位取り記数法と2進数・16進数

山本昌志*

2007年2月2日

概 要

位取り記数法を説明し，2進数や16進数での整数の表し方を述べる．

1　本日の内容

いよいよC言語の最難関であるポインターの学習を始めることになる．ポインターについての深い知識を得ようとすると，メモリーのことを理解しなくてはならない．メモリーのことを理解するためには，2進数と16進数が分からなくてはならない．そこで，本日は2進数と16進数について説明する．

諸君の大部分の者は，コンピューター内部では2進数が使われていることを知っていると思う．2進数が使われているので，それの取り扱いになれる必要がある．16進数は2進数の親戚で，表示に便利なのでコンピューターの世界でよく使われる．そのため，16進数も学習する．コンピューターで2進数が使われる理由は，付録Aに示してある．

2　位取り記数法

2.1　数の表現

通常使われている10進数では，0〜9までの10個の数字を使って，数を表現する．9の次は10で桁が上がる．10進数以外にいろいろな数の数え方がある．2進数，10進数，16進数の数の表現を図1に示す．合わせて，桁上がりという考えの無い漢字や楔形文字，ローマ数字も併記する．桁上がりという考えが無い表記の場合，桁が上がるたびに，新しい記号が必要であることがわかる．大きな数字を表す場合，非常に不便である．また，筆算を用いた計算もできない．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

原始人	2進法	10進法	16進法	漢字	楔形文字	ローマ数字
	0	0	0			
●	1	1	1	一	𒁹	I
●●	10	2	2	二	𒁹𒁹	II
●●●	11	3	3	三	𒁹𒁹𒁹	III
●●●●	100	4	4	四	𒐈	IV
●●●●●	101	5	5	五	𒐈𒁹	V
●●●●●●	110	6	6	六	𒐈𒁹𒁹	VI
●●●●●●●	111	7	7	七	𒐈𒁹𒁹𒁹	VII
●●●●●●●●	1000	8	8	八	𒐈𒐈	VIII
●●●●●●●●●	1001	9	9	九	𒐈𒐈𒁹	IX
●●●●●●●●●●	1010	10	A	十	𒌋	X
●●●●●●●●●● ●	1011	11	B	十一	𒌋𒁹	XI
●●●●●●●●●● ●●	1100	12	C	十二	𒌋𒁹𒁹	XII
●●●●●●●●●● ●●●	1101	13	D	十三	𒌋𒁹𒁹𒁹	XIII
●●●●●●●●●● ●●●●	1110	14	E	十四	𒌋𒐈	XIV
●●●●●●●●●● ●●●●●	1111	15	F	十五	𒌋𒐈𒁹	XV
●●●●●●●●●● ●●●●●●	10000	16	10	十六	𒌋𒐈𒁹𒁹	XVI
●●●●●●●●●● ●●●●●●●	10001	17	11	十七	𒌋𒐈𒁹𒁹𒁹	XVII
●●●●●●●●●● ●●●●●●●●	10010	18	12	十八	𒌋𒐈𒐈	XVIII
●●●●●●●●●● ●●●●●●●●●	10011	19	13	十九	𒌋𒐈𒐈𒁹	XIX

図 1: 数の数え方

2.2 現代の数の表現 (位取り記数法)

現代の便利な数の表現は桁上がりの考えがあるからこそである．この桁上がりの考えは，ゼロが発見されたので可能となった．このように桁上がりの考えで，数を示すのが位取り記数法 (place value sysytem) である．10 進数は 9 の次で桁上がりが生じ 10 となる．0〜9 の数字を使って，数を表すのである．この 10 を基数と，0〜9 間での数を底と言う．10 進数の他，いろいろの基数の数が考えられるが，コンピューター科学で使われるのは，主に 2 進数と 16 進数である．それぞれの基数と底を表 1 に示す．

表 1: 基数と底

数の表現	基数	底
2 進法	2	0, 1
10 進法	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 進法	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

図 1 や表 1 から，自然数の数え方は理解できたと思う．そうすると相互の変換ができれば，ある程度それを応用することができるようになる．それぞれの変換を考える前に，数の表記方法について，勉強することにする．位取り記数法での数の表し方が理解できれば，それぞれの変換が分かるはずである．

例えば，今年は 2007 年である．10 進法の 2007 の表記はどのような意味があるか?．これは，次のように解釈する．大げさではあるが，こう解釈すると，他の基数の数字の意味はっきりする．

$$(2007)_{10} = (2 \times 10^3 + 0 \times 10^2 + 0 \times 10^2 + 7 \times 10^0)_{10} \tag{1}$$

括弧の下の 10 は 10 進法の意味で，非常に簡単な話である．これさえ，分かれば基数の変換なんか，怖くない．この式の右辺の $\times$ と 10^p をとって，それを並べたのが位取り記数法である．

コーヒーブレイク

ゼロがない時代は，大変だった．ゼロが無いと，式 (1) の左辺のような表記は出来ない．その 0(ゼロ) が発見されたのは，6 世紀頃のインドと言われている．西暦 0 年がないのは，このためである．キリストが生まれた頃は，ゼロがなかったのである．

3 基数の変換

3.1 2 進数と 10 進数の関係

3.1.1 2 進数 → 10 進数

2 進数から 10 進数への変換は簡単である．式 (1) を理解していれば，分かる．2 進数であろうが 10 進数であろうが，表記法は同じで，約束に従って変形すれば良い．次のようにする．

$$\begin{aligned}(10011)_2 &= (1 \times 10^{100} + 0 \times 10^{11} + 0 \times 10^{10} + 1 \times 10^1 + 1 \times 10^0)_2 \\ &= (1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} \\ &= (16 + 0 + 0 + 2 + 1)_{10} \\ &= (19)_{10}\end{aligned} \tag{2}$$

順を追って説明すると，以下のようになる．

1. まずは，1 行目右辺のように位取り記数法で表現する．
2. そうして，表 1 に従い，式を変換したい基数に直す．
3. 後は地道に計算するだけ．

通常は，1 行目の右辺は省き，2 行目から計算する．

- 2 進数の各桁の 10 進数での値 (重み) を覚えておくと便利である．
  1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096

3.1.2　10 進数 → 2 進数

今度は逆で 10 進数から 2 進数への変換である．原理的に，先ほどと同じように変換ができるが，計算してみるとそれは難しい[1]．これは，我々は 10 進数の演算になれていることが原因となっている．自然では，10 進数であろうと 2 進数であろうと優位性は無い．

10 進数から 2 進数へ変換する計算は簡単であるが，その内容を理解することが大事である．計算手法を忘れても，内容が理解できていれば，その方法はいつでも自分で作ることができる．また，応用範囲も広がる．では，簡単な例で説明する．10 進数の $(19)_{10}$ を 2 進数に変換する方法を示す．具体的には，19 を

$$(19)_{10} = (\cdots + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0)_{10} \tag{3}$$

と表現したい．これは式 (2) の 2 行目の式で，ここで求められた係数を $(\cdots a_4 a_3 a_2 a_1 a_0)_2$ と並べれば位取り記数法になる．それぞれ，a_n を求めなくてはならない．そこで，次のように 19 を 2 で割った商と余りを考える．これは，

$$(9 \times 2 + 1)_{10} = (\cdots + a_4 \times 2^3 + a_3 \times 2^2 + a_2 \times 2^1 + a_1 \times 2^0)_{10} \times 2 + a_0 \tag{4}$$

と書ける．これをよくにらむと，$a_0 = 1$ ということが分かる．すなわち，a_0 は 19 を 2 で割ったあまりである．残りの部分は，

$$(9)_{10} = (\cdots + a_4 \times 2^3 + a_3 \times 2^2 + a_2 \times 2^1 + a_1 \times 2^0)_{10} \tag{5}$$

となることも分かるでろう．商について同じことをすると，

$$(4 \times 2 + 1)_{10} = (\cdots + a_5 \times 2^3 + a_4 \times 2^2 + a_3 \times 2^1 + a_2 \times 2^0)_{10} \times 2 + a_1 \tag{6}$$

となる．したがって，$a_1 = 1$ である．しつこいが，さらに商について同様に進めると，

$$(2 \times 2 + 0)_{10} = (\cdots + a_6 \times 2^3 + a_5 \times 2^2 + a_4 \times 2^1 + a_3 \times 2^0)_{10} \times 2 + a_2 \quad \Rightarrow \quad a_2 = 0 \tag{7}$$

$$(1 \times 2 + 0)_{10} = (\cdots + a_7 \times 2^3 + a_6 \times 2^2 + a_5 \times 2^1 + a_4 \times 2^0)_{10} \times 2 + a_3 \quad \Rightarrow \quad a_3 = 0 \tag{8}$$

$$(0 \times 2 + 1)_{10} = (\cdots + a_8 \times 2^3 + a_7 \times 2^2 + a_6 \times 2^1 + a_5 \times 2^0)_{10} \times 2 + a_4 \quad \Rightarrow \quad a_4 = 1 \tag{9}$$

となる．最後の式から，$a_n = 0 \quad (5 \leq n)$ が分かる．以上をまとめると

$$\begin{aligned} (19)_{10} &= (a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0)_{10} \\ &= (1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} \\ &= (10011)_2 \end{aligned} \tag{10}$$

となる．要するに，2 で割ったあまりを書いていけば良いのである．計算方法は分かった．だがこの方法は実際的ではない．よく使われるのは，図 2 のように 2 で割った余りを並べる．これは，$(19)_{10} = (10011)_2$，$(2005)_{10} = (11111010011)_2$ を示している．

[1] 本当に難しいか，試して見よ

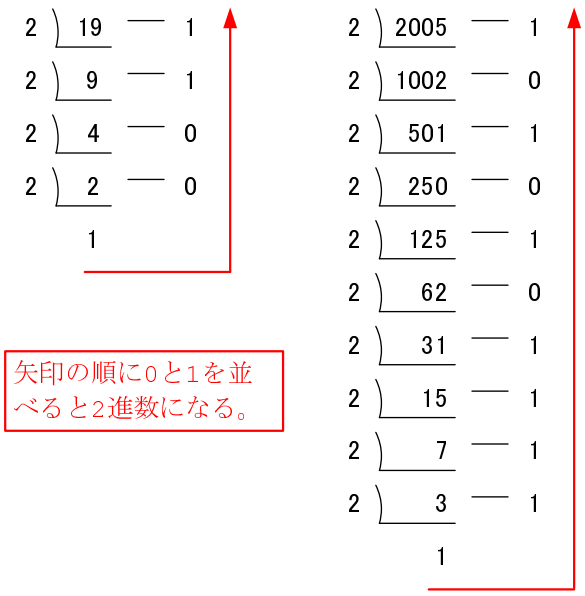

図 2: 10 進数から 2 進数への変換方法.

内容を十分理解し，変換の練習をしなくてはならない.

3.2 16 進数

2 進数の変換が理解できたら，16 進数の変換はまったくもって簡単である.

3.2.1 16 進数 → 10 進数

2 進数と同じで次のようにする.

$$\begin{aligned}(376)_{16} &= (3\times 10^2 + 7\times 10^1 + 6\times 10^0)_{16} \\ &= (3\times 16^2 + 7\times 16^1 + 6\times 16^0)_{10} \\ &= (3\times 256 + 7\times 16 + 6\times 1)_{10} \\ &= (886)_{10} \end{aligned} \tag{11}$$

3.2.2 10 進数 → 16 進数

これも 2 進数と同様に考える. 16 で割ったあまりを並べれば良い. 図 3 のようにして，$(25391)_{10} = (632F)_{16}$ を計算する.

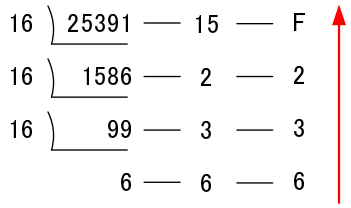

図 3: 10 進数から 16 進数への変換方法

手計算ではこの方法を用いるが，実際のエンジニアーは電卓の変換機能を使う．

コーヒーブレイク

昔から言われるジョークをひとつ．プログラマーは，クリスマス (12 月 25 日) とハロウィーン (10 月 31 日) が区別できない．なぜか?．ヒント

- 10 進数 (decimal number) のことを DEC と書く．DEC 23 と書けば，10 進数の 23 をあらわしている．
- 8 進数 (octal number) はのことを OCT とかく．OCT 23 と書けば，8 進数の 23 を表している．

3.3 2 進数と 16 進数の変換

2 進数と 16 進数の相互の変換は簡単である．$2^4 = 16$ なので，2 進数の 4 桁は 16 進数の一桁に対応している．図 4 のように，1 桁の 16 進数を 4 桁の 2 進数に変換すれば良い．反対に 4 桁の 2 進数は，1 桁の 16 進数に変換できる．

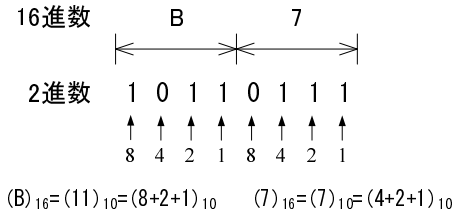

図 4: 2 進数と 16 進数の相互変換方法

4 課題

次の講義 (2 月 9 日) の AM8:45 までに，以下の課題をレポートとして提出すること．表紙等は，いつもの通り．表紙のタイトルは「位取り記数法と 2 進数・16 進数」とすること．

[**問 1**] (予) 教科書 p.270–292 を 2 回読め．レポートには「2 回読んだ」と書け．

[問 2] (復) 本日配布したプリントを 2 回読め．ただし，付録は興味のある者のみでよい．レポートには「2 回読んだ」と書け．そして，誤字脱字，日本語の文章のおかしなところ，間違いがあれば，レポートに記述せよ．

[問 3] (復) 以下の 10 進数を 2 進数と 16 進数へ変換せよ．

$(1)_{10}$	$(2)_{10}$	$(4)_{10}$
$(8)_{10}$	$(16)_{10}$	$(32)_{10}$
$(65536)_{10}$	$(2004)_{10}$	$(999)_{10}$

[問 4] (復) 以下の 2 進数を 10 進数と 16 進数へ変換せよ．

$(1)_2$	$(10)_2$	$(100)_2$
$(1000)_2$	$(11111)_2$	$(10101111)_2$
$(10010101)_2$	$(10101010)_2$	$(11111111)_2$

[問 5] (復) 以下の 16 進数を 2 進数と 10 進数へ変換せよ．

$(8)_{16}$	$(F)_{16}$	$(1F)_{16}$
$(AF)_{16}$	$(F98)_{16}$	$(89AB)_{16}$
$(CDEF)_{16}$	$(FFFF)_{16}$	$(A000)_{16}$

付録 A　コンピューターで 2 進数が使われる理

付録 A.1　2 進数のメリット

人間の指は 10 本あることは，よく知られている．そのため，人類は 10 進法を使っていると言われている．小学校の低学年では指を使って計算する子供がいることからも分かる．コンピューターの内部のハードウェアーでは，電圧が 0V か 5V(もっと低い場合もある) でデータやプログラムを表現している．指が 2 本しかないのと同じ．だから，コンピューターは 2 進法を使う．2 進法を使うメリットに，何があるか? という疑問が湧くであろう．その答えとして，以下のようなことが考えられる．

- ノイズに強い

  0〜5V で動作する素子からできたコンピューターを考える．2 ビットと 10 ビットの場合，割り当てられる電圧のレベルは，図 5 の通りである．図からも分かるように，許されるノイズは，2 ビットの方が格段に大きくなる．1 ビットのエラーも許されないデジタルコンピューターにおいては，この差は非常に大きい．

- ハードウェアーを実現するのが容易

  コンピューター内部には，単純な動作をする同じような部品が数多くある．2 進数であれば，入力は 0 と 1，出力も 0 と 1 なので，構成する 1 個の部品が非常に単純になる．要するに 2 進数を採用すると，部品が簡単になるのである．

- 演算が簡単

  例えば，掛け算九九を考えると分かる．10 進数だと，0〜9 までの掛け算，合計 100 通りある．2 進数だと，4 通りで済む．

- ブール代数が使えて，論理演算が容易

  ブール代数については，他の授業で勉強することになっている．それまで待てない人は，私の講義ノート[2]でも見ろ．

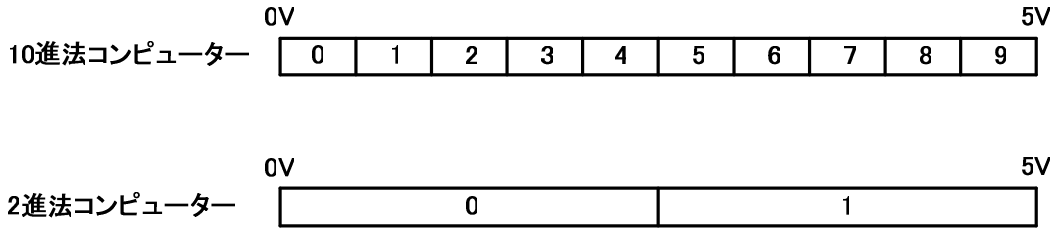

図 5: 2 進法と 10 進法のコンピューターのノイズレベル

[2] http://www.akita-nct.jp/ yamamoto/lecture/2003/2E/boolean_algebra/index.html

付録 A.2　2 進数のデメリット

それでは，2 進数のデメリットとはどのようなことが考えられるであろうか? すぐに分かることは，桁数が多くなることである．例えば，十進法の $(99)_{10}$ は，二進法では $(1100011)_2$ となる．十進法であれば 2 桁で済むのに，2 進法であれば 7 桁も必要になる．

このことは，コンピューターが発明された当初問題とされたが，すぐにこれは間違いだと気づく．ある正の整数 k をそれぞれの位取記数法で表した場合，その底の数 α と桁数 β を表 2 にしめす．n 進数の場合，整数 k を表すの必要な桁数 $\log_n k$ は次のようにして理解できる．n 進数が β 桁あると，それが表すことができる組み合わせの数は，n^β となる．これが，s 整数 k まで表すことができるから，

$$n^\beta = k \tag{12}$$

となる．したがって，必要な桁数は，

$$\beta = \log_n k \tag{13}$$

となる．

表 2: 整数 k を表すときの底と桁数

底の数 α	桁数 β
1	k
n	$\log_n k$
k	1

それでは，一番効率のよい底の数はいくつであろうか?．効率の定義はいろいろ出来るが，ここでは

- 底の数と桁数で評価することにする．ある整数を表す場合，これらの数の合計が小さいことが，効率がよい．

とする．n 進数で正の整数 k を表す場合，効率は，

$$\alpha + \beta = n + \log_n k \tag{14}$$

となる．これが最小となるのは，n で微分[3]したときゼロとなる，

$$\begin{aligned} \frac{\mathrm{d}(\alpha+\beta)}{\mathrm{d}n} &= 1 - \frac{\log k}{n(\log n)^2} \\ &= 0 \end{aligned} \tag{15}$$

である．この方程式の解，すなわちもっとも効率の良い底を図 6 に示す．この図から分かるように，比較的小さな数字 $(\leq 10^5)$ では 4 進数あたりが効率がよい．大きな数になると 10 進数程度が最適な底となる．コ

[3] これはまだ学習していないがカンベン．2 年生の時に学習するのでその時読み返せば良い．

ンピューターが扱う最大の数は，大体 2^{32} 程度[4]である．これだと，2 進数で 32 桁，10 進数で 10 桁である．この場合，図から分かるようにもっとも効率の良いのが 6 進数であるが，これだと，13 桁が必要である．2 進数，6 進数，10 進数をつかっても，桁数は 10〜32 である．コンピューターにとって，32 桁を取り扱うことは簡単なことなので，2 進数で数字を表現しても全く問題ない．10 進数をつかうことに比べて 3 倍程度の桁数の増加にしかすぎないのである．2 進数を使うことによる桁数の増加は，さほど大きなデメリットではない．それよりも，2 進数を使うメリットの方が圧倒的に多い．

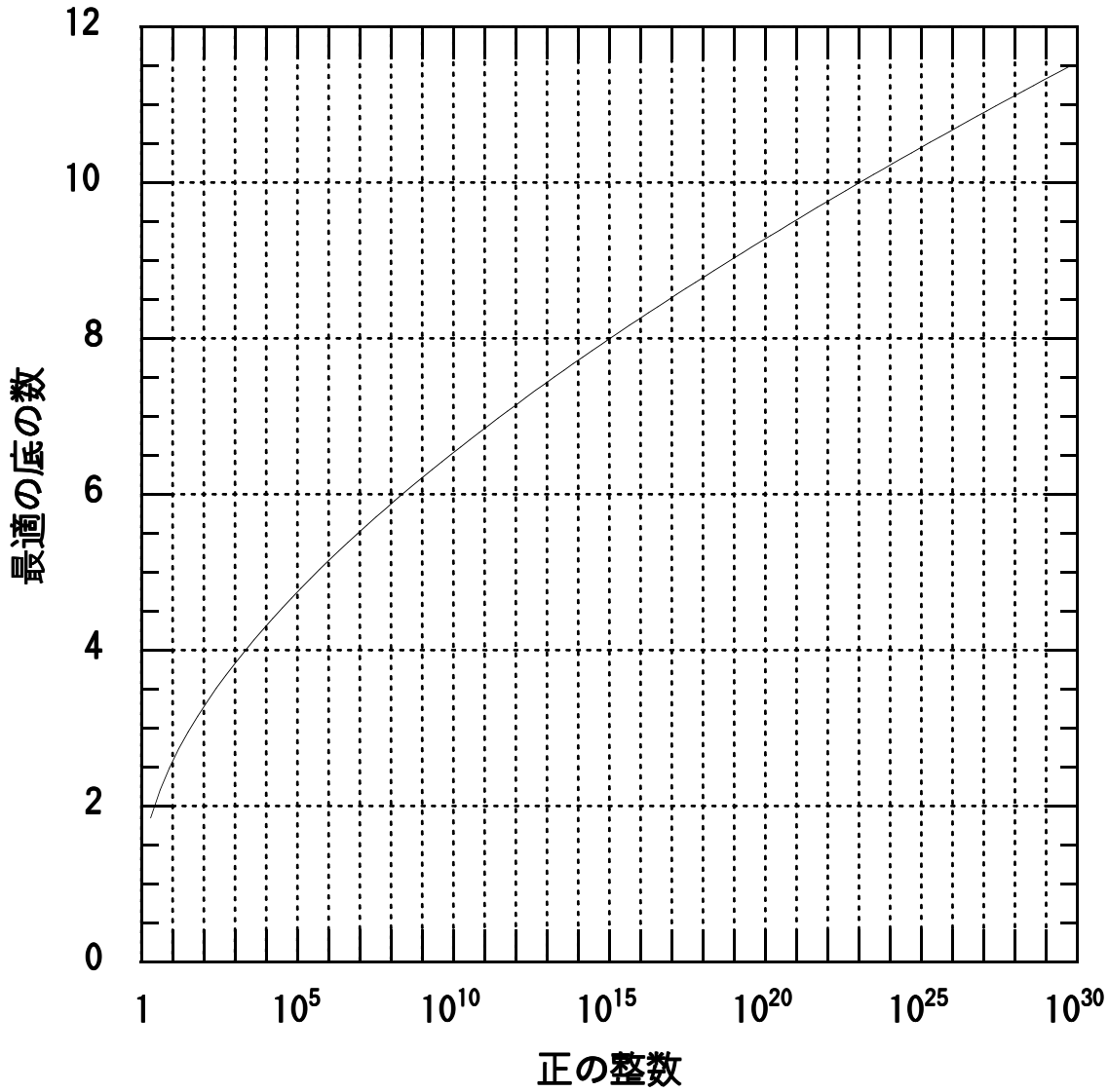

図 6: もっとも効率の良い底．横軸は整数で，縦軸はその整数を表すときのもっとも効率の良い底．

[4]C 言語の int 型の最大値

付録 B　2 進数を用いた小数の表現

付録 B.1　位取り記数法

10 進数での小数の表現を考える．例えば，次の小数は

$$(0.1235)_{10} = (1 \times 10^{-1} + 2 \times 10^{-2} + 3 \times 10^{-3} + 5 \times 10^{-4})_{10} \tag{16}$$

となっており，整数の場合と同じである．小数点を境に，右側の指数部が-1, -2, -3 と 1 ずつ減少する．これは，先に示した整数の場合と全く同じで，簡単である．当然，

$$\begin{aligned} 10^{-1} &= \frac{1}{10^1} = \frac{1}{10} = 0.1 \\ 10^{-2} &= \frac{1}{10^2} = \frac{1}{100} = 0.01 \\ 10^{-3} &= \frac{1}{10^3} = \frac{1}{1000} = 0.001 \\ &\vdots \\ 10^{-N} &= \frac{1}{10^N} \end{aligned} \tag{17}$$

は理解していなくてはならない．

付録 B.2　基数の変換 (2 進数 →10 進数)

2 進数での少数の表記も，10 進数の場合と同じである．だから，2 進数少数を 10 進数少数に変換するのは簡単である．たとえば，

$$\begin{aligned} (0.10101)_2 &= (1 \times 10^{-1} + 0 \times 10^{-10} + 1 \times 10^{-11} + 0 \times 10^{-100} + 1 \times 10^{-101})_2 \\ &= (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5})_{10} \\ &= (1 \times 0.5 + 0 \times 0.25 + 1 \times 0.125 + 0 \times 0.0625 + 1 \times 0.03125)_{10} \\ &= (0.5 + 0.125 + 0.03125)_{10} \\ &= (0.65625)_{10} \end{aligned} \tag{18}$$

となる．当然

$$\begin{aligned} 2^{-1} &= \frac{1}{2^1} = \frac{1}{2} = 0.5 \\ 2^{-2} &= \frac{1}{2^2} = \frac{1}{4} = 0.25 \\ 2^{-3} &= \frac{1}{2^3} = \frac{1}{8} = 0.125 \\ &\vdots \\ 2^{-N} &= \frac{1}{2^N} \end{aligned} \tag{19}$$

は理解していなくてはならない．

付録 B.3　基数の変換 (10 進数 →2 進数)

つぎは，先ほどと逆を考える．たとえば，先ほどの例の $(0.65625)_{10}$ を 2 進数で表現する．そのためには，

$$(0.65625)_{10} = (a_1 \times 2^{-1} + a_2 \times 2^{-2} + a_3 \times 2^{-3} + \cdots)_{10} \tag{20}$$

と書き直せばよい．ただし，a_n は 0 または 1 である．そして，この a_n を並べると，

$$(0.65625)_{10} = (a_1 a_2 a_3 a_4 \cdots)_2 \tag{21}$$

と 2 進数で表現できる．ここで，問題は a_n を求めることである．そこで，式 (20) の両辺を 2 倍する．すると，

$$(1.3125)_{10} = (a_1 \times 2^{0} + a_2 \times 2^{-1} + a_3 \times 2^{-2} + \cdots)_{10} \tag{22}$$

となる．この式の両辺の整数部と小数部は等しいので，

$$(1)_{10} = (a_1)_{10} \qquad (0.3125)_{10} = (a_1 \times 2^{0} + a_2 \times 2^{-1} + a_3 \times 2^{-2} + \cdots)_{10} \tag{23}$$

となる．これで，$a_1 = 1$ が求まった．同じ ように，残りの小数部分を 2 倍すると，

$$(0.625)_{10} = (a_2 \times 2^{0} + a_3 \times 2^{-1} + a_4 \times 2^{-2} + \cdots)_{10} \tag{24}$$

となる．これも，両辺の整数部と小数部が等しいので，

$$(0)_{10} = (a_2)_{10} \qquad (0.625)_{1}0 = (a_3 \times 2^{-1} + a_4 \times 2^{-2} + a_5 \times 2^{-3} + \cdots)_{10} \tag{25}$$

が得られる．これで，$a_2 = 0$ が求まった．同様に以下の通り，残りの小数部分の計算を進めると，全ての a_n が求まる．

$$\begin{cases} (1.25)_{10} = (a_3 \times 2^{-1} + a_4 \times 2^{-2} + a_5 \times 2^{-2} + \cdots)_{10} \\ (1)_{10} = (a_3)_{10} \qquad (0.25)_{10} = (a_4 \times 2^{-1} + a_5 \times 2^{-2} + a_6 \times 2^{-2} + \cdots)_{10} \end{cases} \tag{26}$$

$$\begin{cases} (0.5)_{10} = (a_4 \times 2^{-1} + a_5 \times 2^{-2} + a_6 \times 2^{-2} + \cdots)_{10} \\ (0)_{10} = (a_4)_{10} \qquad (0.5)_{10} = (a_5 \times 2^{-1} + a_6 \times 2^{-2} + a_7 \times 2^{-2} + \cdots)_{10} \end{cases} \tag{27}$$

$$\begin{cases} (1.0)_{10} = (a_5 \times 2^{-1} + a_6 \times 2^{-2} + a_7 \times 2^{-2} + \cdots)_{10} \\ (1)_{10} = (a_5)_{10} \qquad (0.0)_{10} = (a_6 \times 2^{-1} + a_7 \times 2^{-2} + a_8 \times 2^{-2} + \cdots)_{10} \end{cases} \tag{28}$$

最後に，小数部がゼロとなったので計算は，完了となる．以上をまとめると

$$\begin{aligned} (0.65625)_{10} &= (a_1 \times 2^{-1} + a_2 \times 2^{-2} + a_3 \times 2^{-3} + \cdots)_{10} \\ &= (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} +)_{10} \\ &= (0.10101)_2 \end{aligned} \tag{29}$$

となる．要するに，小数部を 2 倍して，その整数部を書いていけばよい．

普通は，図 7 のようにして計算を進める．2 倍して，整数部を書き出して，小数部を再度 2 倍する．これを繰り返すと，10 進数小数を 2 進数小数に変換することができる．10 進数の 0.1 は循環小数ではないが，2 進数にすると，

$$(0.1)_{10} = (0.00011001100110011\cdots)_2 \tag{30}$$

と循環小数になる．通常は，途中まで (必要な精度まで) で計算を打ち切る．

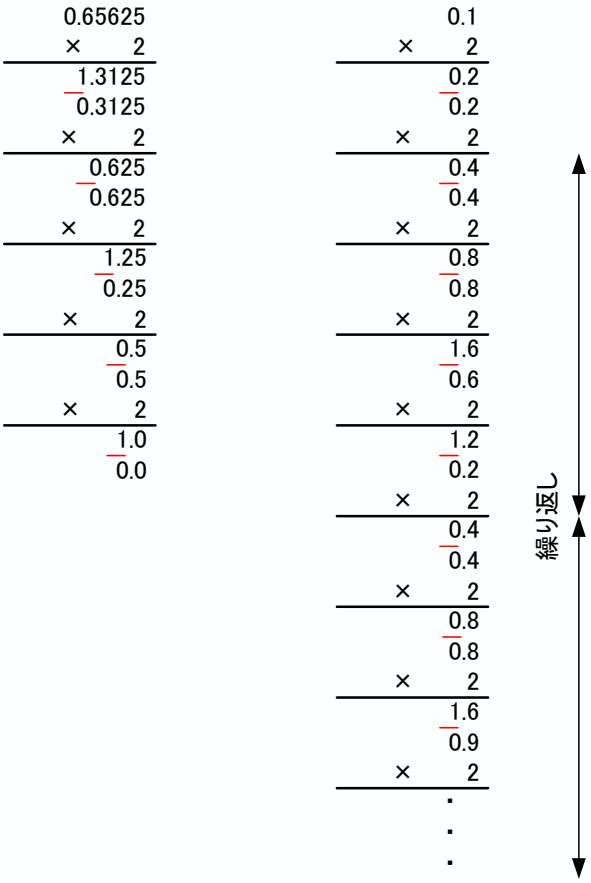

図 7: 小数の基数変換 (10 進数 →2 進数)

付録 C　コンピューター内部での正の整数以外の表現

これは少しレベルの高い内容で，もう少し時間をかけないと説明できない．将来，情報関係の仕事に就きたい者，あるいは自分の能力に自信のある者は独力で以下の内容を理解せよ．非常におもしろい内容であるはずである．

付録 C.1　負の整数の表現

普通の C 言語の整数型 (int) は，4 バイト (32 ビット) であるが，ここでは図を簡単にするため，16 ビットで説明する．32 ビットでも同じである．

負の整数は，補数 (complement) を使って，コンピューター内部では表現される．それを図 8 に示すが，手順は，次の通りである．

1. 絶対値を 2 進数のビットパターンで表現し，その反転を行う．
2. 反転されたビットパターンに 1 を加算する．

このようにしてできたビットパターンをメモリーに記憶させ，それを負の数として取り扱う．

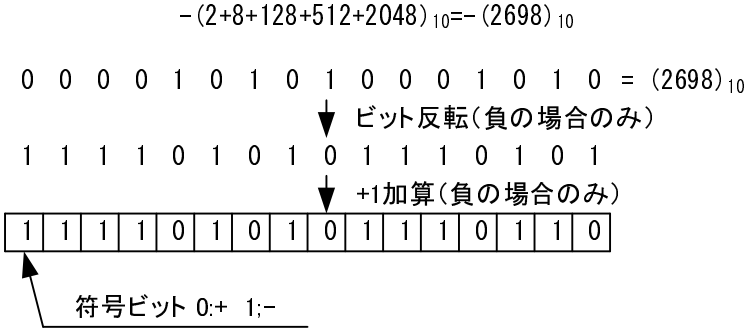

図 8: 負の整数をメモリーに格納する方法

この方法のメリットは，減算が加算器でできることである．補数表現のイメージは，図 9 の通りである．車の距離計に似ている．

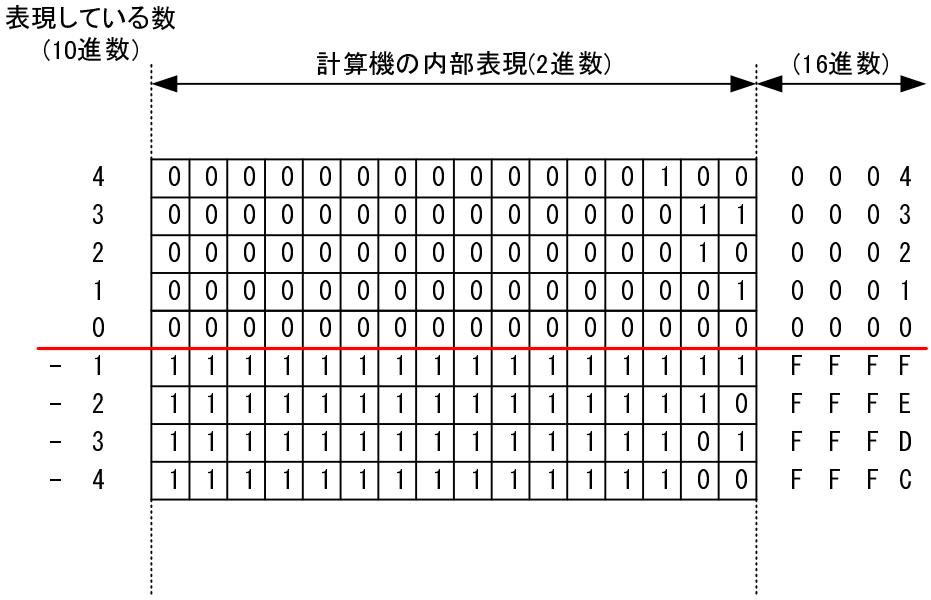

図 9: 距離計イメージ (2 の補数表示)

それでは，なぜ，補数表現だと，減算が加算器で可能なのだろうか?．減算の演算は，負の数の加算と同じである．したがって，図 9 のように負の数を表現すると，負の整数の加算は正の整数の加算と同じと分かるであろう．したがって，加算器で減算が可能となる．実際，正の数の減算を行うときは，ビットの反転と $+1$ 加算を実施して，加算器で計算する．イメージは，図 9 の通りであるが，もう少し，理論的に説明をおこなうとしよう．ある正の整数を x とする．その負の数，$-x$ は補数表現では，

$$[-x] = (FFFF - x + 1)_{16} \tag{31}$$

となる．左辺の $[-x]$ が $-x$ の意味である．[　] の意味は，括弧内の負の整数を計算機内部の表現を表している．これは，私が作った表記なので，一般には用いられていない．右辺の $FFFF - x$ がビット反転になっている．ここでは，16 ビットで整数を表現しようとしているので，$FFFF$ から x を引いてビット反転させている．疑問に思う者は実際に計算して見よ．それに 1 を加えて，補数の表現としている．つぎに，ある整数 y を考えて，$y - x$ を計算してみよう．

$$[y - x] = (y + FFFF - x + 1)_{16} \tag{32}$$

$FFFF - x + 1$ は，あらかじめ計算されて，コンピューター内部のメモリーに格納されているので，$[y - x]$ は加算器で可能である．これは，あたりまえです．重要なことは，この結果が，負の場合，2 の補数表現になっており，正の場合，そのままの値になっていることである．

演算の結果，$y - x$ が負になる場合を考えよう．すると式 (32) は，

$$[y - x] = (FFFF - (x - y) + 1)_{16} \tag{33}$$

と変形できる．この場合，絶対値が $(x-y)$ なので，絶対値のビット反転と+1 加算となっていることが理解できる．つぎに，$y-x$ が正になる場合を考えましょう．すると式 (32) は，

$$\begin{aligned}[y-x] &= (y-x+FFFF+1)_{16} \\ &= (y-x+10000)_{16} \end{aligned} \tag{34}$$

となる．10000 は計算機内部では，桁上がりを示す．16 ビットの表示では無視される．したがって，内部の表現は，正しく表せる．

コーヒーブレイク

この方法で負の数を表すことは，1970 年頃には常識となったようである．驚いたことに，負の数をこの補数で表すアイディアは，パスカルが最初です．パスカルは，パスカリーヌという歯車式計算機を 1642 年頃に製作しています．そこの減算を加算器で行うために，補数というものを考えたようです．

付録 C.1.1　ビット反転と+1 加算の意味

x を正の整数として，$-x$ をコンピューターの内部で表現する場合，

1. x をビット反転する．
2. +1 加算する．

の操作で得られたものその内部表現になる．式で表すと，

$$[-x] = (FFFF-x+1)_{16} \tag{35}$$

である．この操作の意味を調べてみよう．結論から言うと，この操作は符号反転 (-1 乗算) の操作になっている．それを示すために，もう一度この操作を繰り返してみる．すると，

$$\begin{aligned}\{FFFF-(FFFF-x+1)+1\}_{16} &= (x)_{16} \\ &= [x] \end{aligned} \tag{36}$$

となる．このことから，この操作は，符号反転であることが理解できる．式 (35) は，x の符号反転を示しており，式 (36) は $-x$ の符号反転を示している．

式 (35) は，-x のコンピューターの内部表現を表している．従って，元の x を求めるためには，その逆の操作

1. 1 減算 (-1 加算) する．
2. ビット反転する．

をすればよく，式だと

$$\begin{aligned}[FFFF-\{(FFFF-x+1)-1\}] &= (x)_{16} \\ &= [x] \end{aligned} \tag{37}$$

となる．しかし，元の表現を得るためには，式 (36) の演算でも良いはずである．式 (37) を変形すると，容易に式 (36) を導くことができる．これらのことから，以下の結論を導くことができる．

- ビット反転と+1 加算の操作は，整数のコンピューターの内部での表現の符号反転である．
- この符号反転の反対の操作である 1 減算とビット反転の操作は，ビット反転と 1 加算と同じ操作である．

付録 C.2　浮動小数点表示

実際のコンピューターを用いた計算では，実数がよく使われる．ここでは，C 言語の倍精度実数型「`double`」で変数を宣言したときの，データの格納の仕方を示す．

付録 C.2.1　浮動小数点表示とは

浮動小数点表示とは，指数化（例えば，-0.123×10^{-2}）して数値を表現する．これは非常に便利な方法で，自然科学では多くつかわれる．コンピューターでも同様で，データが整数と指定されない限りこの浮動小数点が用いられる．実際，この仮数部の (-0.123) と指数の (-2) をメモリーに格納する．この方法の長所と短所は，以下の通りである．

長所　決められたビット数内で，非常に小さな数値から大きな数値まで表現可能になる．
短所　桁落ち誤差が発生する場合がある．

浮動小数点表示を学習するために，必要な言葉の意味は，図 10 の通りである．1 年生の数学の授業で学習したはず．

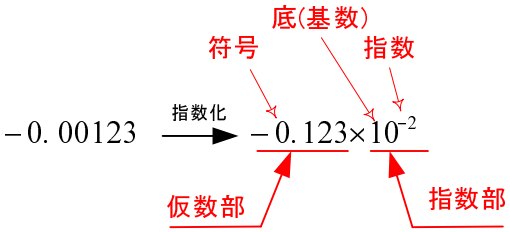

図 10: 指数表現の名称

付録 C.2.2　C 言語の倍精度実数型

IEEE の規格の C 言語の倍精度実数型の「`double`」の表現について説明する．まず，浮動小数点表示のための正規化を図 11 に示す．当然，仮数部，指数部とも 2 進数表現である．仮数部は，符号と `1.XXXX` のように表す．

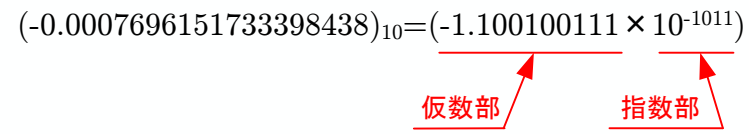

図 11: IEEE 規格表現のための規格化

つぎに，これを IEEE 規格の浮動小数点に表すことを考える．まずその規格の仕様は，以下のようになっている．

- 64 ビット (第 0 ビット〜第 63 ビット) で，浮動小数を表わす．各ビットの構成は，図 12 の通りである．
- 最上位の第 63 ビットが仮数部の符号ビットである．正の場合ゼロで，負の場合 1 になる．
- 指数は 11 ビットでオフセットバイナリ方式で表す．11 ビットで 0〜2047 の値になる．ただし，指数部 11 ビットの値 0 と 2047 は例外処理のために予約されている．11 ビットで表現される値からオフセット値 1023 を引くことにより指数の値が-1022〜1023 の範囲になるように定められている．
- 仮数部は 52 ビットである．小数点以下を，絶対値で表現する．規格化のための整数部は 1 と分かっているので，このためのビットは割り当てられていない．

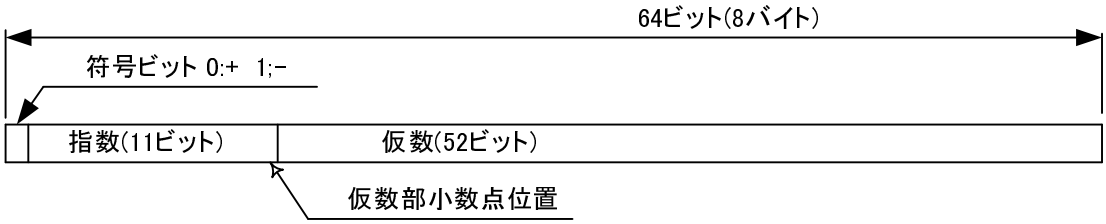

図 12: IEEE 規格 (C 言語の倍精度実数) 表現のビットの内訳

以上の仕様をもとに，図 11 で規格化された数を浮動小数点表示を示す．ほとんどの部分は規格化で分かるが，指数のみ計算が必要である．指数は，オフセットバイナリーで計算するために，まず 10 進数で表す．

$$(-1011)_2 = (-8 - 2 - 1)_{10} = (-11)_{10} \tag{38}$$

不動小数表示の指数は，この式の値に 1023 を加算して求める．すると，

$$(-11 + 1023)_{10} = (1012)_{10} = (1111110100)_2 \tag{39}$$

となる．

これで，すべて準備が整った．不動小数点表示は，図 13 のようになる．実際のコンピューターには，この 64 ビットのデータが格納される．メモリーは 8 ビット (1 バイト) 毎アドレスが割り当てられているので，8 番地分のデータ領域が必要である．

$$\left(-0.0007696151\ 733398438\right)_{10} = \left(-1.100100111 \times 10^{-1011}\right)_2$$

指数オフセットバイナリーの計算

$$\left(-11+1023\right)_{10} = (1012)_{10} = (1111110100)_2$$

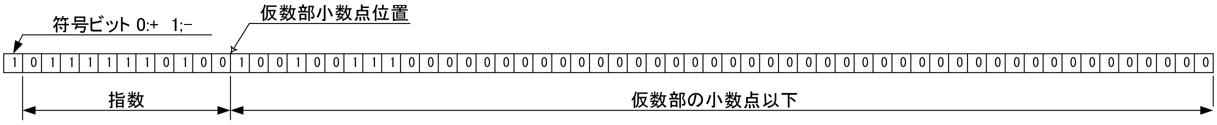

図 13: IEEE 規格の浮動小数点表示の例