コンピュータにおける数の表現

コンピュータの内部ではすべてのプログラムやデータが 0 と 1 だけで表現されている．ここでは，数がどのような形式でコンピュータ内に収められているかを知ろう．

1.　2 進数

0 と 1 だけで表現されているということは 2 進数 (binary numbers) と関連が深いが，そもそも 2 進数がどのようなものかをここで復習しよう．

10 進数では 0 から 9 までの 10 個の数字を用いて値を表し，ある桁が 10 を超えると左隣の桁に繰り上がるように，2 進数では 0 と 1 だけしかないので，ある桁が 2 になると左隣の桁に繰り上がる．したがって，整数を 0 から大きい方へ順番に並べていくと，0，1，10，11，100，101，110，111，1000，…という具合になる．n 桁の 2 進数 $b_{n-1}b_{n-2}...b_1b_0$ は $b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$ という計算をすれば 10 進数に変換できる．10 進数から 2 進数へ変換するには，図 1 のような計算をすればよい．

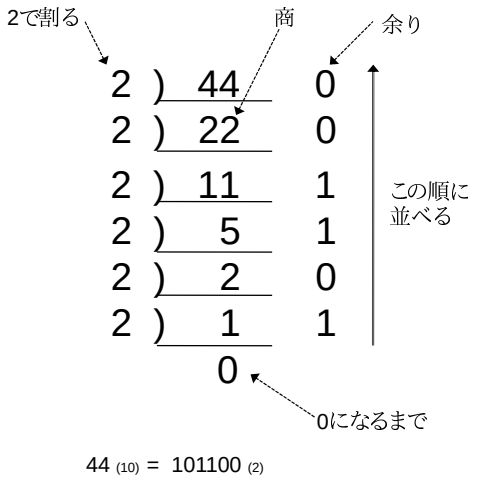

図 1　10 進数整数の 2 進数への変換

一方，2 進数の負の数については，0 から小さい方へ並べていくと，0，−1，−10，−11，−100，−101，…となるので，単にマイナスの符号をつければよいことになる．

以上の 2 進数の話は，あくまで数学（算数）の世界での表現である．

2.　16 進数

2 進数で数を表現するとどうしても桁が多く必要となり，人間にとっては分かりにくい．そこで，2 進数を 8 進数や 16 進数で表現することがしばしば行われる．表 1 の左側の 2 列に示す通り，16 進数を使うと 4 桁の 2 進数が 1 つの 16 進数として表現でき，表記上の幅が圧縮できる．

3.　2 の補数表示

これまで述べた 2 進数をコンピュータ内ではどう表しているだろうか? ここで一つ重要なことは，コンピュータ内では有限の桁，すなわち有限のビット列を使って数を表現しているのである．ここでは，整数につい

て 4 桁で考えてみよう．零および正の整数については 1 節と同様の表現となるが，一番左のビット (most significant bit; MSB) を符号のために使い，MSB が 0 の時は正と零，MSB が 1 の時は負数とする．すなわち，$0_{(10)}$ は 0000 となり，1 は 0001，2 は 0010，…，7 は 0111 となる．一方，−1 は 1111，−2 は 1110，…，−8 は 1000 である．このような負数の表現は **2 の補数表示** (two's complement representation) と呼ばれ，多くのコンピュータで採用されている．表 1 の 3 列目に 2 の補数表示による整数の例を示す.

ある負の整数の 2 の補数表示を求めるには簡易な方法がある．たとえば −3 の 2 の補数表示を求めるには，正数 3 の 2 進数表示 0011 に対して，各桁の 0 と 1 を反転して 1100 にした後，1 を足して，1101 とするのである．負の数に 2 の補数表示を用いることで，
$2 + (-3) \ = \ 0010_{(2)} + 1101_{(2)} \ = \ 1111_{(2)} \ = \ -1$
のように加減算が負の数に対しても足し算で計算できる．ただ，4 ビットの場合は −8〜7 の整数しか表現できないので，4+6 の結果が −6 になってしまうなど，計算結果が表現幅に収まらないときは正しい結果が得られない.

整数値として 32 ビットを用いる場合，-2^{31} (= −2147483648) 〜 $2^{31} - 1$ (= 2147483647) の整数値が表わされることになる.

表 1　4 ビットのパターンに対するさまざまな解釈

ビットパターン (2 進数)	16 進数	2 の補数表示	符号なし整数 (10 進数)
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	−8	8
1001	9	−7	9
1010	a	−6	10
1011	b	−5	11
1100	c	−4	12
1101	d	−3	13
1110	e	−2	14
1111	f	−1	15

4.　符号なし整数

ときには負の整数を表す必要がないことがある．すると MSB で符号を示す必要がなくなるので，最左のビットも他のビットと区別なく使って，4 ビットの場合は 0〜15 の整数を表すことができる．これが**符号なし整数** (unsigned integer) と呼ばれる表現形式である．表 1 の 4 列目に符号なし整数の例を示す.

整数値として 32 ビットを用いる場合，符号なし整数として 0 〜 $2^{32} - 1$ (= 4294967295) の整数値が表現できることになる.

5. 実数

実数については話がやや複雑になる．まず，たとえば，10 進数で 23.4 という数は 234×10^{-1} とも表現できるし，2.34×10^{1} とも表現できる．2 進数でも事情は同様で，$10.01_{(2)}$ は　1001×2^{-2} とも 1.001×2^{1} とも表現できる．そこで，実数は常に $1.abc\cdots \times 2^{p}$ と表現することにする．これを正規化 (normalization) と呼び，小数点が移動することから**浮動小数点表示** (floating point representation) という．そして正規化された数の $abc\cdots$ の部分 (仮数部; significand) と p の部分 (指数部; exponent) をコンピュータ内に収めることにする．具体的には，整数の時と同様に最左ビットを仮数の符号に割り当て，その右に指数，そして残りを仮数に割り当てる．指数については 0 を一番小さな指数値 (たいていは負の数)，1111…1 を一番大きな指数値に割り当てるような**げたばき表示** (biased notation) にする．通常は 0 乗をその真ん中あたりにし，指数部は「31 のげた (bias) とする (指数 31 を 0 乗として使うの意味)」などという．結果として，

$$(-1)^{\mathrm{MSB}} \times (1 + \text{仮数}) \times 2^{(\text{指数}-\text{げた})}$$

という 2 進数を表すことになる．なお，全部のビットを 0 にしたものは特別扱いで，零 (0.0) とする．

このような表現法は IEEE754 (IEEE はアイ・トリプル・イーと読む) という規格で多くのコンピュータで現在採用されており，単精度浮動小数点数では指数部 8 ビット (127 のげた)，仮数部 23 ビットとなっている．結果としておよそ $\pm 3.4 \times 10^{-38}$ 〜$\pm 3.4 \times 10^{38}$ の範囲の数値を表せる．一方，倍精度では指数部 11 ビット (1023 のげた)，仮数部 52 ビットとなっている．この場合はおよそ $\pm 1.7 \times 10^{-308}$〜$\pm 1.7 \times 10^{308}$ の範囲の数値を表せる．図 2.3 に実数 -3.625 を単精度で表現した例を示す．

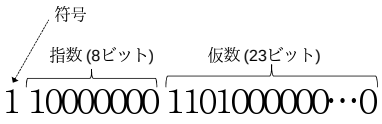

図 2.3　$-3.625_{(10)}$ の単精度浮動小数点での表現

ここで注意してほしいのは，10 進数の 0.1 のように 2 進数にすると無限小数になってしまうものについては，仮数部に入らない部分は捨てられてしまい，誤差の原因となることである．これを**丸め誤差**という．