IEEE754 倍精度浮動小数点数のフォーマット

桂田祐史

2005 年 5 月 2 日

1 解説

IEEE 754 規格は「有名」であるが、きちんと説明してある書籍は案外入手しづらい (少し解説してある本も、冗長になりがちなので、かなりはしょることになって、知らない人には読みにくいものになっていると感じる)。個人的に意外に便利だったのは、SunOS のコンパイラーのマニュアルであったが、ここでは、倍精度浮動小数の表現に絞って、self-contained に解説する。

64 ビットあるわけだが、各ビットを、最上位ビット MSB (=most significant bit) から順に番号をつけて b_i $(i = 0, \cdots, 63)$ と表わそう。

1.1 符号

b_0 は符号ビットである。つまり $b_0 = 1$ は広い意味での負の数を表わす。

「広い意味で」と書いたのは、「-0」と「負の無限大」という、「普通の負の数」でないものを表現する場合があるからである。詳しくは後述するが、例えば $b_0 = 1, b_i = 0$ $(i = 1, 2, \cdots, 63)$ であるパターンは 0 を表わしていて、しばしば -0 と呼ばれる。

1.2 二つの 0

0 を表わすビット・パターンには二つある。すべてのビットが 0、つまり

$$b_i = 0 \quad (i = 0, 1, \cdots, 63)$$

である場合と、最初のビットだけが 1 で、後はすべて 0, つまり

$$b_0 = 1, \quad b_i = 0 \quad (i = 1, 2, \cdots, 63)$$

である場合である。区別をするため、前者を $+0$, 後者を -0 と呼ぶことがある。

1.3 指数部

b_i $(i = 1, 2, \cdots, 11)$ は指数部を表わす。b_i $(i = 1, 2, \cdots, 11)$ の表わす整数を e としよう:

$$e = \sum_{i=1}^{11} 2^{11-i} b_i.$$

e の範囲は

$$0 \leq e \leq 2^{11} - 1 = 2047.$$

となる。

$e = 2^{11} - 1 = 2047$、つまり

$$b_i = 1 \quad (i = 1, 2, \cdots, 11)$$

である場合は無限大を表わすのに使われる。

$$b_0 = 0, \quad b_i = 1 \quad (i = 1, 2, \cdots, 11), \quad b_i = 0 \quad (i = 12, 13, \cdots, 63)$$

は $+\infty$ を、

$$b_0 = 1, \quad b_i = 1 \quad (i = 1, 2, \cdots, 11), \quad b_i = 0 \quad (i = 12, 13, \cdots, 63)$$

は $-\infty$ を表わす。

$e = 0$ の場合は、正規化されていない数、つまり ± 0 や絶対値が 2^{-1023} 以下の数を表現するのに使われる (詳しくは後述)。

1.4 正規化数

$1 \leq e \leq 2^{11} - 2 = 2046$ の場合は正規化されている数 x を表わす。

$$x = (-1)^{b_0} 2^{e-1023} \left(1 + \sum_{i=12}^{63} \frac{b_i}{2^{i-11}} \right), \quad e = \sum_{i=1}^{11} 2^{11-i} b_i.$$

b_i $(i = 12, 13, \cdots, 63)$ が仮数部を表わすのに用いられているが、仮数部が

$$\left(\sum_{i=12}^{63} \frac{b_i}{2^{i-11}} \right) \quad \text{などではなく} \quad \left(1 + \sum_{i=12}^{63} \frac{b_i}{2^{i-11}} \right) \quad \text{である}$$

こと、いわゆる**ケチ表現**[1]を採用していることに注意しよう。

絶対値が最小の数は

$$e = 1, \quad b_i = 0 \quad (i = 12, \cdots, 63)$$

の場合で、

$$\pm 2^{1-1023} = \pm 2^{-1022} \fallingdotseq \pm 2.2250738585072014 \times 10^{-308}.$$

絶対値が最大の数は

$$e = 2046, \quad b_i = 1 \quad (i = 12, \cdots, 63)$$

の場合で、

$$\begin{aligned} \pm 2^{2046-1023} \left(1 + \sum_{i=12}^{63} \frac{1}{2^{i-11}} \right) &= \pm 2^{1023} \left(1 + \sum_{i=1}^{52} \frac{1}{2^i} \right) = \pm 2^{1023} \frac{1 - (1/2)^{53}}{1 - 1/2} \\ &= \pm 2^{1024} \left(1 - (1/2)^{53} \right) \fallingdotseq \pm 1.7976931348623157081 \times 10^{308}. \end{aligned}$$

[1] 0 でない実数の 2 進数表現は、当然 0 でない桁があるわけだから、最上位の 1 を省略することで、桁数が稼げるというアイディアである。

1.5 非正規化数

$e = 0$ の場合は、

$$(-1)^{b_0} 2^{-1023} \sum_{i=12}^{63} \frac{b_i}{2^{12-i}}$$

を表わすと約束する。ぼうっと見ていると、正規化数の場合の式と同じように思えるかもしれないが、ケチ表現を使っていないことに注意が必要である。

0 でない数のうちで、絶対値が最小の数は

$$b_i = 0 \quad (i = 12, 13, \cdots, 62), \quad b_{63} = 1$$

の場合で、

$$\pm 2^{-1023} \times \frac{1}{2^{12-63}} = \pm 2^{-1023} \cdot 2^{-51} = \pm 2^{-1074} \fallingdotseq \pm 4.94065645841246544 \times 10^{-324}.$$

絶対値が最大の数は

$$b_i = 1 \quad (i = 12, 13, \cdots, 63)$$

の場合で、

$$\pm 2^{-1023} \sum_{i=12}^{63} \frac{1}{2^{12-i}} = \pm 2^{-1023} \frac{1 - (1/2)^{52}}{1 - (1/2)} = \pm 2^{-1022} \left(1 - (1/2)^{52}\right)$$

である。これはもちろん、正規化数のうちで絶対値が最小の数に近い (ほんの少し小さい) — そうなるように設計しているわけだから。

1.6 むすび

10 年以上前に書かれた数値解析、数値計算法の本には、一般の浮動小数点数の解説があるが、最近は IEEE 754 規格のみを解説してある本が目につくようになってきた。ユーザーが実際に触れる機会のあるコンピューター・システムのほとんどが IEEE 754 規格を採用していることからも、またそれを理解すれば他の浮動小数点数システムも容易に理解可能なことからも、そのやり方で十分なのだと思う。

一方で、非正規化数や無限大は、すべての浮動小数点数システムに備わっているわけではないが、非常に有用なことは明らかに近いと思う。

2 bdsd — 実験のための小プログラム

double データのビット・パターンを 0, 1 からなる文字列 (型名を bdsd) に変換する関数 `void double_to_bdsd(double, bdsd)` と、bsds データを表示する `void print_bdsd(bdsd)` を用意した。

bdsd.h

```
/*
 * bdsd.h
 */

#define NUM_BIN_DIGITS_DOUBLE 64
typedef char bin_digit_string_double[NUM_BIN_DIGITS_DOUBLE+1];
typedef bin_digit_string_double bdsd;

void double_to_bin_digits(double, bdsd);
void print_bdsd(bdsd);
```

bdsd.c

```
/*
 * bdsd.c
 */

#define NUM_BIN_DIGITS_INT 32
typedef char bds_int[NUM_BIN_DIGITS_INT+1];

#include <stdio.h>
#include "bdsd.h"

static void int_to_bin_digits(int x, bds_int s)
{
  int keta;
  s[NUM_BIN_DIGITS_INT - 1] = '0' + (x & 1);
  x = (x >> 1) & 0x7fffffff;
  for (keta = NUM_BIN_DIGITS_INT - 2; keta >= 0; keta--) {
    s[keta] = '0' + (x & 1);
    x = x >> 1;
  }
  s[NUM_BIN_DIGITS_INT] = 0;
}

void double_to_bin_digits(double x, bdsd bds)
{
  union {
    double x;
    int xa[2];
  } data;
  data.x = x;
  int_to_bin_digits(data.xa[1], bds);
  int_to_bin_digits(data.xa[0], bds+NUM_BIN_DIGITS_INT);
}

void print_bdsd(bdsd a_bdsd)
{
  int i;
  printf("%c ", a_bdsd[0]);
  for (i = 1; i <= 11; i++)
    putchar(a_bdsd[i]);
  printf(" %s", a_bdsd + 12);
}
```

3　先頭のビットが符号を表わすこと

sign.c

```
/*
 * sign.c --- MSB が符号を表わすことの確認
 */

#include <stdio.h>
#include <math.h>
#include "bdsd.h"

int main()
{
  double x;
  bdsd s;

  x = 1; double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");
  x *= -1; double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");

  x = 4.0 * atan(1.0); double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");
  x *= -1; double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");

  x = 0; double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");
  x *= -1; double_to_bin_digits(x, s);
  printf("x=%g\n", x); print_bdsd(s); printf("\n");

  return 0;
}
```

sign 実行結果

```
yurichan% ./sign
x=1
0 01111111111 0000000000000000000000000000000000000000000000000000
x=-1
1 01111111111 0000000000000000000000000000000000000000000000000000
x=3.14159
0 10000000000 1001001000011111101101010100010001000010110100011000
x=-3.14159
1 10000000000 1001001000011111101101010100010001000010110100011000
x=0
0 00000000000 0000000000000000000000000000000000000000000000000000
x=0
1 00000000000 0000000000000000000000000000000000000000000000000000
yurichan%
```

先頭のビットが符号を表わしていること、また確かに 0 にも二種類あることが分かる。

4　絶対値の小さな数

small.c

```
/*
 * small.c
 */

#include <stdio.h>
#include "bdsd.h"

int main()
{
  int i;
  double x;
  bdsd s;
  x = 1;
  for (i = 1; i <= 1080; i++) {
    double_to_bin_digits(x, s);
    printf("2^{-%-4d}=", i-1); print_bdsd(s); printf("\n=%25.20e\n", x);
    x /= 2;
  }

  return 0;
}
```

要するに 1 から始めて、1/2 倍していった数の 2 進表現を表示している。

$2^{-1022} \fallingdotseq 2.2250738585072014 \times 10^{-308}$ までは正規化浮動小数点数として表現できるが、それ以降は非正規化浮動小数点数になり、$2^{-1074} \fallingdotseq 4.94065645841246544 \times 10^{-324}$ が正の最小の浮動小数点数で、その後は 0 にアンダーフローする。

```
yurichan% ./small
2^{-0   }=0 01111111111 0000000000000000000000000000000000000000000000000000000000000000
=1.0000000000000000000e+00
2^{-1   }=0 01111111110 0000000000000000000000000000000000000000000000000000000000000000
=5.0000000000000000000e-01
2^{-2   }=0 01111111101 0000000000000000000000000000000000000000000000000000000000000000
=2.5000000000000000000e-01
2^{-3   }=0 01111111100 0000000000000000000000000000000000000000000000000000000000000000
=1.2500000000000000000e-01
2^{-4   }=0 01111111011 0000000000000000000000000000000000000000000000000000000000000000
=6.2500000000000000000e-02
中略
2^{-1019}=0 00000000100 0000000000000000000000000000000000000000000000000000000000000000
=1.78005908680576110647e-307
2^{-1020}=0 00000000011 0000000000000000000000000000000000000000000000000000000000000000
=8.90029543402880553236e-308
2^{-1021}=0 00000000010 0000000000000000000000000000000000000000000000000000000000000000
=4.45014771701440276618e-308
2^{-1022}=0 00000000001 0000000000000000000000000000000000000000000000000000000000000000
=2.22507385850720138309e-308
 ↑これが最小の正規化浮動小数点数、これ以降は仮数部に 1 が現れ、シフトしていく
2^{-1023}=0 00000000000 1000000000000000000000000000000000000000000000000000000000000000
=1.11253692925360069155e-308
2^{-1024}=0 00000000000 0100000000000000000000000000000000000000000000000000000000000000
=5.56268464626800345773e-309
2^{-1025}=0 00000000000 0010000000000000000000000000000000000000000000000000000000000000
=2.78134232313400172886e-309
2^{-1026}=0 00000000000 0001000000000000000000000000000000000000000000000000000000000000
=1.39067116156700086443e-309
中略
2^{-1071}=0 00000000000 0000000000000000000000000000000000000000000000000000000000001000
=3.95252516672997235341e-323
2^{-1072}=0 00000000000 0000000000000000000000000000000000000000000000000000000000000100
=1.97626258336498617671e-323
2^{-1073}=0 00000000000 0000000000000000000000000000000000000000000000000000000000000010
=9.88131291682493088353e-324
2^{-1074}=0 00000000000 0000000000000000000000000000000000000000000000000000000000000001
=4.94065645841246544177e-324
 ↑これが正の最小の浮動小数点数（仮数部のはしっこに風前の灯火ビットが見える）
2^{-1075}=0 00000000000 0000000000000000000000000000000000000000000000000000000000000000
=0.00000000000000000000e+00
 ↑アンダー・フローしてオール・ゼロになった。
後略
yurichan%
```

5　絶対値の大きな数

large.c

```
/*
 * large.c
 */

#include <stdio.h>
#include "bdsd.h"

int main()
{
  int i;
  double x;
  bdsd s;
  x = 1;
  for (i = 1; i <= 1040; i++) {
    double_to_bin_digits(x, s);
    printf("2^%-4d=", i-1); print_bdsd(s); printf("\n=%g\n", x);
    x *= 2;
  }
  return 0;
}
```

large 実行結果

```
yurichan% ./large
2^0   =0 01111111111 0000000000000000000000000000000000000000000000000000
=1
2^1   =0 10000000000 0000000000000000000000000000000000000000000000000000
=2
2^2   =0 10000000001 0000000000000000000000000000000000000000000000000000
=4
2^3   =0 10000000010 0000000000000000000000000000000000000000000000000000
=8
2^4   =0 10000000011 0000000000000000000000000000000000000000000000000000
=16
2^5   =0 10000000100 0000000000000000000000000000000000000000000000000000
=32
中略
2^1020=0 11111111011 0000000000000000000000000000000000000000000000000000
=1.12356e+307
2^1021=0 11111111100 0000000000000000000000000000000000000000000000000000
=2.24712e+307
2^1022=0 11111111101 0000000000000000000000000000000000000000000000000000
=4.49423e+307
2^1023=0 11111111110 0000000000000000000000000000000000000000000000000000
=8.98847e+307
Floating exception (core dumped)
yurichan%
```

今度は倍々ゲームで 2^{1023} の次に 2^{1024} を計算しようとして、例外が発生して終了した。

もちろん (既に説明したように)、表現可能な最大の数は 2^{1023} よりも大きく、$2^{1024}(1-(1/2)^{53}) \fallingdotseq 1.797 \times 10^{308}$ である。

large2.c

```
/*
 * large2.c
 */

#include <stdio.h>
#include "bdsd.h"

int main()
{
  double x;
  bdsd s;

  x = 1.7976931348623157e308;
  double_to_bin_digits(x, s);
  print_bdsd(s); printf("\n=%25.20e\n", x);
  x = 1.8e308;
  double_to_bin_digits(x, s);
  print_bdsd(s); printf("\n=%25.20e\n", x);
  x = -1.8e308;
  double_to_bin_digits(x, s);
  print_bdsd(s); printf("\n=%25.20e\n", x);

  return 0;
}
```

large2 実行結果

```
yurichan% ./large2
0 11111111110 1111111111111111111111111111111111111111111111111111
=1.79769313486231570815e+308
0 11111111111 0000000000000000000000000000000000000000000000000000
=                       Inf
1 11111111111 0000000000000000000000000000000000000000000000000000
=                      -Inf
yurichan%
```