計算機における数の表現

桂田祐史

2000 年 7 月 6 日

1　はじめに

計算機の中では、数をどのように表現しているのか？

「実数体系という名称はあまり適切とは思えないが、微積分をはじめ高等な解析学の基盤をなしているのがこの実数体系である。そのために、すべての実数を現実の有限な計算機で表現することが不可能なことをつい忘れがちになる。しかし、実数体系が理論的解析をやさしくしている部分は、実際の計算では逆にそれ無しでやってゆかなければならないのである。」 — Forsythe (森 正武 訳)

ここで FORTRAN や C 等のプログラミング言語であらかじめ用意されているデータ型について列挙してみよう。

FORTRAN intger, real, real*8, real*16, complex, complex*16, complex*32

C short, unsigned short, int, unsigned int, float, double, long double

色々あるが、いずれもあらかじめ定まった量の記憶域を用いて数を表現することに注意しよう。従って**有限個の数しか表現することはできない**。

これらのデータ型は大きく二つに分類することが出来る。整数のみを表現できるものと、それ以外のもの — 実数（あるいは複素数）を表現出来る — ものである。

（多くの Lisp や数式処理言語では、bignum とか「無限多倍長」と呼ばれるデータ型があり、これは数を表すために用いる記憶域の大きさが動的に変化する。しかし、この場合も表す数の個数は有限個であることは変わらないし、また数表現の基本的な考え方は後述のものと同じである。）

2　予備的な知識

現在の（ほとんど）すべての電子計算機の内部では、データは 0 または 1 の値を取りうる数（**ビット bit** と呼ぶ）の列として表される（要するにデジタルということ）。これが「計算機内部では数は 2 進法だ」と言われる由縁である。

数を 2 進表示すると、桁数が多くなって人間には読みにくいので、4 桁ずつまとめて **16 進法**にすることが行われている。表記に使う文字は 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F の 16 文字である。（一部のシステムでは 3 桁ずつまとめた **8 進法**も使われる。）

8 ビットを**バイト** (**byte**) と呼ぶことが多い（過去には 8 ビットでない 1 バイトが存在したが）。

実際の計算機では適当な個数のビットをまとめて処理することが多い。いわば 2 進法の算盤(そろばん)が並んでいるようになっている。例えばパソコンや多くのワークステーションでは主記憶装置はバイト単位に番地をつけられて、読み書きの最少単位はバイトである。また **CPU** (central processing unit, 中央演算装置) には**レジスター** (**置数器**) と呼ばれる演算用の算盤があるが、この桁数をそのマシンのビット数ということがある (ビット数は現在では 8,16,32,64 のような 2 のべきであることが多い)。例えば SPARCstation は 32 ビット・マシンである。このように CPU のレジスターに収められるサイズのデータは、その計算機に取って最も基本的なデータの取り扱い量であるとみなすことが出来る。そのため、その量のことをその計算機の**ワード** (**word**) と呼ぶこともある。例えば、SPARCstation では 1 word = 32 bits である。

3 整数

整数[1]は数値シミュレーションではデータとして使われることは滅多にないが、簡単に解説しておく。

m ビット・マシンでは、m ビットのデータ (ワード) を標準の整数データを表現するために使うことが多い。m ビットで 2^m 個のデータを表現することが出来る ($m = 16$ では 65536 個、$m = 32$ では 4294967296 個)。正の数だけあれば間に合うならば、ごく自然に $0, 1, \cdots, 2^m - 1$ の数を対応させることが出来る (実際 $b_{m-1}, b_{m-2}, \cdots, b_1, b_0$ を m ビットのデータとして、$N = 2^{m-1}b_{m-1} + \cdots + 2^k b_k + \cdots + 2^2 b_2 + 2b_1 + b_0$ を対応させればよい)。C 言語で "unsigned (符号無)" を冠したデータ型はまさにこれである。

しかし大抵は負の数をも必要とする。素朴に考えると、1 ビット (例えば b_{m-1}) を符号を表すのに用いて、残りの $m-1$ ビットで絶対値を表すようにする方法が思い浮かぶ。これは**絶対値表示**と呼ばれ、ある程度使われて来た。しかし現在ポピュラーな方法は**補数表示**と呼ばれる方法である。大抵は **2 の補数表示**であるが、1 の補数というものもある。

1 の補数

$b_{m-1}b_{m-2}\cdots b_1 b_0$ というビットの列は 1 の補数表示では次のように解釈される。

- $b_{m-1} = 0$ ならば $N = b_0 + 2b_1 + 2^2 b_2 + \cdots + 2^k b_k + \cdots + 2^{m-2}b_{m-2}$. $(0 \le N \le 2^{m-1} - 1$ となる)
- $b_{m-1} = 1$ ならば $N = -(c_0 + 2c_1 + 2^2 c_2 + ... + 2^k c_k + ... + 2^{m-2}c_{m-2})$. ただし $c_i = 1 - b_i$ とした。 $(-2^{m-1} + 1 \le N \le 0$ となる)

要するに 1 の補数表示は本質的には絶対値表示とあまり変わらない。
例えば 8 ビットの場合、-1 を表すには 00000001 の各桁を反転して 11111110 となる。

2 の補数表示では、ビット列 $b_{m-1}b_{m-2}\cdots b_1 b_0$ の表す数は

$$N = b_0 + 2b_1 + 2^2 b_2 + \cdots + 2^k b_k + \cdots + 2^{m-2}b_{m-2} - 2^{m-1}b_{m-1}$$

とする。$-2^{m-1} \le N \le 2^{m-1} - 1$ となる。

[1]いわゆる「固定小数点数」もこの範疇に含まれると思って良い。

2 の補数表示を用いる理由は、符号無しの場合の演算をする場合と回路が共通化出来ること、複数のワードを用いて広い範囲の数を表現する**多倍長整数**の演算の実現が簡単なことである。

演算をすることによって、結果が m ビットの範囲に収まらなくなった場合は、とにかく下位 m ビットは残し、溢れ出たものは適当に処理する (ことが多い)。加算、減算ならば溢れるのは 1 ビットなので、CPU に備わっているフラグに記憶する。乗算の場合は、溢れたものを捨ててしまうか、別のレジスターに収めることにする。加減算についてはデータを符号無しと考えても、符号有りと考えても、結果の下位 m ビットは同じものになることに注意。10 進数で説明すると

$$\begin{array}{rrrr} & 9 & 9 & 9 \\ & 9 & 9 & 8 \\ 1| & 9 & 9 & 7 \end{array} \qquad \begin{array}{rrrr} & 9 & 9 & 9 \\ & 9 & 9 & 8 \\ 1| & 9 & 9 & 7 \end{array} \qquad \begin{array}{r} -1 \\ -2 \\ -3 \end{array}$$

m ビット・マシンで 2 の補数表示を使うのは、加減乗算については、2^m を法とする剰余系を考えて、代表元として $-2^{m-1}, -2^{m-1}+1, \cdots, -1, 0, 1, 2, \cdots, 2^{m-1}$ を取っているとして理解できる。

これに対して割算については、正数同士の場合はあまり問題がないが、負数を含む演算の結果についてはあまり合理的な説明は出来ない。割り切れない場合、剰余の符号をどう取るか、それに関連して商 (整数) をどうするか。($-7 \div 2$ は "-3 余り -1" なのか、 "-4 余り 1" なのか。割る数×商＋剰余が割られた数にならない場合もある（ひどい)。)

4 浮動小数点数 (floating-point numbers)

> 「実際の計算機で浮動小数点表現が実現される形は、ここで議論する理想的なものとは異なるかもしれない。しかしその差異は小さく、**丸め誤差**という基本的な問題を扱う場合には常にほとんど無視することが出来る」 — Forsythe

数値計算で扱う数の範囲は極めて広い。整数や固定小数点数では不便である (ENIAC を開発した von Neumann はそれくらい何でもないと思ったらしいが)。科学で現れる数の表現には「科学的表記法」 (scientific notaion) (あるいは「指数形式」 (exponential form)) と呼ばれるものがあるが、これを取り入れたものが、以下に説明する**浮動小数点数** (floating point numbers) である。

以下に見るように、浮動小数点数の体系は数学的にはあまりきれいなものではなく、実用的な観点からも完璧なものからはほど遠いものである。

この体系とつき合うには、物理や化学の実験でおなじみの有効数字の概念を勉強した時の感覚が多少は役に立つが、かなり独特のものがあると思う。

基数 β を固定する時、0 でない任意の実数 x は

$$(*) \qquad x = \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \cdots \right) \times \beta^m$$

の形に表現することが出来る。ただし、$d_1, d_2, d_3, \cdots$ は

$$0 < d_1 \leq \beta - 1, \quad 0 \leq d_k \leq \beta - 1 \quad (k = 2, 3, \cdots)$$

を満たす整数である。($x = \pm\alpha \times \beta^m, \quad 1/\beta \leq \alpha < 1, m$ は整数)

そこで計算機の内部では上式を有限項で打ち切った

$$\pm\left(\frac{d_1}{\beta}+\frac{d_2}{\beta^2}+\frac{d_3}{\beta^3}+\cdots+\frac{d_n}{\beta^n}\right)\times\beta^m$$

で表現する。ただし m の範囲にも制限がつく： $-m_{_L}\le m\le m_{_U}$。

これを β 進 n 桁浮動小数点表示と呼び、この形に表現される数を浮動小数点数 (floating-point numbers) という。

現在の実際の計算機で採用されている基数は、主として 2, 10, 16 である (パソコン、ワークステーションでは 2、スーパーコンピューターを除く大型機では 16 が多い)。

通常は最上位の桁 d_1 は (上に書いたように) 0 にならないようにしておく。この条件を満たしている表示を正規化 (normalize) された浮動小数点表示という。

$$f=\frac{d_1}{\beta}+\frac{d_2}{\beta^2}+\frac{d_3}{\beta^3}+\cdots+\frac{d_n}{\beta^n}$$

を仮数部 (mantissa) または小数部 (fraction)、β^m を指数部、m を指数 (exponent) という。

0 は浮動小数点表示においては特別扱いする。通常はその仮数部を all zero ($d_1=d_2=\cdots=d_n=0$)、指数 $m=-m_L$ とすることにより表現する。

条件 $x_1\neq 0$, $-m_{_L}\le m\le m_{_U}$ を満たす全ての ($*$) に、この 0 を加えたもので一つの浮動小数点の体系 F が出来る。

例 (ミニ浮動小数点体系). $\beta=2, n=3, m_{_L}=1, m_{_U}=2$ とすると？是非各自で F の要素を書き出して、数直線上にメモって欲しい。(まず原点に関して対称。最大の絶対値 $(1/2+1/4+1/8)2^2=7/2$. 0 でない最小の絶対値 $(1/2)2^{-1}=1/4$. 各々の区間 $[1/4,1/2),[1/2,1),[1,2),[2,4)$ では等間隔。)

数直線上に表してみると、左右に広漠な空きがあること、またゼロの付近もひどく空いていることに気付く。これは一般的な特徴である。

実際の計算機では、符号、仮数部、小数部のそれぞれを 2 進数 (ビットの列) で表すことになる ($\pm\beta^n f$ が前節で説明した方法で表現される)。

SPARCstation 上の FORTRAN では、単精度 (32bits) の場合、符号 1 ビット、指数部 7 ビット、仮数部 24 ビット (ただしいわゆるケチ表現をしている)。倍精度の場合は、符号 1 ビット、指数部 10 bit、仮数部 53 bit である。

問題点

(i) 絶対値が極端に大きい数、極端に小さい数は表現できない。入力が出来ないのはもちろんであるし、 演算の結果がそうなることもある（オーバーフロー (overflow)、アンダーフロー (underflow))。

(ii) F は有限集合なので、ほとんどすべての数は近似的にしか表現できない。入力の際も演算の際も近似をする必要がある。例. $\beta=1/2$ では $\frac{1}{10}$ すら表現できない。

$$\frac{1}{10}=(0.0001100110011001100\cdots)_2$$

(iii) 演算に伴う丸め誤差, 情報落ち, 狭い意味の丸め（演算の結果は F に属さないので適当に丸める)。

(iv) 桁落ち

大雑把に言って、**浮動小数点数の体系は相対誤差を一定量で押えるようになっている**。 F に属する数の絶対値の最大値を U, $F \setminus \{0\}$ に属する数の絶対値の最小値を L とし、$D(F) = \{x; L \leq |x| \leq U\}$ とおく。$x \in D(F)$ に対し、

$fl(x) =$ “x に最も近い F の元（2 つある時はどちらか一方を適当なルールで決める）”

とおく。$fl(x)$ は x を近似的に表現する F の要素で最良のものだと考えられる。このとき

$$\left| \frac{fl(x) - x}{x} \right| \leq \frac{1}{2} \beta^{1-n}$$

演算の際の丸めについて。 $x, y \in F$ としても $x + y \in F$ であるとは限らない。計算機の中の加算 $\oplus$ はうまく実現した場合で

$$x \oplus y = fl(x + y)$$

が成立する。 $x \oplus y$ と $x + y$ の差を浮動小数点加算 $\oplus$ で生じた丸め誤差と呼ぶ。

$\oplus$ はあまり良い性質を持たない。(F が体のような代数系にならないのはまあ仕方ないにしても）可換則は OK であるが、結合則や分配則は成立しない。このことは誤差の解析を難しくする原因になっている。

例. $s_N = \sum_{n=1}^{N} \frac{1}{n}$ を計算することを考える。$N = 100000$ とした計算例。

```
oyabun% test1
s=1.209085083007812e+01          n が小さい方から足していった（単精度）
s=1.209015274047852e+01          n が大きい方から足していった（単精度）
s=1.209014612986334e+01          n が小さい方から足していった（倍精度）
S=1.209014612986341e+01          n が大きい方から足していった（倍精度）
```

consider $1.0 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + \cdots + 0.010$

これは情報落ち（桁揃えの段階で仮数部が捨てられる）と呼ばれる現象。

計算機イプシロン (machine epsilon) 計算機の精度を特徴付ける量として、計算機イプシロンと呼ばれるものがある。普通、$1 + \varepsilon$ が 1 より大きくなるような ε のうちで最小のもの、と定義される:

$$\varepsilon_M \stackrel{\text{def.}}{=} \min\{\varepsilon \in F; 1 \oplus \varepsilon > 1\}.$$

厳密な値を知らなくても、大体の値が分かれば十分。次のような FORTRAN プログラムの断片で調べることが出来る。

```
      EPS = 1.0
 10   EPS = EPS * 0.5
      EPSP1 = EPS + 1.0
      IF (EPSP1 .GT. 1) GOTO 10
```

演習 自分が使える言語処理系で machine epsilon を求めよ。単精度、倍精度、両方求めること。

その他の実数表現法

上で説明した浮動小数点数以外の実数表現法も提唱されている。特に溢れを回避するための工夫として、指数部可変長の形式（松井正一、伊理正夫)、URR（universal representation of real numbers, 浜田穂積）などがある。

計算結果の精度保証をするための区間演算と、それをサポートするための実数表現法も重要な話題である。

IEEE Standard 754

浮動小数点数については、IEEE（アメリカ電気電信技術者協会のこと）の規格が、有名かつ重要である。特に 1985 年に公表された P754 という 2 進浮動小数点数演算規格は、現在のパソコン、ワークステーションのマイクロプロセッサーに広く採用されているので、特に微妙な計算を必要とする場合は、規格を学ぶ価値がある。この規格では、∞ 等の NAN (not-a-number, 非数)、不正規化数の導入、例外処理等の工夫がある。(もちろんデータの可搬性も保証される。)

IEEE 単精度 指数 8, 仮数 24 最小の非正規数 1.401e-45 最小の正規数 1.175e-38 最大の数 3.403e+38 b31（符号 s）, b30..b23（指数 e）,b22..b0（仮数部 f）(implicit MSB があるので、仮数部は実質 24 bits) 正規数値 ($0 < e < 255$)

$$(-1)^s \times 2^{e-127} \times 1.f$$

非正規数値 ($e = 0$)

$$(-1)^s \times 2^{e-126} \times 0.f$$

符号つきの 0 ($e = 0$)

$$(-1)^s \times 0$$

符号つきの ∞ ($e = 255$)

$$s = u; e = 255; f = .000 \cdots 000$$

非数 ($e = 255$) シグナルを発生するもの

$$s = u; e = 255; f = .0uuu \cdots uuu \quad \text{少なくとも 1 bit は 0 でない}$$

シグナルを発生しないもの

$$s = u; e = 255; f = .1uuu \cdots uuu$$

IEEE 倍精度 指数 11, 仮数 53 最小の非正規数 4.941e-324 最小の正規数 2.225e-308 最大の数 1.798e+308 b63（符号 s）, b62..b52（指数 e）,b51..b0（仮数部 f）(implicit MSB があるので、仮数部は実質 53 bits) 正規数値 ($0 < e < 2047$)

$$(-1)^s \times 2^{e-1023} \times 1.f$$

非正規数値 ($e = 0$)

$$(-1)^s \times 2^{e-1022} \times 0.f$$

符号つきの 0 ($e = 0$)

$$(-1)^s \times 0$$

符号つきの ∞ ($e = 2047$)

$$s = u; e = 2047; f = .000 \cdots 000$$

非数 ($e = 2047$) シグナルを発生するもの

$$s = u; e = 2047; f = .0uuu \cdots uuu \quad \text{少なくとも 1 bit は 0 でない}$$

シグナルを発生しないもの

$$s = u; e = 2047; f = .1uuu \cdots uuu$$

NaN – Not a Number の略。日本語では非数と訳される。

denormalized number （非正規数） subnormal number の古い言い方。

subnormal number (非正規数) この規格では、下駄ばき表現の指数が 0 になっている、ゼロでない浮動小数点数のこと。

exception（例外） 算術例外とは、ある算術演算を試みた時に、一般的に受け入れられる結果が生成されないことを意味する。

gradual underflow （漸近的アンダーフロー） 浮動小数点演算がアンダーフローした時、0 の代わりに非正規化数を返すこと。

	大きい数×大きい数	+Inf	オーバーフロー
	大きい数× (-大きい数)	-Inf	オーバーフロー
	正数/0.0	+Inf	0 除算
IEEE 例外	負数/0.0	-Inf	0 除算
	0.0/0.0	NaN	演算不可能
	小さい数/大きい数	非正規化数	アンダーフロー
	2.0/3.0	丸めが起こる	結果不正確

IEEE 拡張倍精度 指数 15, 仮数 64

IEEE 4 倍精度 指数 15, 仮数 113 最小の非正規数 6.475e-4966 最小の正規数 3.362e-4932 最大の数 1.190e+4932