\,0000\,0000\,0000\,0000\,0010)_2$$

と丸められる。この $b$ が仮数部に加算され、加算結果の内部表現は

$$a + b = (0\,100\,1011\,1\,000\,0000\,0000\,0000\,0000\,0010)_2$$

となる。これを 10 進数に変換すれば 16777220 となる。


図 4　浮動小数点の加算と丸め誤差 (3)

3.4 $a = 16777216,\ b = 1.1\ \Rightarrow\ a + b = 16777218$

$a = 16777216$ と $b = 1.1$ の指数部の差は 24 bit となるため、右シフト後の $b$ の仮数部は

$$b \to b' = (0000\,0000\,0000\,0000\,0000\,0000 \mid 1000\,1100\,1100\,1100\,1100\,1100)_2$$

となる。この $b'$ に最隣接する 2 つの浮動小数点 $b_1$, $b_2$ とそのちょうど真ん中の値 $b_{\mathrm{mid}}$ は

$$\begin{aligned} b_1 &= (0000\,0000\,0000\,0000\,0000\,0000)_2 \\ b_2 &= (0000\,0000\,0000\,0000\,0000\,0001)_2 \\ b_{\mathrm{mid}} &= (0000\,0000\,0000\,0000\,0000\,0000 \mid 1)_2 \end{aligned}$$

である。この場合、$b' > b_{\mathrm{mid}}$ であるので **(I)** の場合に相当し、$b_2$ が選択され

$$b = (0000\,0000\,0000\,0000\,0000\,0001)_2$$

と丸められる。この $b$ が仮数部に加算され、加算結果の内部表現は

$$a + b = (0\,100\,1011\,1\,000\,0000\,0000\,0000\,0000\,0001)_2$$

となる。これを 10 進数に変換すれば 16777218 となる。


図 5　浮動小数点の加算と丸め誤差 (4)

## 参考文献


[1] 浮動小数点の丸めの方向と性質 (1)
http://math-koshimizu.hatenablog.jp/entry/2016/12/04/224208

[2] 浮動小数点加算器 - 桁合わせと加算
https://news.mynavi.jp/article/architecture-96/

[3] 情報の表現

http://ss.sguc.ac.jp/~rider/basic/float.html

[4] 浮動小数点数内部表現シミュレーター

https://tools.m-bsys.com/calculators/ieee754.php