2. Unconventional Fixed-Radix Number Systems

負数の 2 の補数表現を用いた従来の 2 進数系は一般の計算機で使われるが、ほかに、一定の応用のため有用な数系がある。

- 負の基数数系 (negative-radix number systems)
- 符号桁付き数系 (signed-digit number systems)

2.1 負の基数数系

従来の数系は固定基数系で、 i 番目の桁 x_i の重み w_i は r^i で、すべての桁の表現範囲が $\{0, 1, \cdots, r-1\}$ となる数系である。

$$X = \sum_{i=0}^{n-1} x_i \, w_i = \sum_{i=0}^{n-1} x_i \, r^i \qquad (1)$$

基数 r については普通、正数を選ぶ。しかし、正数に r を制限する必要はなく、 β を正数として $r = -\beta$ と選んでも良い。

桁集合： $x_i \in \{0, 1, \cdots, \beta - 1\}$

$(x_{n-1}, x_{n-2}, \cdots, x_0)$ の値は以下の通りである。

$$X = \sum_{i=0}^{n-1} x_i (-\beta)^i \qquad (2)$$

$$w_i = \begin{cases} \beta^i & (\text{ i が偶数のとき }) \\ -\beta^i & (\text{ i が奇数のとき }) \end{cases}$$

[例 2.1]

$\beta = 10$ となる負の基数数系を負の 10 進数系と呼ぶ。次の 3 桁の負の 10 進数系を考える。

$$(192)_{-10} = 100 - 90 + 2 = 12,$$

$$(012)_{-10} = -10 + 2 = -8$$

$(x_2, x_1, x_0)_{-10}$ で表せる最も大きな正の値は、$(909)_{-10} = 909_{10}$ 、最も小さな値は $(090)_{-10} = -90_{10}$ である。

この負の 10 進数系で表せる値の範囲は $-90 \leq X \leq 909$ である。

この範囲は非対称的で、正数は負数のほぼ 10 倍ある。これは n が奇数の時いつも成り立つ。もし n が偶数の時は、逆のことが成り立つ。例えば、 $n = 4$ のとき $-9090 \leq X \leq 909$ となる。

負の基数系の正負：

負の基数系では符号桁をとる必要がなく、負数を表すとき基数補数法のような特別な方法を用いる必要はない。数の符号は最初の 0 でない桁によって決められる。

正負の区別をしないということは、算術演算における数の符号を無関係にする。しかし、負の基数数系における基本的な算術演算のアルゴリズムは、一般的な数系のアルゴリズムよりわずかに複雑である。

[例 2.2]

$\beta = 2$ で長さ $n = 4$ の負の基数数系を考える。

この数系は負の 2 進数系と呼ばれている。4 ビットの負の 2 進数系の範囲は以下の通りである。

$$(-10)_{10} = (1010)_{-2} \leq X \leq (0101)_{-2} = (+5)_{10}$$

負の基数の数を加算するとき、正負両方の桁上げが行われる。

$$
\begin{array}{cccccc}
 & -8 & +4 & -2 & +1 & \\
 & 0 & 1 & 0 & 1 & 5 \\
+ & 1 & 1 & 0 & 1 & -3 \\
\hline
 & 0 & 1 & 1 & 0 & 2
\end{array}
$$

重み -2 の列には重みが $+2$ の繰り上がりがある。

それを扱うには $+2$ を $+4-2$ に変換する方法しかない。すなわち、重みが -2 の

総ビットと +4 の列への正の繰り上がりを生む。

　　+4 の列でも、重み +8 の繰り上げが起こっている。

　　この桁上げは −8 の縦の列の演算ビットとお互い打ち消され、和のビットが 0 となる。

2.2 固定基数数系の一般クラス

負の基数数系や多くの他の固定基数系数は、無冗長な数系の大きなクラスのメンバーである。このクラスでは、どんな n 桁の数系でも正の基数 β と長さ n のベクトル $\Lambda = (\lambda_{n-1}, \lambda_{n-2}, \cdots, \lambda_0)$(ただし $\lambda_i \in \{-1, 1\}$) で表現できる。

固定基数数系の一般表現：

基本的な桁の値 $\{0, 1, \cdots, \beta - 1\}$ を持つようなシステムは 3 つの組 $\langle n, \beta, \Lambda \rangle$ で表せる。

この数系 $\langle n, \beta, \Lambda \rangle$ での n タップルの $(x_{n-1}, x_{n-2}, \cdots, x_0)$ の値は以下のように与えられる。

$$X = \sum_{i=0}^{n-1} \lambda_i x_i \beta^i \tag{3}$$

λ_i との積を用いて、各桁に $-\beta^i$ と β^i と 2 種類の重みを付けることができる。

基数 β に対して、このクラスに Λ の値に対応している 2^n の通りの数表現がある。

正の基数数系： $\lambda_i = 1$;

負の基数数系： $\lambda_i = (-1)^i$;

基数補数数系： $\Lambda = (-1, 1, 1, \cdots, 1)$。

数系クラスの性質：

一般数系 $\langle n, \beta, \Lambda \rangle$ において、表現できる最大と最小の整数をそれぞれ P と N とする。

$P = (p_{n-1}, p_{n-2}, \cdots, p_0)$ は次の式を満たす。

$$p_i = \begin{cases} \beta - 1 & (\ \lambda_i = +1\ ) \\ 0 & (\ \lambda_i = -1\ ) \end{cases}$$

ただし $i = 0, 1, 2, \cdots, n-1$。

p_i の一般的な表現は、 $p_i = \frac{1}{2}(\lambda_i + 1)(\beta - 1)$ となる。

$$
\begin{aligned}
P &= \sum_{i=1}^{n-1} \frac{1}{2}(\lambda_i + 1)(\beta - 1)\beta^i && (4) \\
&= \frac{1}{2}\left[\sum_{i=0}^{n-1} \lambda_i(\beta - 1)\beta^i + \sum_{i=0}^{n-1}(\beta - 1)\beta^i\right] \\
&= \frac{1}{2}\left[Q + (\beta^n - 1)\right] && (5)
\end{aligned}
$$

Q は $\langle n, \beta, \Lambda \rangle$ における $(\beta - 1, \cdots, \beta - 1)$ の値である。

Q はとても重要な量である。

同様に表現できる最も小さい数を表す

$N = (y_{n-1}, y_{n-2}, \cdots, y_0)$ の各桁は以下のようになる。

$$y_i = \begin{cases} \beta - 1 & (\ \lambda_i = -1\ ) \\ 0 & (\ \lambda_i = +1\ ) \end{cases}$$

ただし $i = 0, 1, \cdots, n-1$。

これより N の値は以下のようになる。

$$N = \sum_{i=0}^{n-1} \frac{1}{2}(\lambda_i - 1)(\beta - 1)\beta^i = \frac{1}{2}\left[Q - (\beta^n - 1)\right] \tag{6}$$

範囲 $N \leq X \leq P$ にある整数の数は
$P - N + 1 = \beta^n$ で、一般的にその範囲は非対称である。

非対称の大きさは、最大数と最小数の絶対値の差で求まる。

$$P - |N| = P + N = Q \tag{7}$$

[例 2.3]

$\Lambda = (\cdots, -1, +1, -1, +1)$ である負の基数数系は非対称な範囲を持つ。

n が偶数のときは、正数の β 倍の負数が存在する。より多くの正数を持つなら、代わりに $\langle n, \beta, \Lambda = (\cdots, +1, -1, +1, -1)\rangle$ を使えばよい。

次の 2 つの 2 進数系はかなり対称的である。

1. **２の補数数系：**

$\langle n, \beta = 2, \Lambda = (-1, 1, 1, \cdots, 1) \rangle$

$P + N = Q = -ulp$

2. **次のような数系：**

$\langle n, \beta = 2, \Lambda = (+1, -1, -1, \cdots, -1) \rangle$

$P + N = Q = +ulp$

補数の概念：

数系 $\langle n, \beta, \Lambda \rangle$ における数 X の補数 $\overline{X}$ は

$\overline{x_i} = (\beta - 1) - x_i$ により定義される。

$$\begin{aligned} \overline{X} &= \sum_{i=0}^{n-1} \overline{x_i} \lambda_i \beta^i \\ &= \sum_{i=0}^{n-1} \lambda_i (\beta - 1) \beta^i - \sum_{i=0}^{n-1} \lambda_i x_i \beta^i \\ &= Q - X \end{aligned} \tag{8}$$

したがって

$$-X = \overline{X} - Q = \overline{X} + (-Q) \qquad (9)$$

X の加法逆数は Q の加法逆数と補数 $\overline{X}$ との和によって作られる。

[例 2.4]

2 の補数数系において、$Q = -ulp$ であるから、$-X = \overline{X} + ulp$ と書ける。

負の 2 進数系においては、$Q = (\cdots, 1, 1, 1, 1) = -(\cdots, 0, 1, 0, 1)$ なので、$-Q = (\cdots, 0, 1, 0, 1)$ である。

例として $(01011)_{-2} = (-9)_{10}$ の加法逆数を求めると以下のようになる。

$$
\begin{array}{lccccccl}
 & 16 & -8 & +4 & -2 & +1 & & \\
\overline{X} & 1 & 0 & 1 & 0 & 0 & & \\
-Q \quad + & 1 & 0 & 1 & 0 & 1 & & \\
\hline
 & 1 & 1 & 0 & 0 & 1 & & = 9_{10}
\end{array}
$$

この加算は負の 2 進系の加算の規則を用いて行なわれた。

結果はもとの数に、その加法逆数を加えることによって確かめられる。また、負の 2 進数の規則に従うと

$$(01011)_{-2} + (11001)_{-2} = (00000)_{-2}$$

となる。

減算：

加法逆数として次の等式を利用することによって行われる。

$$X - Y = X + \overline{Y} + (-Q) \tag{10}$$

この等式によれば、桁上げを含む 2 回の加算を実行する。以下の表現を利用すると

$$X - Y = \overline{\overline{X} + Y} \tag{11}$$

２回の補数演算を用いることにより、１回の加算だけですむ。

2.3 SIGNED-DIGIT NUMBER SYSTEM

今までに検討したすべての固定基数系において、用いる数集合は $\{\mathbf{0}, ...\mathbf{r-1}\}$ に制限されている。

代わりに次の数集合を与える。

$$\mathbf{x_i} \in \{\overline{(\mathbf{r-1})}, \overline{(\mathbf{r-2})}, \cdots, \overline{\mathbf{1}}, \mathbf{0}, \mathbf{1}, \cdots, (\mathbf{r-1})\}$$

ここで、$\bar{i}$ は $-i$ と等しく、前述のように $(r-1)-i$ ではない。混同するかもしれないが、この表記法が一般に専門文献などの中で使われているので、この表記法を使う。

SD **数系の特徴：**

各々の桁が正または負数であるので、特別の符号桁は必要ない。

この数系は、符号付き桁数系 $(SD\ system)$ と呼ばれる。

[例 2.5]

$r = 10$ の時、使用できる数は

$\{\overline{9}, \overline{8}, \cdots, \overline{1}, 0, 1, \cdots, 9\}$

である。もし $n = 2$ ならば、その範囲は、$\overline{9}\ \overline{9} \leq$x$\leq 9\ 9$ になる。この範囲に 199 の数を含んでいる。

しかし、(x_1, x_0) の 2 桁で、 それぞれの数が 19 通りずつあるので、 $19^2 = 361$ 通りの表現がある。

幾つかの数は、2 通り以上の表現を

持っているため、この数系は、冗長である。例えば、

$$(0\ 1) = (1\ \overline{9}) = 1, (0\ \overline{2}) = (\overline{1}\ 8) = -2$$

0 の表現は特別であり、また 10 の表現もそうである。 361 の表現のうち、 $361\ -\ 199 = 162$ の数表現は、冗長なものである。 81% の冗長率をもつ。

冗長量の減少：

数系において幾つかの冗長性を加えることが、とても有益になりうる。

一方、より大きな数集合では各々の数を表現するために、より多くのビットを必要とするので、冗長性が多すぎるのは、高コストになりすぎるにちがいない。ここで、次のように数集合を制限することによって、冗長の量をへらす。

$$x_i \in \{\overline{a}, \overline{a-1}, \cdots, \overline{1}, 0, 1, \cdots, a\}$$
$$with \quad \lceil \tfrac{r-1}{2} \rceil \leq a \leq r-1$$

数 x の上限 $\lceil x \rceil$ は、 x より大きいか、 x に等しい最も小さい整数である。

基数 r の数系を表現するために、少なくとも r 個の異なる数が必要になる。

また、 $\overline{a} \leq x_i \leq a$ で、 $2a+1$ の数をもつから、

$2a+1 \geq r$ を満たさなければならない。

[例 2.6]

$r = 10$ の時、 a の範囲が $5 \leq a \leq 9$ とする。

$a = 6$、 $n = 2$ の時、 $\overline{66} \leq x \leq 66$ の中に 133 の数がある。

各桁は、 13 通りの値を持つので、総計 $13^2 = 169$ の表現がある。

これは、 27% の冗長率がある。

ここで、 1 は (01) という 1 つの表現しか持たない。

$(1\overline{9})$ は、正当な表現ではない。なぜなら 、$\overline{9}$ は不当な表現である。

4 はそれでも (04) と $(1\overline{6})$ の 2 つの表現を持つ。

SD **数表現による並列加算：**

SD 数表現は、乗除算のアルゴリズムを開発する際に有用である。

SD 数の導入の最初のきっかけは、加減算における桁上げ伝搬連鎖を取り除くことであった。

$$(x_{n-1}, \cdots, x_0) \pm (y_{n-1}, \cdots, y_0) = (s_{n-1}, \cdots, s_0)$$

４つの演算数 $x_i, y_i, x_{i-1}, y_{i-1}$ にだけ依存する総和数 s_i を用いることによって、桁上げ連鎖を無くしたい。

加算アルゴリズム：

ステップ 1. 中間和 u_i と桁上げ数 c_i を計算する

$$u_i = x_i + y_i - rc_i$$

ただし

$$c_i = \begin{cases} 1 & if \quad (x_i + y_i) \geq a \\ \overline{1} & if \quad (x_i + y_i) \leq \overline{a} \\ 0 & if \quad |x_i + y_i| < a \end{cases}$$

ステップ 2. 最終和 $s_i = u_i + c_{i-1}$ を計算する。

[例 2.7]

$r = 10$、 $a = 6$ とするとき、

$x_i \in \{\overline{6}, \cdots, 0, 1, \cdots, 6\}$ となる。

ステップ 1 は次のようになる。

$u_i = (x_i + y_i) - 10c_i$、

$$c_i = \begin{cases} 1 & if \quad (x_i + y_i) \geq 6 \\ \overline{1} & if \quad (x_i + y_i) \leq \overline{6} \\ 0 & \text{その他} \end{cases}$$

従来の十進数系において、 4536 と

1466 を加算

$$\begin{array}{rcccc} & 4 & 5 & 3 & 6 \\ + & 1 & 4 & 6 & 6 \\ \hline & 6 & 0 & 0 & 2 \end{array}$$

この場合、桁上げが最小桁から最大桁まで及んでしまう。しかし、その代りに次の計算においては桁上げの連鎖は生じない。

$$
\begin{array}{rccccc}
 & 4 & 5 & 3 & 6 & \\
+ & 1 & 4 & 6 & 6 & \\
\hline
0 & 1 & 1 & 1 & & c_i \\
+ & 5 & \overline{1} & \overline{1} & 2 & u_i \\
\hline
 & 6 & 0 & 0 & 2 & s_i
\end{array}
$$

ここで、桁上げのビットは、アルゴリズムの第二ステップの操作を簡単にするため左にシフトされている。

上の例題において、演算数は従来の十進数か数集合 $\{\overline{6}, \cdots, 0, 1, \cdots, 6\}$ をもつ SD 十進数のどちらかとみなせるように選ばれる。

通常、従来の十進数は用いられる SD 数集合に定義されていない $7, 8$ や 9 のような数字を使う。

上の加算アルゴリズムは、従来の十進数を和 $(x_i + y_i)$ として、 SD 形式に変換するために使われる。

[例 2.8]

10 進数 27956 の SD 数表現を作るために、先のアルゴリズムを適用すると、結果は次のようになる。

$x_i + y_i$		2	7	9	5	6
c_i	0	1	1	0	1	
u_i		2	$\overline{3}$	$\overline{1}$	5	$\overline{4}$
s_i		3	$\overline{2}$	$\overline{1}$	6	$\overline{4}$

SD 数表現を従来の十進表現に変換するために、負の重みを持つ数を正の重みを持つ数から、引くことができる。

$3\overline{21}6\overline{4}$ より、次を得る。

$$\begin{array}{rccccc} & 3 & 0 & 0 & 6 & 0 \\ - & 0 & 2 & 1 & 0 & 4 \\ \hline & 2 & 7 & 9 & 5 & 6 \end{array}$$

新しい桁上げが生じない条件：

新しい桁上げが発生しないことを保証するために、$u_i + c_{i-1}$ から計算された s_i の和が、$|s_i| \leq a$ を満たさなければならない。

また、$|c_i| \leq 1$ から、条件 $|u_i| \leq a - 1$ は、すべての考えうる x_i の値において満たされなければならない。

例えば、 $x_i + y_i$ により取れる最大値は $2a$ で、
$c_i = 1, u_i = 2a - r$。

$a \leq r - 1$ より、 $u_i = 2a - r \leq a - 1$ は明らかに満たされている。

$c_i = 1$ となる最小値である $x_i + y_i = a$ ならば、
$u_i = a - r < 0$ である。

$|u_i| = r - a$ を $|u_i| \leq (a - 1)$ に代入して、不等式 $2a \geq r + 1$ が得られる。

数集合の制約：

それゆえ、選ばれた数集合は、次を満たさなければならない。

$$\lceil \frac{r+1}{2} \rceil \leq a \leq r-1$$

$a \geq \lceil \frac{r+1}{2} \rceil$ ならば、条件 $|u_i| \leq a-1$ が満たされる。

[例 2.9]

$SD10$ 進数は、先のアルゴリズムにおいて、新しい桁上げが生成されないことを保証するために、$a \geq 6$ を満たさなければならない。

2.4 2進 SD 数系

$r = 2$ のとき、許される数集合はただ一つで、

$$\{\overline{1}, 0, 1\}$$

である。 a は 1 に等しくなければならない。

以前の加算アルゴリズムにおいて中間和と桁上げは

$$u_i = (x_i + y_i) - 2c_i$$

と

$$c_i = \begin{cases} 1 & if \quad (x_i + y_i) \geq 1 \\ \overline{1} & if \quad (x_i + y_i) \leq \overline{1} \\ 0 & if \quad (x_i + y_i) = 0 \end{cases}$$

これらの規則は、表 2.1 に纏められている。加算 $x_i + y_i$ は可換の演算であるので、表では $x_i y_i = 10, x_i y_i = \overline{1}0$ の組合わせは含まない。

表 2.1 2進 SD 数の加算規則

$x_i y_i$	00	01	$0\overline{1}$	11	$\overline{1}\ \overline{1}$
c_i	0	1	$\overline{1}$	1	$\overline{1}$
u_i	0	$\overline{1}$	1	0	0

上記の加算規則の問題点：

２進の場合は、条件 $a \geq \lceil \frac{r+1}{2} \rceil = 2$ は満たされない。

その結果、アルゴリズムのステップ２において、新しい桁上げが生成されないことを保証できない。

もし演算数が数 $\overline{1}$ を含まなければ、新しい桁上げは発生しない。

例えば、次の加算例の場合、従来の 2 進数系において最下位桁から最上位桁まで及ぶ桁上げが発生されるが、

$$
\begin{array}{cccccccc}
 & 1 & 1 & \cdots & 1 & 1 & & \\
+ & 0 & 0 & \cdots & 0 & 1 & & \\
\hline
1 & 1 & 1 & \cdots & 1 & & & c_i \\
\overline{1} & \overline{1} & \cdots & \overline{1} & 0 & & u_i \\
\hline
1 & 0 & 0 & \cdots & 0 & 0 & & s_i
\end{array}
$$

ここでは、桁上げ伝搬の連鎖は存在しない。

しかし、$\overline{1}$ を含む SD 数が加算されたら、新しい桁上げが生成されるかも知れない。

例えば、$x_{i-1}y_{i-1} = 01$ ならば、$c_{i-1} = 1$ になる。そして、$x_iy_i = 0\overline{1}$ ならば $u_i = 1$ になる。

これから、$s_i = u_i + c_{i-1} = 1 + 1$ となり、新しい桁上げが生成される。

$$
\begin{array}{cccccccc}
 & & 0 & \overline{1} & 1 & \overline{1} & 1 & 1 & \\
+ & & 1 & 0 & 0 & \overline{1} & 0 & 1 & \\
\hline
 & 1 & \overline{1} & 1 & \overline{1} & 1 & 1 & & c_i \\
 & & \overline{1} & 1 & \overline{1} & 0 & \overline{1} & 0 & u_i \\
\hline
 & & * & * & * & 1 & 0 & 0 & s_i
\end{array}
$$

星印が、新しい桁上げが生成され、伝搬を許さなければならない場所を示す。

加算規則問題点の解析：

$x_i y_i = 0\overline{1}$ の同時、 $x_{i-1} y_{i-1} = 11$ または $x_{i-1} y_{i-1} = 01$ のとき、 $c_{i-1} = u_i = 1$ が生じる。

これらの場合に、 $c_i = 0$ を選ぶことによって $u_i = 1$ を避けられる。このとき $u_i = \overline{1}$ になる。

しかし、 $x_{i-1} y_{i-1} = \overline{1}\,\overline{1}$ のとき、表 2.1 における $x_i y_i = 0\overline{1}$ に対して $c_i = 0$ と $u_i = \overline{1}$ に変えるべきではない。

このとき、 $c_{i-1} = \overline{1}$ になり、 $c_i = \overline{1}$ と $u_i = 1$ にしなければならない。

同様に、 $c_{i-1} = u_i = \overline{1}$ は、 $x_i y_i = 01$ または $x_{i-1}y_{i-1} = \overline{1}\,\overline{1}$ と $x_{i-1}y_{i-1} = 0\overline{1}$ のときに生じる。

この場合、 $c_i = 0$ と $u_i = 1$ を選ぶことによって、 $u_i = \overline{1}$ になることを避けられる。

u_i と c_i を決めるとき、右に 2 つのビット $x_{i-1}y_{i-1}$ を調べ、表 2.2 に示されている規則を満たすことによって、新しい桁上げが発生しないことを保証できる。並列にすべてのビットで c_i と u_i を計算できる。

表 2.2 2進 SD 数加算の改良規則

$x_i y_i$	00	01	01	$0\overline{1}$	$0\overline{1}$	11	$\overline{1}\,\overline{1}$
$x_{i-1} y_{i-1}$	–	neither is $\overline{1}$	at least one is $\overline{1}$	neither is $\overline{1}$	at least one is $\overline{1}$	– –	– –
c_i	0	1	0	0	$\overline{1}$	1	$\overline{1}$
u_i	0	$\overline{1}$	1	$\overline{1}$	1	0	0

[例 2.10]

先の例を繰り返して、次を得る

$$
\begin{array}{cccccccccc}
 & & & 0 & \overline{1} & 1 & \overline{1} & 1 & 1 & \\
+ & & & 1 & 0 & 0 & \overline{1} & 0 & 1 & \\
\hline
 & & 0 & 0 & 0 & \overline{1} & 1 & 1 & & c_i \\
 & & & 1 & \overline{1} & 1 & 0 & \overline{1} & 0 & u_i \\
\hline
 & & & 1 & \overline{1} & 0 & 1 & 0 & 0 & s_i
\end{array}
$$

２つの演算数の直接和が、$1\overline{1}1\overline{1}00$ になり、010100 と同値で、20_{10} を表している。

２進数を加算するときの桁上げ伝搬を取り除くことが、乗除算のような演算の速度を上げることを可能にする。

加減算の実行には、たくさんの加減算を含んでいるため２進 SD 数は、とても有益であり、特に、乗除算の高速アルゴリズムの開発である。

この場合、０でない桁数が最小の SD 数表現（最小 SD 数表現）に興味を持つ。

０でない数が加減算に対応するが、０の数がシフトのみの演算に対応する。

[例 2.11]

$x = 7$ で、次の表現を得る。

	8	4	2	1
	0	1	1	1
	1	$\overline{1}$	1	1
	1	0	$\overline{1}$	1
	1	0	0	$\overline{1}$
1	$\overline{1}$	$\overline{1}$	1	1

...

これらの表現において、$100\overline{1}$ が最小の表現である。