2.5 補数

ある決まった正の数 (普通は基数のベキ数が用いられる) からある正の数を引くことで負の数を表現することを**補数表示**といい、補数表示された負の数を**補数**(complement) といいます。n 進数の場合、決まった正の数には $n^k (k > 0, n^k > |-a|)$ が用いられ、負の数 $-a$ の補数は $n^k - a$ で与えられます。例えば、10 進数の場合に 3 桁 (ある決まった正の数を $10^3 = 1000$) で -10 を補数表示すると $990 (= 1000 - 10)$ となります。

ここで、補数を使った演算 (引き算) を考えて見ましょう。例えば、$30 - 10$ は

$$30 - 10 = 30 + (-10)$$

と考えることができ、-10 の補数 990 を使って 3 桁で補数表示すると

$$030 + 990$$

と書き直すことができます (補数を使って演算を行うときは、正の数も k 桁で表示)。計算すると 1020 となりますが、補数を使った演算では有効桁数 3 桁 (k 桁) で考えるため、第 4 桁の 1 を無視することで $20 (= 020)$ という正しい値が求められます。もう 1 つの例として $5 - 10$ を考えてみましょう。同様に 3 桁で補数表示すると

$$005 + 990 = 995$$

となりますが、普通の数に戻すために補数を求める方法と逆の計算を行えば $1000 - 995 = 5$ となり、-5 という正しい値が求められることがわかります。以上のように、補数表示を用いることによって、負の数を正の数として、引き算を足し算として扱うことができるようになります。そのため、演算装置に減算回路が不必要となり、回路が単純になるという利点があります。この事については、3 章及び 4 章で詳しく説明します。

補数表示についてもう少し詳しく見ていきましょう。上記の例で、-10 は補数表示によって 990 と表されましたが正の数 990 とも解釈できます。そこで、補数表示では半分を正の数、残り半分を負の数として扱うのが普通で、0 から $n^k - 1$ の内、

$$\begin{array}{lll} 0 & \text{から} & \dfrac{n^k}{2} - 1 \quad \text{を正の数} \\ \dfrac{n^k}{2} & \text{から} & n^k - 1 \quad \text{を負の数} \end{array}$$

として扱います (注意点として、演算を行うときは上記の範囲内で行う必要があります)。従って、上記の例では表 2.6 のように対応しています。また、正の数であるか負の数であるかを判別するには、第 3 桁が 0, 1, 2, 3, 4 のとき正の数、5, 6, 7, 8, 9 のとき負の数となります。一般の n 進数の場合は、第 k 桁が 0 から $\dfrac{n}{2} - 1$ のとき正の数を表し、$\dfrac{n}{2}$ から $n - 1$ のとき負の数を表します。

実際の数	-500	-499	-498	$\cdots$	-3	-2	-1	0	1	2	3	$\cdots$	498	499
補数	500	501	502	$\cdots$	997	998	999	000	001	002	003	$\cdots$	498	499

表 2.6: 補数表現

コンピュータが扱う 2 進数の補数表示については、もう一つおもしろい性質があります。例えば、$-1011(2)$ を 8 桁で補数表示するためには

$$100000000(2) - 1011(2)$$

を計算すれば良いわけですが、

$$(100000000(2) - 1(2)) - 1011(2) + 1(2) = 11111111(2) - 1011(2) + 1(2)$$

と考えて計算してみましょう。この計算を完了するには、最初に $11111111(2) - 1011(2)$ を求めて、その後で $1(2)$ を足せば良いことがわかります。最初の計算は

$$\begin{array}{rr} & 11111111(2) \\ -) & 00001011(2) \\ \hline & 11110100(2) \end{array}$$

となりますが、注意深く観察すると $00001011(2)$ と $11110100(2)$ は 1 と 0 をひっくり返しただけで、3 章で扱う論理演算 (否定) で求められることがわかります。このような数を **1 の補数**と呼びます。また、1 の補数に対して今まで求めてきた補数 (1 の補数に 1 を足した数) を **2 の補数**と呼びます。一般の n 進数の場合にも、n の補数と $n-1$ の補数[6]があり、

$$「n \text{ の補数}」 = 「n-1 \text{ の補数}」 + 1$$

が成り立ちます。上記の例、10 進数 -10 について 9 の補数を求めると

$$\begin{array}{rr} & 999 \\ -) & 010 \\ \hline & 989 \end{array}$$

となり、各桁で引き算を行うことで求まります。もちろん、9 の補数に 1 を足せば 10 の補数になることは明らかです (この方法だと計算が楽で計算ミスが少なくなります)。

例題 1　$-123(10)$ について 10 の補数及び 9 の補数を求めなさい。ただし、桁数は 4 桁。

解答例　10 の補数　$10000-123=9877(10)$
　　9 の補数　$9999-123=9876(10)$

例題 2　$-1010100(2)$ について 2 の補数及び 1 の補数を求めなさい。ただし、桁数は 8 桁。

解答例　2 の補数　$100000000-1010100=10101100(2)$
　　1 の補数　$11111111-1010100=10101011(2)$

問題 1　$-123(10)$ を 8 桁の補数で表しなさい。

問題 2　$-123(4)$ を 4 桁の補数で表しなさい。

問題 3　$-1000(10)$ を 16 桁の 2 の補数で表しなさい (2 進数に直しなさいという意味)。

問題 4　補数表示された $1111000011110000(2)$ を普通の 10 進数で答えなさい。

[6] $n-1$ の補数では 0 の表現が 2 種類あり、$+0$ 及び -0 と呼ばれます。

2.6 シフト演算

3 節の例題 5 及び例題 6 に挙げた、基数のベキ乗数で掛けたり割ったりする演算を、情報科学では**シフト演算**と呼んでいます。例えば、1234.5678×10^3 の値は、小数点の位置を右に 3 桁移動させることで 1234567.8 と直ちに答えることができますし、$1234.5678 \div 10^3$ の値は、逆に小数点の位置を左に 3 桁移動させることで 1.2345678 と答えることができます。前者は、小数点を基準にすると数値を左に動かしているため**左シフト演算**と呼ばれ、後者は、小数点を基準に数値を右に動かしているため**右シフト演算**と呼ばれます。また、実際のシフト演算では、シフト演算する回数 (k 回) を付加して、k 回左シフト演算や k 回右シフト演算といいます。上記の例では、前者は 3 回左シフト演算といい、後者は 3 回右シフト演算といいます。

例題 1 123(10) を 2 回右シフト演算しなさい。

解答例 1.23(10)(= 123(10) ÷ 100(10))

例題 2 123(8) を 4 回左シフト演算しなさい。

解答例 1230000(8)(= 123(8) × 10000(8))

例題 3 123(16) を 6 回右シフト演算しなさい。

解答例 0.000123(16)(= 123(16) ÷ 1000000(16))

コンピュータは、掛け算や割り算の代わりに、シフト演算と足し算 (引き算は補数を用いる) を使って積や商を計算します。

掛け算 12×233 をシフト演算と足し算のみを使って計算してみましょう。12×233 は 12 を 233 回足すことで求まりますが、

$$\begin{aligned} 12 \times 233 &= 12 \times 200 + 12 \times 30 + 12 \times 3 \\ &= (12 + 12) \times 100 + (12 + 12 + 12) \times 10 + (12 + 12 + 12) \\ &= ((12 + 12) \times 10 + (12 + 12 + 12)) \times 10 + (12 + 12 + 12) \end{aligned}$$

と変形できることから、次の手順によって積を効率的に求めることができます。

① 12 を 2 回足す $(12 + 12 = 24)$
② 24 に対して 1 回左シフト演算を行う $(24 \times 10 = 240)$
③ 240 に 12 を 3 回足す $(240 + 12 + 12 + 12 = 276)$
④ 276 に対して 1 回左シフト演算を行う $(276 \times 10 = 2760)$
⑤ 2760 に 12 を 3 回足す $(2760 + 12 + 12 + 12 = 2796)$

2 進数の場合はもっと簡単で、例えば $1011(2) \times 1010(2)$ は

$$\begin{aligned} 1011 \times 1010 &= 1011 \times 1000 + 1011 \times 10 \\ &= (1011 \times 100 + 1011) \times 10 \end{aligned}$$

と変形し、以下の手順によって積 1101110(2) を求めることができます。

①1011(2) に対してを 2 回左シフト演算を行う (101100(2))
② 101100(2) に 1011(2) を足す (101100(2) + 1011(2) = 110111(2))
③ 110111(2) に対してを 1 回左シフト演算を行う (1101110(2))

割り算　72207 ÷ 355 をシフト演算と足し算のみを使って計算してみましょう。72207 ÷ 355 は、言い換えると 72207 から何回 355 を引くことができるかという問題なります。上記のように引く回数を数えても良いのですが、下記の手順のようにシフト演算を用いることで、効率的に商を求めることができます。

①355 に対してを 2 回左シフト演算を行う (35500)
② 72207 から 2 回 35500 を引く ($72207 - 355 \times 100 \times 2 = 1207$)
　*72207 から 200 回355 を引いたことと同じになります。
③355 に対して 1 回左シフト演算を行う (3550)
④1207 から 0 回 3550 を引く ($1207 - 355 \times 10 \times 0 = 1207$)
⑤355 に対して 0 回左シフト演算を行う (355)
⑥ 1207 から 3 回 355 を引く ($1207 - 355 \times 1 \times 3 = 142$)

①から⑥の手順を実行することで商203 余り142 を得ますが、さらに下記の手順を実行することで商203.4 まで求めることもできます。

⑦355 に対して 1 回右シフト演算を行う (35.5)
⑧ 142 から 4 回 35.5 を引く ($142 - 355 \times 0.1 \times 4 = 0$)
　*142 から 0.4 回355 を引いたことと同じになります。

2 進数の場合も同様に考えると、1101110(2) ÷ 1010(2) の商1011(2) は、以下の手順によって求めることができます。

①1010(2) に対して 3 回左シフト演算を行う (1010000(2))
② 1101110(2) から (1 回)1010000(2) を引く
$$(1101110(2) - 1010(2) \times 1000(2) \times 1 = 11110(2))$$
③1010(2) に対して 2 回左シフト演算を行う (101000(2))
④ 11110(2) から 0 回 101000(2) を引く
$$(11110(2) - 1010(2) \times 100(2) \times 0 = 11110(2))$$
⑤1010(2) に対して 1 回左シフト演算を行う (10100(2))
⑥ 11110(2) から (1 回)10100(2) を引く ($11110(2) - 1010(2) \times 10(2) \times 1 = 1010(2)$)
⑦1010(2) に対して 0 回左シフト演算を行う (1010(2))
⑧ 1010(2) から (1 回)1010(2) を引く ($1010(2) - 1010(2) \times 1(2) \times 1 = 0(2)$)

問題 1　11111111(2) に対して 4 回右シフト演算を行うといくつになるか 10 進数で答えなさい。

問題 2　60752(8) ÷ 123(8) を例にならってシフト演算を用いて計算しなさい。

問題 3　A45F(16) × CDE(16) を例にならってシフト演算を用いて計算しなさい。

2.7 数値データ

コンピュータが直接扱うことのできるデータは、0 と 1 を有限個並べて表現する 2 進数ですが、構造の違いにより**固定小数点表示**と**浮動小数点表示**と呼ばれる表示方法があります。また、各表示方法によって表現された数を、**固定小数点数**及び**浮動小数点数**とそれぞれ呼びます。

固定小数点数 この表示方法は、主に整数を表示する場合に用いられ、1 語を 8, 16, 32, 64 などのビット数で表します。例えば、k ビットで表すことのできる 0 と 1 の組み合わせ数は 2^k 通りですから、自然に 2 進数に対応させれば 0 から $2^k - 1$ までの整数を表すことができます。また、補数表示された数として扱えば、-2^{k-1} から $2^{k-1} - 1$ まで表示することができます。以上のような 2 つの場合は、小数点が第 0 ビットの右側にある (固定されている) と仮定していますが、小数点の位置はどこに仮定しても良いことから、小数[7]を扱うこともできます (問題 2)。逆にいえば、上記の 2 つの例は、小数点が第 0 ビットの右側に固定されている特殊な場合といえます。

浮動小数点数 この表示方法は、実数を表示する場合に用いられ、固定小数点表示に比べ非常に大きな数や非常に小さな数を表示するのに適しています。科学の分野では、

$$-1.0 \times 10^{16}, \quad 2.5 \times 10^{-8}$$

といった表示方法を用いますが、これが浮動小数点表示です。この方法に習い、

$$M \times B^E$$

で表すと、M は**仮数部**(mantissa)・E は**指数部**(exponent)・B は**底**(base) といい、図 2.3 のような構造で浮動小数点表示を行います。指数部は、仕様により決まった数になるので省略されますが、2, 8, 16 などの 2 のベキ乗数が使用されます。なお、浮動小数点表示には精度によって、1 語を 32 ビットで表す**単精度浮動小数点表示**と、1 語を 64 ビットで表す**倍精度浮動小数点表示**があります。

S ：仮数部の符号を 1 ビットで表します (正 $\to$ 0, 負 $\to$ 1)。
E ：指数を 2 進数で補数表示します (ビット数は表示方法によります)。
$|M|$ ：仮数を 2 進数で表示します ($|M| \geq 0$ より、補数表示しません)。

図 2.3: 浮動小数点数の構造

浮動小数点数について詳しく見ていきましょう。現在使用されている主な浮動小数点表示には「**IBM 方式**」と「**IEEE 方式**」[8]があります。

IBM 方式 System360 以来、メインフレーム (汎用コンピュータ) で採用された方式で、底を 16 とし、単精度及び倍精度の指数部と仮数部のビット数は表 2.7 を採用しています。

[7]現在では固定小数点数と浮動小数点数の処理時間の差がなくなったため、小数は浮動小数点数で扱います。
[8]the Institute of Electrical and Electronic Engineers inc. の略で、米国電気電子技術者協会を示す。

	単精度	倍精度
仮数の符号	1 ビット	1 ビット
指数部	8 ビット	8 ビット
仮数部	23 ビット	55 ビット
合計	32 ビット	64 ビット

表 2.7: IBM 方式の浮動小数点表示

IEEE 方式　高精度の数値計算を最適に実行するために米国電気電子技術者協会で提唱した規格で、現在、多くのワークステーションやパーソナルコンピュータで採用されている方式です。単精度及び倍精度の指数部と仮数部のビット数は表 2.7 のとおりで、断りのない場合はこの方式に従います。

	単精度	倍精度
仮数の符号	1 ビット	1 ビット
指数部	8 ビット	12 ビット
仮数部	23 ビット	51 ビット
合計	32 ビット	64 ビット

表 2.8: IEEE 方式の浮動小数点表示

浮動小数点表示では、浮動小数点数が一意かつ有効数字が長くなるように**正規形**という形をとります。例えば、$\frac{1}{3}$ の近似として

$$0.3333 \times 10^0, \quad 0.03333 \times 10^1, \quad 0.003333 \times 10^2$$

は全て同じ数ですが、表現が異なっています。このとき、0. で始まる形として一意かつ有効数字が長くなる表現は、一番左の 0.3333×10^0 であることがわかります。その他の例としては、

$$0.123 \times 10^{-10}, \quad -0.5 \times 10^{17}, \quad 0.9999 \times 10^{-30}$$

があります。上記に習って、0.000001010111(2) を正規形に直すと

$$0.1010111(2) \times 2^{-5} = 0.1010111(2) \times 2^{11111011(2)}$$

となります。従って、定義どおり浮動小数点表示を行うと

0	11111011	10101110000000000000000

となりますが、IEEE 方式では、仮数部の上位 1 ビットは必ず 1 となるので、この 1 も省略して

0	11111011	01011100000000000000000
		^
		1

のように表します。この工夫により、IEEE 方式では有効桁数が 1 ビット長くなります。逆に、IBM 方式では、底を 16(2 進数の 4 ビットに相当) として処理するため、0.000001010111(2) を正規形に直す (小数点を 4 ビットづつ移動する) と

$$0.01010111(2) \times 16^{-1} = 0.01010111(2) \times 16^{11111111(2)}$$

となり、単精度浮動小数点表示を行うと

0	11111111	01010111000000000000000

となります。従って、この場合は IEEE 方式と比べて有効桁数を 2 ビット損しています (最大 4 ビット損をします)。なお、IBM 方式及び IEEE 方式共通の注意点として、0(2) を浮動小数点表示する場合、全てのビットは 0 になります。

例題 1 $-110110101(2)$ を IBM 方式及び IEEE 方式で単精度浮動小数点表示しなさい。

解答例

IBM 方式： $110110101(2) = 0.000110110101(2) \times 16^{3}$

$= 0.000110110101(2) \times 16^{00000011(2)}$

従って、仮数の符号が負であることに注意して

1	00000011	00011011010100000000000

IEEE 方式： $110110101(2) = 0.110110101(2) \times 2^{9}$

$= 0.110110101(2) \times 2^{00001001(2)}$

従って、仮数の符号が負であることに注意して

1	00001001	10110101000000000000000
		∧ 1

IEEE 方式の単精度浮動小数点数の内、0 を除く絶対値の一番大きな数は

0	01111111	11111111111111111111111

$$0.11111111111111111111111(2) \times 2^{01111111(2)}$$

で表され、一番小さな数は

0	10000000	00000000000000000000000

$$0.10000000000000000000000(2) \times 2^{10000000(2)}$$

で表されます。10 進数に直すと、表示可能な数値の概略の大きさは約 10^{-38} から 10^{38} で、有効桁数は約 7 桁になります。

表示可能な数値の概略の大きさの求め方は、指数部が 01111111(2)(= 127) であることから、

$$2^{127} = 10^x$$

が成り立ち、両辺に常用対数をとることで

$$x = \log_{10} 2^{127} = 127 \times 0.301 = 38.227$$

が求まります。同様に、10000000(2)(= −128) について計算すれば −38 が求まります。また、有効桁数を求めるには、仮数部を (23+1) ビットで表すことから

$$2^{24} = 10^y$$

が成り立ち、両辺に常用対数をとることで

$$y = \log_{10} 2^{24} = 24 \times 0.301 = 7.224$$

が求まります。

問題 1　64 ビットで固定小数点表示したとき、補数表示した場合としない場合について最大値と最小値をそれぞれ求めなさい。

問題 2　8 ビットで固定小数点表示したとき、小数点が第 4 ビットと第 3 ビットの間にあると仮定し、補数表示しない場合について最大値と最小値を求めなさい (補数表示した場合についても考察しなさい)。

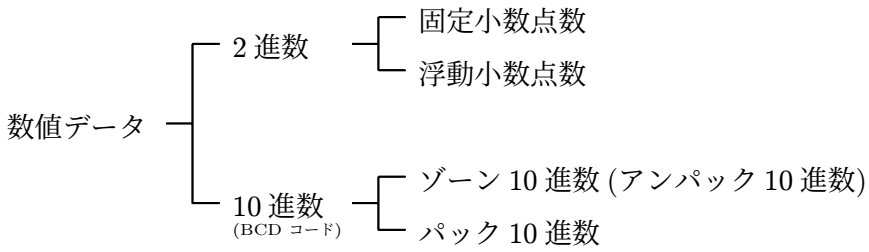
問題 3　0.00000000001(2) を IBM 方式及び IEEE 方式で単精度浮動小数点表示しなさい。

問題 4　IEEE 方式の倍精度浮動小数点数の内、0 を除く絶対値の一番大きな数と絶対値の一番小さな数を求め、仮数部の有効桁数と表示可能な数値の範囲を求めなさい。

問題 5　IBM 方式の単精度浮動小数点数の内、0 を除く絶対値の一番大きな数と絶対値の一番小さな数を求め、仮数部の有効桁数と表示可能な数値の範囲を求めなさい (最大 4 ビット有効桁数が短くなることに注意しなさい)。

● 参考資料 — BCD コード —

コンピュータ内部での数値表示には、固定小数点表示や浮動小数点表示以外にも、10 進数を表示するのに適した **BCD コード**(binary coded decimal notation code) があります。主なもとしては、**ゾーン 10 進数**(**アンパック 10 進数**) と**パック 10 進数**があります。これらの表示方法は、大量の桁の 10 進数を正確に扱うことができ、財務など多額の金額を正確に扱う必要のあるビジネス分野などで利用されています。



下図のように、**ゾーン形式**(**アンパック形式**) によるゾーン 10 進数は、10 進数の 1 桁分を 1 バイトで表しますが、上位 4 ビットはゾーン部として定数が入り、下位 4 ビットには0000(2)=0 から1001(2)=9 までのいずれかの値が入ります (ただし、10 進数の 1 桁目に相当する部分のゾーン部は符号を表す)。逆に、**パック形式**によるパック 10 進数は、ゾーン 10 進数からゾーン部を取りのぞき、無駄な部分を省いた形で表します (ただし、後ろに 4 ビット付加して符号を表す)。なお、ゾーン 10 進数からパック 10 進数への変換をパックと呼び、逆の変換をアンパックと呼びます (下図参照)。