1. Conventional Number Systems （慣用な数系）

- The Binary Number System （２進数系）
- Machine Representations of Numbers （数の機械表現）
- Radix Conversion （基数変換）
- Representations of Negative Numbers （負数の表現）
- Addition and Subtraction （加算と減算）

1.1 ２進数系

一般の計算機では整数が固定の語長をもつ２進数で表現される。すなわち、以下のような２進数の数列である。

$$(x_{n-1}, x_{n-2}, \cdots, x_1, x_0)$$

上記の数列（n-tuple) は整数の値を表している。

$$X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2 + x_0 = \sum_{i=0}^{n-1} x_i 2^i \qquad (1$$

数 x_i の重みは２の i 乗であり、２を (radix) 基数と呼ぶ。

算術ユニット上のオペランド（演算数）と結果は固定された長さのレジスタに保存されるので、算術ユニットの範囲内で表現できるのは有限な数である。

すなわち、$[X_{min}, X_{max}]$ は表現可能な数の範囲である。

[例 1.1]

符号なしの 2 進数 4 ビットで表現する整数の例。

$X_{max} = (15)_{10} = (1111)_2$ and
$X_{min} = (0)_{10} = (0000)_2$

$$(16)_{10} = (10000)_2 - - > (0000)_2 = (0)_{10}$$

$$[X_{min}, X_{max}] = [0, 15]$$

and

$$(X\ modulus\ 16)\ or\ (X\ mod\ 16)$$

また、加算の場合、

$X = (11)_{10} = (1011)_2, Y = (7)_{10} = (0111)_2$ のとき、
$X + Y = (18)_{10} = (10010)_2$ である。

4 ビットのレジスタに収めなければならないとすると 5 ビット目を切捨てて、結果は $2^4 = 16$ となってしまう。すなわち、

$$(0010)_2 = 2 = 18 \; mod \; 16$$

1.2 数の機械表現

２進法は算術ユニットにおいて数値を表現する数系の特例である。

一般の数系の特徴： 非冗長、重み付き、位置上の数系。

非冗長：全ての数は唯一の表現しか持たない。

重み付き：

$$X = \sum_{i=0}^{n-1} x_i w_i \qquad (1.2)$$

によって与えられた $(x_{n-1}, x_{n-2}, \cdots x_0)$ の値を決める重みの数列

$$w_{n-1}, w_{n-2}, \cdots, w_1, w_0$$

があるということである。従って、 w_iはx_i 番目の場所で数を割り当てられた重みである。

位置上の数系：重み w_iはx_i の i 番目にしか依存していない。

一般の数系では重み w_i は固定した整数 r の i 乗でありそれは基数である。

$$w_i = r^i$$

このような数系は固定基数数系とも呼ばれる。

x_iに重みr^i を割り当てる時、 $0 \leq x_i \leq r-1$ を満たさなければならない。

$x_i \geq r$ である時、$(\cdots, x_{i+1}, x_i, \cdots)$ と $(\cdots, x_{i+1}+1, x_i - r, \cdots)$ のような同じ数値に対して、二つの機械表現において、

$$x_i r^i = (x_i - r) r^i + 1 \cdot r^{i+1}$$

といった結果を許してしまう。すなわち、$x_i \leq r$ にすれば、固定基数体系中に冗長性を得る。

実数の表現：

実数の場合、数列を小数部と整数部との組み合わせにする。

n 桁を 2 つの組みに分割する。 k 数を整数部、 m 数を小数部とすると、 $k + m = n$。

$$\left(\underbrace{x_{k-1}x_{k-2}\cdots x_1x_0}_{\text{整数部}} \cdot \underbrace{x_{-1}x_{-2}\cdots x_{-m}}_{\text{小数部}}\right)_r$$

$$
\begin{aligned}
X &= x_{k-1}r^{k-1} + x_{k-2}r^{k-2} + \cdots \\
&\quad + x_1 r + x_0 + x_{-1}r^{-1} + \cdots + x_{-m}r^{-m} \\
&= \sum_{i=-m}^{k-1} x_i r^i
\end{aligned}
$$

小数点はレジスタに保存されないが、 kとm の間の固定された場所にある。

普通、小数点の位置は数の最右端
($m = 0$である整数) か数の最左端
($k = 0$である小数) とされる。

ulp:

長さ n の演算数が与えられた時、最下位ビットの重み r^{-m} は小数点の位置を指し示す。

最下位における数 (ulp) という概念を導入する。これは、最下位ビットの重みである。

$$ulp = r^{-m} \qquad (1.4)$$

1.3 基数変換

基数変換：ある数系の数 X の表現を他の数系の表現に変換すること。

変換の主な理由は、より少ない桁数で表現できる１０進法に我々が慣れているのに対して、多くの算術ユニット演算は２進法を用いていることがあげられる。

整数の場合：

教科書の式 (1.3) を次のように書き直せる。

$$X_I = \{[\cdots(x_{k-1}r_D+x_{k-2})r_D+\cdots+x_2]r_D+x_1\}r_D + x_0 \qquad (1.5)$$

ここで、 $0 \leq x_i < r_D$。

X_I が r_D で割ると、余りとして x_0 が得られる。
同時に商は

$$\{[\cdots(x_{k-1}r_D + x_{k-2})r_D + \cdots + x_2]r_D + x_1\}$$

となる。

さらに、r_D で割ると、x_1 を余りとして得られる。

繰り返し商が０になるまで r_D で割ることにより、すべてのあまりとして x_i が求まる。

小数の場合：

小数部分 X_F の $(x_{-1}x_{-2}\cdots x_{-m})_{r_D}$ を見つけるために、式 (1.3) の小数部分を次のように書き直す。

$$X_F = r_D^{-1}\{x_{-1} + r_D^{-1}[x_{-2} + r_D^{-1}(x_{-3} + \cdots)]\}$$

$X_F \times r_D$ により、整数部分として x_{-1} と次のような小数部分が得られる。

$$r_D^{-1}[x_{-2} + r_D^{-1}(x_{-3}\cdots)]$$

手順：

繰り返し小数部分に r_D を掛けることで、必要とする数の整数を生成していく。

ある数系において有限小数は他の数系において無限小数となる可能性がある。

[例 1.2]

$X = 46.375_{10}$ を 2 進数に変換する。

まず、$X_I = 46$ として 2 で繰り返し割っていくことにより次の商と余りを得る。

商	余り
23	$0 = x_0$
11	$1 = x_1$
5	$1 = x_2$
2	$1 = x_3$
1	$0 = x_4$
0	$1 = x_5$

小数部分 $X_F = 0.375$ を変換し、繰り返し 2 を掛けていくことにより整数と小数を得る。

整数部	小数部
$0 = x_{-1}$	0.75
$1 = x_{-2}$	0.5
$1 = x_{-3}$	0.0

結果は $46.375_{10} = 101110.011_2$。

もし、与えられた 10 進数の小数部分が $X_F = 0.3$ であるとすると、上記のアルゴリズム では 10 進の小数 0.3_{10} が無限 2 進小数 $(0.0100110011\cdots)_2$ であるので決して止まることがない。

1.4 負数の表現

一般に、二つの異なる形が使われている。

1. 符号と絶対値による表現。符号 - 絶対値表現とも言う。
2. 補数表現（二種類)

(1)Signed-magnitude:

符号と絶対値は分けられて表現される。最初の桁は符号桁で残りの $n-1$ 桁は絶対値を表す。

2 進の場合、符号ビットが 0 ならば正、 1 ならば負である。 r 進の場合、符号桁が 0 と $(r-1)$ の時、それぞれ正と負である。

この場合、 r^n 通りある数列のうち使用するのは $2 \cdot r^{n-1}$ 通り。

数の表現範囲：

$(n-1)$ 桁は整数部と小数部にそれぞれ $(k-1)$ と m 桁とする。

$$X_{max} = (r^{k-1} - ulp) \quad , \quad \text{但し } ulp = r^{-m}$$

対応する表現は $0(r-1)\cdots(r-1)$ となる。

正数の範囲： $[0, r^{k-1} - ulp]$

負数の範囲： $[-(r^{k-1} - ulp), -0]$、
$((r-1)(r-1)\cdots(r-1)$ から $(r-1)0\cdots0$ まで。

[例 1.3]

2 進の場合、全て 2^n の数列を用いたとする。

正数表現： $00\cdots0$ から $01\cdots1$ （2^{n-1} の数列）。

負数表現： $10\cdots0$ から $11\cdots1$ （2^{n-1} の数列）。

$k = n$ （$m = 0, ulp = 2^0 = 1$）のとき、正の数の範囲は $[0, 2^{n-1} - 1]$ で、負数の範囲は $[-(2^{n-1} - 1), -0]$ である。

符号 - 絶対値数表現の欠点：

演算数が符号に依存して演算を実行すること。

例えば、正数 X と負数 $-Y$ を足す時、
$X + (-Y)$ という計算を実行する。

もし、 $Y > X$ ならば、 $-(Y - X)$ という解を得る。従って、 $Y - X$ という計算結果にマイナス符号を付け加えるということである。この決定過程における結果は非常に多くの論理制御と実行時間を要する。

(2) Complement representations:

補数表現は 2 種類ある。

(i) 基数補数 (2 の補数表現)

(ii) 減少基数補数 (1 の補数表現)

両方の補数方法において、正数は符号 - 絶対値方法と同様に表現される。

補数の概念：

負数 $-Y$ は 補数 $(R-Y)$ で表現される。 R は定数。

$$-(-Y) = Y \qquad (1.7)$$

これは、$(R-Y)$ の補数が $R-(R-Y)$ だからである。

補数表現の主な長所は、(R の値を無視して) 加減算の実行前に特別な操作は必要ないことである。

正数と負数の加算：

$$X + (R - Y) = R - (Y - X)$$

$Y > X$ の場合、負の結果の $-(Y - X)$ は、補数の $R - (Y - X)$ で表現される。

$X > Y$ の時、 $X + (R - Y) = R + (X - Y)$ から R を放棄（無視）することにより $(X - Y)$ が得られる。

R の選択：

与えられた数値 Y の補数 $(R - Y)$ の計算が高速で行えるようなもの。

R の値を決定する前に、$\bar{x}_i$ で表現される、一桁の補数を次のように定義する。

$$\bar{x}_i = (r - 1) - x_i \qquad (1.8)$$

$\overline{X} = (\bar{x}_{k-1}, \bar{x}_{k-2}, \cdots, \bar{x}_{-m})$ は X の数列においてすべてのビットを補した後に得られる。

$\overline{X} + X$ は、$x_i + \bar{x}_i = (r-1)$ から得られる。ここで、各桁においての演算は独立。ulp と X と $\overline{X}$ の和は

$$
\begin{array}{rccccc}
X & x_{k-1} & x_{k-2} & \cdots & x_{-m} & \\
+\overline{X} & \bar{x}_{k-1} & \bar{x}_{k-2} & \cdots & \bar{x}_{-m} & \\
\hline
 & (r-1) & (r-1) & \cdots & (r-1) & \\
+\mathrm{ulp} & & & & 1 & \\
\hline
 & 1 & 0 & \cdots & 0 & = r^k
\end{array}
$$

$$X + \overline{X} + ulp = r^k \qquad (1.9)$$

上記の結果が長さ n $(n = k + m)$ のレジスタに収まる時、最上位の桁は切り捨てられ、最終的な答えは 0 となる。

一般に、多くの算術演算の結果を固定長のレジスタ中に収めることは、r^k で割った後の余りを取ることに等しい。

上式の項を並び代えると

$$r^k - X = \overline{X} + ulp$$

その結果として、もし R の値として r^k を取ったとすると、以下が得られる。

$$R - X = r^k - X = \overline{X} + ulp \qquad (1.10)$$

定義された X の補数 $(R - X)$ の計算は極めて単純で k の値に独立であり、その数表現は基数補数表現と呼ぶ。

$R = r^k$ の時：

前の演算の結果、 $X + (R - Y)$ が正 $(X > Y)$ である時、 $R = r^k$ は $R + (X - Y)$ を計算した時に切り捨てられるので、修正する必要はない。

[例 1.4]

$r = 2$、 $k = n = 4$

$(m = 0\ , ulp = 2^0 = 1)$。

基数の補数（二進法の場合、２の補数ともいう）： $2^4 - X = (\bar{X}) + 1$。

この例の場合、 0000 から 0111 の数列でそれぞれ正数の (0_{10}) から (7_{10}) を表す。

一番大きい正数の２の補数は

$1000 + 1 = 1001$ であり、 $(-7)_{10}$ という値を表す。

0 の 2 の補数は
$1111 + 1 = 10000 = 0\ mod\ 2^4$ である。

このように、各正数はそれぞれ１で始まる負数に対応している。 1000 は対応している正数が存在しなくて、負数の $(-8)_{10}$ を表す。

従って、 $k = n = 4$ （整数）のとき、２の補数表現において２進数の範囲は
$-8 \leq 0 \leq 7$ である。

$Y > X$ の時 $X + (-Y)$ という簡単な演算

例として、$2 + (-7)$ を計算してみる。

$$\begin{array}{rrr} & 0010 & 2 \\ + & 1001 & -7 \\ \hline & 1011 & -5 \end{array}$$

２の補数法で正しい結果は表された。しかも、前判断や結果修正など必要は要らない。

$X > Y$ の時、 0111 と 1110 で表し、次の結果が得られる。

$$\begin{array}{rrl} & 0111 & 7 \\ + & 1110 & -2 \\ \hline & 10101 & 5 \end{array}$$

後ろからの 4 桁とる、それが 0101 である。

$R = r^k - ulp$ **の時：**

$$R - X = (r^k - ulp) - X = \bar{X}$$

この補数の導出が基数補数の導出より簡単で、すべての一桁数 $\bar{x}_i$ の計算より $\bar{X}$ 高速な演算が行われる。

しかし、 $R + (X - Y)$ と得られてし かも $(X - Y)$ が正数の時修正が必要である（後で解説する。）

[例 1.5]

$r = 2$、 $k = n = 4$。減少基数補数（２進数の場合１の補数と呼ぶ）

$((2^4) - 1) - X = \bar{X}$。

数列 0000 から 0111 は正数の 0 から 7 まで表す。

一番大きい正数の補数は 1000 であり、$(-7)_{10}$ を表している。

０の１の補数は 1111 である；すなわち、０に対して二つの表現がある。

$k = n = 4$ の時、２進表現による１の補数表現できる範囲は
$(-7) \leq X \leq 7$ である。

$k = n = 4$ の時、三種類の正負数表現は次の表に示される。

表 1.1 ２進数表現（前半）

sequence	２の補数	１の補数	符号ー絶対値
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0

表 1.1 ２進数表現（後半）

sequence	２の補数	１の補数	符号ー絶対値
1111	-1	-0	-7
1110	-2	-1	-6
1101	-3	-2	-5
1100	-4	-3	-4
1011	-5	-4	-3
1010	-6	-5	-2
1001	-7	-6	-1
1000	-8	-7	-0

非二進数表現の場合：

最上位桁において二つの値 0 と $(r-1)$ を有効数字として決めてあるため、利用される数列のパーセント率がかなり減少する。

r^n から取り出した $2 \cdot r^{n-1}$(或は r から取り出したの 2) 使われる。

正数と負数の数が同じあるいはほぼ同じのように、数列の総数 r^n を二分割することによってすべての可能な値で最上位桁を決めることができる。

整数の負数の表現領域：

一般に与えられた数 X を、正数の場合 X、負数の場合 $R - |X|$ による補数表現で表す。

明らかに正数と負数の表現範囲は重複にならない。すまわち

$$|X| \leq R/2$$

必ず成立する。

もし値 $X = R/2 + 1$ が表現領域にあることが許されれば、負数 $-X$ は $R - X = R/2 - 1$ であり、正数 $R/2 - 1$ の表現と同じになってしまう。

基数 r が偶数の場合、正数と負数の重複領域のないことが簡単に実現できる。不等式
$|X| \leq R/2 = r/2 \cdot r^{n-1}$(正数だけの表現で)、有効数字の値 $0, 1, ..., r/2 - 1$ には正数に当たる、そして、値 $r/2, ..., r - 1$ が負数である。

しかし、基数が奇数の場合、 0 から $(r^n/2 - 1)$ が正数で、残りが負数である、正、負数を区別するにはもっと難しくなる。

[例 1.6]

10 進の基数補数法において、有効数字が 10 通りある。

最上位桁が 0,1,2,3,4 となる数列は正数を、最上位桁が 5,6,7,8,9 となる数列は負数をそれぞれ表現する。

$n = 4$ の時、一番大きな正数は 4999 であり、そして数列 5000 から 9999 が負数の -5000 から -1 を表現する。

数表現の範囲は $-5000 \leq X \leq 4999$。

$Y = 2345$； $-Y = -2345$ の表現を見つけるために (いわゆる $R = 10^4$ の時基数補数 $R - Y$)

$R - Y = \bar{Y} + ulp$ という表現を用いる。

桁の補は 9 の補数で、$\bar{Y} = 7654$。

従って $\bar{Y} + 1 = 7655$ である。この結果は、$Y + \bar{Y}$ を計算することにより、$2345 + 7655 = 10^4 = 0\ mod\ 10^4$ 検証できる。

1.4.1 2 の補数表現

k=n の時、２の補数方式によるの数表現の範囲：

$$-2^{n-1} \leq X \leq (2^{n-1} - ulp)$$

$$ulp = 2^0 = 1$$

この範囲は正数より負数の方が一個多いから、わずか不対称である。

２進数 $-2^{n-1}(10\ ...\ 0$ で表す) は同じ値を有する正数を持ってない。

従って、この数に対して補数演算はすれば、桁溢れは必ず起こる。

０の表現は唯一である。

２の補数表現の値：

２の補数表現による数列 $(x_{n-1},x_{n-2},\ldots,x_0)$、下の手順で X の値を求める。

$x_{n-1} = 0$ のとき、 $X = \sum_{i=0}^{n-1} x_i 2^i$。

x_{n-1}=1, 与えられた数列の補数を求めることにより絶対値を得る。

[例 1.7]

1011 が与えられたとき、その補数を求めると $0100 + 1 = 0101$ になる。

数列 0101 の値が 5 であるから、本来の数列の値が -5 である。

２の補数の値（一般式：

上記の手続きの代りに、次の式を利用することができる。

$$X = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i \qquad (1.12)$$

1011 を式 (1.12) を適用すると、
$-8 + 2 + 1 = -5$ が得られる。

(1.12) の証明：

$x_{n-1} = 0$ の場合、同じ正数が計算される。

$x_{n-1} = 1$ の場合、

$$
\begin{aligned}
-[\bar{X} + ulp] &= -[\sum_{i=0}^{n-2} \bar{x}_i 2^i + 1] \\
&= = -[\sum_{i=0}^{n-2} (1 - x_i) 2^i + 1] \\
&= -[\sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} x_i 2^i + 1] \\
&= -[(2^{n-1} - 1) - \sum_{i=0}^{n-2} x_i 2^i + 1]
\end{aligned}
$$

$$= \quad -2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

結果は、式 (1.12) 右側が $x_{n-1} = 1$ の時と同じである。

1.4.2 １の補数表現

$k = n$ の時、１の補数法に表される数範囲は左右対称である。

式は：

$$-(2^{n-1} - ulp) \leq X \leq (2^{n-1} - ulp)$$
$$ulp = 2^0 = 1$$

０に対して二つの表現があり、

$000\cdots0$ は正の 0、$111\cdots1$ は負の 0 をそれぞれ表す。

１の補数の値：

１の補数も２の補数に似た式がある。

$$X = -x_{n-1}(2^{n-1} - ulp) + \Sigma_{i=0}^{n-2} x_i 2^i$$

例えば、４ビットの 1010 は値
$-(2^3 - 1) + 2 = -5$ を表す。

１の補数の導出は２の補数の導出より簡単で、

各ビットに対して同時に $\bar{x}_i = 1 - x_i$ ブール補数として計算される。

1.5 **加算と減算**

符号 - 絶対値表現による加算と減算を行うとき、大きさを表すビットだけが算術演算に関わり、符号ビットが別に扱われる。

符号ー絶対値数表現の桁溢れ問題：

計算結果として、上げ桁溢れ [carry-out](借り桁溢れ [borrow-out]）がオーバフローを示す。

```
    0     1011          11
    0    +0110           6
  ----------------------
    0    10001           1 Carry-out
```

最終結果は正数（２つの正数の和）になるが、４ビットによる大きさが 17 の代わりに１になってしまった。なぜなら、$1 = 17\ mod\ 16$ だから。

補数表現の桁溢れ：

補数表現法において、符号桁も含めたすべての桁が加算や減算に参加する。

このような数系においてオーバフローの判断には上げ桁溢れが要らない。

例として、２の補数法で $X = 13$ と $Y = -7$ の加算を表すと、

```
    01101      13
   +11000      -8
  ----------------------
   100101       5 Carry-out but no overflow
```

上げ桁溢れが切り捨てられたが、オーバフローにならない。

事実上に、符号が違うＸとＹとの加算について、桁上げがあるかどうかに関わらず、オーバフローは生じない。

二つの例：

```
        00101        5
       +10110      -10
-----------------------
        11011       -5 No carry-out

        01010       10
       +11011       -5
-----------------------
       100101        5 Carry-out
```

桁溢れの条件：

もし、X、Ｙが同じ符号を持って、また、結果の符号が二つの演算数のと異なるならば、オーバフローが発生する。

例えば、

```
      11001      -7
     +10110     -10
-----------------------
     101111      15  Carry-out and overflow

      00111       7
     +01010      10
-----------------------
      10001     -15  No carry-out but overflow
```

１の補数の桁溢れ補正：

１の補数法において、桁溢れは補正が必要である。例えば、正数 X と負数 $-Y$ の加算を１の補数で表現した場合、

$$X + (2^n - ulp) - Y = (2^n - ulp) + (X - Y)$$

であり、$X > Y$ ならば $X - Y$ を得る。2^n は、桁溢れビットである。これは、最終結果が長さ n のレジスタに収められるので、切り捨てる。

結果として、$(X - Y - ulp)$ で、必要な補正は、ulp を加えることによって行われる。たとえば

```
            01010            10
           +10110            -5
          ----------------------
        1   00100
Correction +                  1   ulp
          ----------------------
            00101            5
```

生成された Carry-out が end-around carry と呼ばれる。

もし Carry-out がなければ、補正が必要ない。

この場合は、$X < Y$ なら、結果は負数
$-(Y - X)$ である。

$(2^n - ulp) - (Y - X)$ で表現する。

```
   00101              5
  +10101            -10
 ---------------------
   11010             -5  No carry-out,no correction
```

どの補数法においても、減算 $X - Y$ は Y の補数を X に加えることによって実現される。

加算に必要な回路：

１の補数法場合、これが $\bar{Y}$ と X の加算、すなわち、$X - Y = X + \bar{Y}$ である。

２の補数法においては、$X - Y = X + (\bar{Y} + ulp)$ を実行する。 ulp が２進加算器に入力することによって。この演算はただ一つの加算器の演算が必要となる。

1.6 **算術シフト演算**

負数を表現するための３つの方式を区別するもう一つ方法は、与えられた数の左、右への無限の数列を考えることである。

符号 - 絶対値方法により、絶対値 $x_{n-2},...,x_0$ が無限の連続と見られる。

$$...0,0,\{x_{n-2},...,x_0\},0,0,...$$

任意の演算結果により前置きが０でない場合、オーバフローが引き起こす。

基数補数における無限の拡張は

$$\ldots x_{n-1},x_{n-1},\{x_{n-1},\ldots,x_0\},0,0,\ldots$$

となり、ここで x_{n-1} は符号桁である。

最後、減少基数補数において、拡張は次のようになる。

$$\ldots\ x_{n-1},x_{n-1},\{x_{n-1},\ldots,x_0\},x_{n-1},x_{n-1},\ldots$$

[例 1.8]

２の補数法により数列 1011.,11011.0, と 111011.00 が全部数値 −5 を表している。

同様に１の補数法により数列 1010.,11010.1, と 111010.11 も全部数値 −5 を表している。

[例 1.9]

２の補数法において

$$Sh.L\{00101_2 = +5\} = 01010_2 = +10$$

$$Sh.R\{00101_2 = +5\} = 00010_2 = +2$$

$$Sh.L\{11011_2 = -5\} = 10110_2 = -10$$

$$Sh.R\{11011_2 = -5\} = 11101_2 = -3$$

１の補数法において

$$Sh.L\{11010_2 = -5\} = 10101_2 = -10$$
$$Sh.R\{11010_2 = -5\} = 11101_2 = -2$$

算術シフト演算は乗算と除算アルゴリズムのためとても役に立つ。

[第１章 終]