情報処理基礎講座

コンピュータシステムの基礎 第6版

電子開発学園出版局

情報処理基礎講座

コンピュータシステムの基礎 第6版

電子開発学園出版局

＊ 本書に掲載した会社名・製品名等は、一般にそれぞれ各社の商号・登録商標または商標です。

はじめに

コンピュータが開発されて以来、コンピュータは多くの分野で利用されてきました。現在では例えば、旅券を発行する、ご飯を炊く、ゲームをする、ニュースを知る、電話をするなど私たちの生活の中に根付いた分野で利用され、なくてはならない物として、また身近な物として存在しています。また現在のネットワーク社会はコンピュータなくして成り立ちません。

コンピュータはこれらの多くの分野で実現する様々な事柄を実現するために、“情報”を取り扱い、制御し、処理しています。コンピュータで取り扱い、処理する“情報”とはいったいどのようなもので、それを取り扱い、処理するためにどのような技術が使用されているのでしょうか。本書『コンピュータシステムの基礎 第６版』および姉妹書の『ネットワークとシステム開発の基礎 第２版』では、このような知識、技術を“インフォメーションテクノロジ（IT）”と総称しております。

この“インフォメーションテクノロジ”を学ぶことは、様々な情報とそれを取り扱うコンピュータを学ぶことであり、それにより現代社会の基盤となるコンピュータについて理解を深めることとなり、そしてその社会についても知ることになります。

本書は、多くの分野から成立している“インフォメーションテクノロジ”のうち、その基礎理論、コンピュータを成立させているハードウェアやソフトウェアなどのシステムについて、基礎から学ぶためのテキストであります。

また、情報処理技術者試験における「基本情報技術者試験」の出題範囲のうち、基礎理論、コンピュータシステム、技術知識の分野を包含しており、姉妹書『ネットワークとシステム開発の基礎 第２版』と併せ、“テクノロジ系”の範囲を網羅しております。

本書によって、情報とはなにか、情報技術とはなにか、コンピュータはどのようなものかを学び、理解を深め、情報処理技術者試験の取得に役立てていただければ幸いです。

編著者

目　　次

はじめに

第１部　IT の基礎理論 1

第１章　基礎数学 2
１．１　離散数学 2
１．１．１　基数 2
１．１．２　数値の表現 11
１．１．３　算術演算と精度 25
１．１．４　集合 33
１．１．５　論理演算 41
１．２　応用数学 50
１．２．１　確率 50
１．２．２　統計 58
１．２．３　数値と数式 69
１．２．４　グラフと待ち行列 79

第２章　情報理論 83
２．１　情報に関する理論 83
２．１．１　情報と符号 83
２．１．２　情報の理論 92

第３章　通信と制御の理論 109
３．１　通信に関する理論 109
３．１．１　伝送理論 109
３．２　制御に関する理論 124
３．２．１　信号の処理 124
３．２．２　制御の理論 124

第２部　コンピュータとシステム ･･･ 127

第４章　コンピュータの構成要素 ･･ 128
４．１　コンピュータの構成 ･･ 128
４．１．１コンピュータの種類 ･･ 128
４．１．２　コンピュータの構成 ･･ 131
４．２　メモリ ･･･ 134
４．２．１　メモリの種類と特徴 ･･ 134
４．２．２　メインメモリの構成 ･･ 139
４．２．３　メモリシステムの構成 ･･ 141
４．３　プロセッサ ･･･ 146
４．３．１　プロセッサの構造と方式 ･･ 146
４．３．２　プロセッサの動作原理 ･･ 155
４．３．３　プロセッサの性能評価 ･･ 163
４．３．４　プロセッサの技術 ･･ 166
４．３．５　プロセッサの種類 ･･ 171
４．４　入出力装置 ･･･ 174
４．４．１　バス ･･･ 174
４．４．２　入出力インタフェース ･･ 177
４．４．３　入出力装置 ･･･ 183
４．５　補助記憶装置 ･･･ 192
４．５．１　記憶媒体とファイル ･･ 192
４．５．２　磁気ディスク装置 ･･ 193
４．５．３　光ディスク装置 ･･･ 198
４．５．４　その他の補助記憶装置 ･･ 200

第５章　システムの構成要素 ･･･ 203
５．１　システムの形態 ･･･ 203
５．１．１　処理形態 ･･･ 203
５．１．２　利用形態 ･･･ 209
５．１．３　操作形態 ･･･ 213
５．２　システムの構成 ･･･ 214
５．２．１　信頼性の向上を目指したシステム構成 ･･････････････････････････････ 214
５．２．２　信頼性の向上を目指したサイト構成 ････････････････････････････････ 217
５．２．３　処理能力の向上を目指したサイト構成 ･･････････････････････････････ 218

５．３　信頼性設計 …… 219
５．３．１　ハードディスク技術 …… 219
５．３．２　信頼性設計 …… 223
５．４　システムの評価 …… 224
５．４．１　システムの性能指標と評価 …… 224
５．４．２　システムの信頼性指標と評価 …… 228
５．４．３　システムの経済性指標と評価 …… 232

第６章　電気と制御 …… 233
６．１　電気・電子回路などハードウェア …… 233
６．１．１　電気・電子回路 …… 233

第３部　オペレーティングシステムとソフトウェア…………235

第７章　オペレーティングシステム……………………………………………………236
７．１　ＯＳの基礎……………………………………………………………………236
７．１．１　ＯＳの種類と特徴………………………………………………………236
７．１．２　ＯＳの構成と機能………………………………………………………240
７．２　ＯＳの管理機能………………………………………………………………242
７．２．１　ジョブとタスク…………………………………………………………242
７．２．２　データと記憶……………………………………………………………253
７．２．３　ネットワークと運用……………………………………………………265

第８章　ミドルウェア……………………………………………………………………273
８．１　ミドルウェアの機能…………………………………………………………273
８．１．１　ミドルウェアの機能……………………………………………………273

第９章　ファイルシステム………………………………………………………………279
９．１　ファイルシステムの機能……………………………………………………279
９．１．１　ファイルシステムの種類………………………………………………279
９．１．２　ファイル編成とアクセス手法…………………………………………283
９．１．３　バックアップ……………………………………………………………287

第10章　開発ツールとオープンソースソフトウェア…………………………………289
１０．１　開発ツール…………………………………………………………………289
１０．１．１　ソフトウェア開発支援ツール………………………………………289
１０．１．２　ＥＵＣとＥＵＤ………………………………………………………295
１０．１．３　言語処理ツール………………………………………………………296
１０．２　オープンソースソフトウェア……………………………………………301
１０．２．１　ライセンス……………………………………………………………301
１０．２．２　オープンソースソフトウェアの種別………………………………304

第 11 章　ヒューマンインタフェース……307
１１．１　ヒューマンインタフェース技術……307
１１．１．１　ＩＡとヒューマンインタフェース……307
１１．２　インタフェース設計……313
１１．２．１　画面設計・帳票設計……313
１１．２．２　コード設計……317
１１．２．３　Ｗｅｂデザイン……320

第 12 章　マルチメディア……323
１２．１　マルチメディア技術……323
１２．１．１　マルチメディア……323
１２．１．２　マルチメディアコンテンツ……325
１２．２　マルチメディア応用……331
１２．２．１　マルチメディアの応用技術……331

【練習問題】ダウンロードのご案内……335

索　引……337

コンピュータシステムの基礎 第6版

第１部

ITの基礎理論

本書は、「はじめに」で記したとおり、多くの分野から成立しているインフォメーションテクノロジ（ＩＴ）のうち、その基礎理論、コンピュータを成立させているハードウェアやソフトウェアなどのシステムについて、基礎から学ぶためのテキストである。
この第１部では、まず、ＩＴの基礎理論として、次の内容について説明する。

・基礎数学（離散数学、応用数学）
・情報理論（情報に関する理論）
・通信と制御に関する理論

これらを最初に学習することにより、コンピュータを含むインフォメーションテクノロジをいっそう理解できるようになる。

第１部

第１章　基礎数学

１．１　離散数学

離散数学とは、連続的ではないバラバラの対象を扱う数学のことであり、アルゴリズムやプログラミングなどコンピュータで扱う数値の分野として知られている。コンピュータを学ぶ前に、離散数学をまず学習することにより、コンピュータを含むインフォメーションテクノロジを理解できるようになる。

１．１．１　基数

（１）基数とは

我々が普段使用している数値は、０～９までの１０種類の組み合わせで表現している。そして、それぞれの桁は１０の累乗（下位の桁から順に０乗、１乗、２乗、３乗…）で重み付けされている。この時、重み付けの基本となる１０を**基数**と呼ぶ。

基数

例）$356_{(10)} = 3 \times 100 + 5 \times 10 + 6 \times 1$

$= 3 \times 10^{2} + 5 \times 10^{1} + 6 \times 10^{0}$

例）$0.284_{(10)} = 2 \times \frac{1}{10} + 8 \times \frac{1}{100} + 4 \times \frac{1}{1000}$

$= 2 \times 10^{-1} + 8 \times 10^{-2} + 4 \times 10^{-3}$

（網掛け：基数）

特にその数値の基数を明記したい場合には、上図の$356_{(10)}$のように右下に括弧書きで基数を書き添える。

我々が普段使用している１０を基数にした数値の表現を１０進数と呼ぶ。

１０進数と同じように、別の数値を基数とした数値がある。例えば２を基数とした数値は２進数、１６を基数とした数値は１６進数となり、Ｎを基数とした数値は**Ｎ進数**と呼ばれる。

(a)　２進数と１６進数

２進数は基数が２であるので、０と１だけの数値の組み合わせで表現する。ただし、２進数では数が大きくなると桁数が非常に増えて見づらくなる。そのため、２進数と変換が比較的簡単な１６進数もよく利用される。１６進数は、０～９、Ａ、Ｂ、Ｃ、Ｄ、Ｅ、Ｆの数字と英字の１６種類の組み合わせで表現される。

また、１６進数は１の場合は“１$_{(16)}$”と表記されるが、“０ｘ１”のように先頭に“０ｘ”をつけて表記することもよく行われる。この本では１$_{(16)}$の表記を使用する。

１０進数、２進数、１６進数を比較すると、次のようになる。

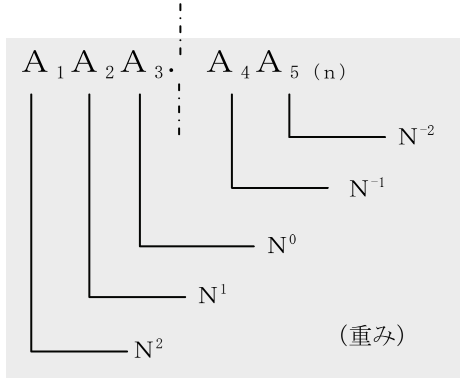
１０進数・２進数・１６進数の比

１０進数	２進数	１６進数
0	0	0
1	1	1
2	１０	2
3	１１	3
4	１００	4
5	１０１	5
6	１１０	6
7	１１１	7
8	１０００	8
9	１００１	9
１０	１０１０	A
１１	１０１１	B
１２	１１００	C
１３	１１０１	D
１４	１１１０	E
１５	１１１１	F
１６	１００００	１０
１７	１０００１	１１
１８	１００１０	１２
１９	１００１１	１３
２０	１０１００	１４

(b)　Ｎ進数

Ｎ進数はＮを基数とした数値である。０～（Ｎ－１）の整数の組み合わせで表現されて、Ｎごとに桁上がりし、各桁ではＮが重みとなる。

Ｎ進数



各桁の数
- ・０～（Ｎ－１）の整数
- ・Ｎごとに桁上がり

$$A_1A_2A_3.\ A_4A_{5\,(n)} = A_1 \times N^2 + A_2 \times N^1 + A_3 \times N^0 + A_4 \times N^{-1} + A_5 \times N^{-2}$$

(c)　Ｎ進数の加減算

Ｎ進数の加減算は、１０進数の計算と同様である。ただし、加算の場合はＮで桁上がりをし、減算の場合は上の桁から借りた場合にＮが足すことが異なる。

１０進数と２進数の加減算（１）

例）１０進数の加算の場合

０	２	３	５
＋２	＋３	＋５	＋８
２	５	８	１３
			(桁上がり)

例）２進数の加算の場合

０	０	１	１
＋０	＋１	＋０	＋１
０	１	１	１０
			(桁上がり)

１０進数と２進数の加減算（２）

例）２進数で複数桁の加算の場合

```
   1 1  (桁上がり)
  10001
+01011
  11100
```

```
 1 1 1  (桁上がり)
 111010
+001001
1000011
```

例）２進数での減算の場合

```
  (1桁上から借り)
  11001
-   101
  10100
```

例題

【問題】

次の計算は何進数で成立するか。

１３１－４５＝５３

【解答】

式を変形すると、「１３１＝５３＋４５」となる。これの、１の桁だけに注目すると、「１＝３＋５」である。これは桁上がりが発生していることがわかるので、「１１＝３＋５」となる。

１１は１０進数ならば「１０＋１」だが、N進数の場合は「N＋１」である。よって、「N＋１＝３＋５」となり、「N＝７」であり７進数であることがわかる。

（２）基数変換

ある基数の数から、別の基数の数に変換することを**基数変換**と呼ぶ。例えば、１０進数を２進数に変換する、またはその逆の２進数から１０進数に変換することが基数変換である。

インフォメーションテクノロジの世界では、特に１０進数、２進数、１６進数をよく使用するため、これらの基数の間での基数変換がよく行われる。

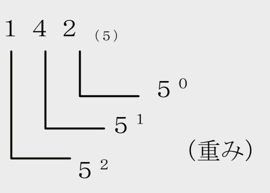
(a)　N進数と１０進数の間での基数変換

（ア）N進数から１０進数へ

N進数から１０進数へ基数変換するには、N進数の各桁に重みを掛け合わせ、各桁から求めた値の総和が１０進数になる。

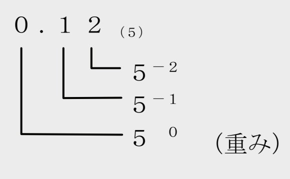
１０進数との基数変換

【整数の場合】例）１４２$_{(5)}$ を１０進数に変換する



$$142_{(5)} = 1\times5^2 + 4\times5^1 + 2\times5^0$$
$$= 25 + 20 + 2$$
$$= 47_{(10)}$$

【小数の場合】例）０．１２$_{(5)}$ を１０進数に変換する



$$0.12_{(5)} = 1\times5^{-1} + 2\times5^{-2}$$
$$= \frac{1}{5} + \frac{2}{25}$$
$$= 0.28_{(10)}$$

（イ）１０進数からN進数へ

【整数の場合】

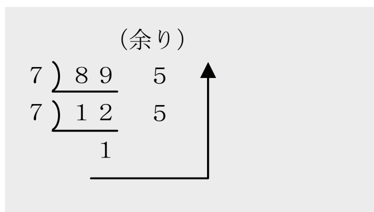
１）与えられた１０進数（Xとする）をNで割り、商と余りを求める。

２）１）で求めた商を新しいXとみなし、さらにNで割り、商と余りを求める。これを商が１になるまで繰り返す。

３）商と余りを求めた逆の順番で並べる。

N進数との基数変換（１）

例）８９$_{(10)}$ を７進数に変換する



商と余りを求めた逆の順番に並べると

$$89_{(10)} = 155_{(7)}$$

となる

【値が小数の場合】

1）１０進数の小数部のみを抜き出した値（Xとする）をN倍して積を求める。

2）求めた積の小数部のみを抜き出して、新たなXとし、N倍して積を求める。これを小数部が０になるまで繰り返す。

3）積の整数部分を並べる。

Ｎ進数との基数変換（２）

例）０．５６８ $_{(10)}$ を５進数に変換する

（桁上がり）

桁上がり	計算	
	０.５６８	
	× 　５	
２ ←	２.８４０	小数部のみ
	０.８４０	
	× 　５	
４ ←	４．２００	小数部のみ
	０.２００	
	× 　５	
１ ←	１.０００	

整数部への桁上がりを並べると

０．５６８ $_{(10)}$ ＝０．２４１ $_{(5)}$

となる

ただし、１０進数の小数は、N進数に変換すると割り切れる数値にならない場合がある。例えば、０．１ $_{(10)}$ を７進数に変換すると、０．０４６２０４６２…と“０４６２”が無限に続くようになる。このように同じ数が繰り返し出てくる小数を**循環小数**と呼び、$0.\dot{0}46\dot{2}_{(7)}$ と表現する。

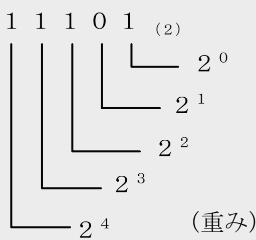
(b)　２進数と１０進数の間での基数変換

インフォメーションテクノロジで最も頻繁に使用するのは２進数と１０進数との間での基数変換である。

（ア）２進数から１０進数へ

１０進数との基数変換

【整数の場合】 例）１１１０１$_{(2)}$ を１０進数に変換する



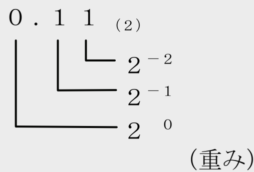
１１１０１$_{(2)}$

$=1\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0$

$=16+8+4+1$

$=29_{(10)}$

【小数の場合】 例）０．１１$_{(2)}$ を１０進数に変換する



０．１１$_{(2)}$

$=1\times2^{-1}+1\times2^{-2}$

$=\frac{1}{2}+\frac{1}{4}$

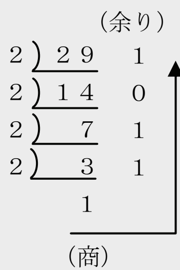
$=0.75_{(10)}$

（イ）１０進数から２進数へ

【値が整数の場合】

２進数との基数変換（１）

例）２９$_{(10)}$ を２進数に変換する



商と余りを求めた逆の順番に並べると

２９$_{(10)}$ ＝１１１０１$_{(2)}$

となる

第１部

【値が小数の場合】

２進数との基数変換（２）

例）０．３７５$_{(10)}$を２進数に変換する

```
(桁上がり)      0.375
                ×   2
  0  ←········  0.750   ⤵ 小数部のみ
                0.750
                ×   2
  1  ←········  1.500   ⤵ 小数部のみ
                0.500
                ×   2
  1  ←········  1.000
```

整数部への桁上がりを並べると

０．３７５$_{(10)}$＝０．０１１$_{(2)}$

となる

(c)　２進数と１６進数の間での基数変換

２進数と同様に１６進数もよく使用される。１６進数の場合、２進数との基数変換が簡単に行える。そのため、１０進数と１６進数の間での基数変換を行う場合、間に２進数をはさみ、１０進数から２進数に変換したうえで１６進数に変換する、もしくは１６進数から２進数に変換したうえで１０進数に変換する、という変換をよく行う。

（ア）２進数から１６進数へ

２進数から１６進数へ変換するときは、小数点を境にして４桁ずつ区切ってブロックとし、ブロックごとを１桁の１６進数に置き換える。

１６進数との基数変換（１）

例）１０１１０１１．１０１$_{(2)}$を１６進数に変換する

```
 ↓ 0を追加              ↓ 0を追加
0101 ┆ 1011 ┆ 1010
  5  ┆   B  ┆   A
```

１０１１０１１．１０１$_{(2)}$

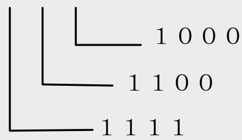
＝５Ｂ．Ａ$_{(16)}$

（イ）１６進数から２進数へ

１６進数から２進数へ変換する場合は、１６進数の各桁を２進数４桁に変換し、先頭から並べる。

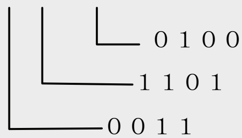
１６進数との基数変換（２）

【整数の場合】 例）ＦＣ８$_{(16)}$ を２進数に変換する



ＦＣ８$_{(16)}$＝１１１１１１００１０００$_{(2)}$

【小数の場合】 例）３Ｄ．４$_{(16)}$ を２進数に変換する。



３Ｄ．４$_{(16)}$＝００１１１１０１．０１００$_{(2)}$

削除（先頭の００と末尾の００）

＝１１１１０１．０１$_{(2)}$

１．１．２　数値の表現

（１）コンピュータでの情報の表現

コンピュータでは数値、文字、画像などのさまざまな情報を取り扱う。まずコンピュータが取り扱う情報の表現方法を説明する。

(a)　情報の表現

（ア）ビット

コンピュータにおける記憶は、“１”と“０”の組み合わせで成り立っている。この“１”と“０”が情報を表現する最小の単位で、**ビット**（bit：binarydigit）と呼ぶ。ビットは“１”と“０”のみしか扱わないので、多くの場合これを２進数の数値として、またはそれを変換して１６進数の数値として表現する。

ビットを複数並べることによって、多くの情報を表現することができる。１ビットでは“１”と“０”の２つの情報だけしか表現できないが、２ビットあれば、“１１”、“１０”、“０１”、“００”の４つの情報を表現できる。

ビットと情報

コンピュータの記憶はスイッチが複数並んでいると考える

ON	OFF	ON
1	0	1

ＯＮを１、ＯＦＦを０とすると３つのスイッチで１０１という情報を保持している

このスイッチが数字を記憶しているとすれば$101_{(2)} = 5_{(10)}$

この３つのスイッチで$5_{(10)}$を記憶していることになる

つまり、ｎビットあれば、2^nのパターンを表現することができることになる。

（イ）バイト

現在のコンピュータでは、人間の使用する文字や数字などの情報を表現するときは多くの場合、８ビットを１つの単位として用いる。これを**バイト**（byte）と呼び、１バイトは$2^8 = 256$通りの情報を表現できる。

（ウ）ワード

コンピュータで一度に取り扱うことのできる情報の大きさを、**ワード**（word：語）と呼ぶ。ワードの大きさはコンピュータの性能によって異なる。例えば、一度に取り扱えるデータの量が１６ビットのコンピュータであれば、そのコンピュータの１ワードは１６ビットで、一度に取り扱えるデータの量が３２ビットのコンピュータであれば、そのコンピュータの１ワードは３２ビットということになる。

(b)　大きさの単位

コンピュータで扱う数値は非常に大きい、または非常に小さい値になることがある。記憶装置に記憶するデータ量では大きな値が使われ、一方でコンピュータの動作時間を表すには小さい値が使われることが多い。そのため、単位に接頭辞と呼ばれる大きさを示すアルファベットをつけることで、表現する桁数を減らすことができる。

例えば、１０Ｋ（キロ）では、１０，０００となり、１万を表す。

接頭辞

接頭辞	記号	大きさ
エクサ	E	10^{18}＝１００京
ペタ	P	10^{15}＝１０００兆
テラ	T	10^{12}＝１兆
ギガ	G	10^{9}＝１０億
メガ	M	10^{6}＝１００万
キロ	K	10^{3}＝１０００
なし		10^{0}＝１

接頭辞	記号	大きさ
ミリ	m	10^{-3}＝１０００分の一
マイクロ	μ	10^{-6}＝１００万分の一
ナノ	n	10^{-9}＝１０億分の一
ピコ	p	10^{-12}＝１兆分の一

（２）コンピュータでの数値の表現

コンピュータではビットで情報が保持される。コンピュータで数字を取り扱う場合には、ビットが“1”と“0”のみを使用するため、２進数での表現方法がコンピュータでは使用される。

一方で、２進数では人間が使用するにはなじみが薄くわかりにくいため、人間にとってわかりやすい１０進数をビットで表現する方法も使用されている。

(a)　１０進数の表現

（ア）ＢＣＤコード

数字を表現する時に１０進数で表現される方法としては、**２進化１０進コード（Binary Coded Decimal：ＢＣＤコード）**が使われる。

ＢＣＤコードは１０進数の各桁をそれぞれ４ビットの２進数で表現する方式である。

ＢＣＤコード

例）１５４１$_{(10)}$を表現する

【２進数で表現した場合】

１５４１$_{(10)}$　＝　１１０００００００１０１$_{(2)}$

【ＢＣＤコードを使った場合】

１	５	４	１	（１０進数）
↓	↓	↓	↓	
０００１	０１０１	０１００	０００１	（ＢＣＤコード）

実際にはこのＢＣＤコードを使用して、**ゾーン１０進数**、**パック１０進数**と呼ばれる表現方法が使用される。

（イ）ゾーン１０進数

ゾーン１０進数は１０進数１桁を１バイト（８ビット）で表現する形式である。各桁を表現する１バイトのうち下位４ビットはＢＣＤコードで表現した数値を入れる。上位４ビットにはゾーンビットと呼ばれる数値を入れる。ただし、桁のうち最下位の桁の上位４ビットは符号を表現する符号ビットをいれる。

ゾーンビットに使用する値は、コンピュータで使用される文字コードによって異なる。文字コードについては第２章で説明する。

ゾーン１０進数

ここではゾーンビットは「１１１１ $_{(2)}$」、符号ビットは正ならば「１１００ $_{(2)}$」、負ならば「１１０１ $_{(2)}$」を入れるものとする。

例）－１９８６ $_{(10)}$ を表現する

1		9		8		6	
ゾーンビット	BCDコード	ゾーンビット	BCDコード	ゾーンビット	BCDコード	符号ビット	BCDコード
1111	0001	1111	1001	1111	1000	1101	0110
F	1	F	9	F	8	D	6

よって、－１９８６ $_{(10)}$ はゾーン１０進数ではＦ１Ｆ９Ｆ８Ｄ６ $_{(16)}$ と4バイトで表現される。

（ウ）パック１０進数

パック１０進数は、１０進数１桁を４ビットで表現する方式である。つまり、１バイトで２桁を表現できる。各桁をゾーン１０進数と同様にＢＣＤコードで表現するが、最下位桁の後ろに符号ビットをつけて符号を表す。ただし、ビット数が８の倍数にならない場合（バイトで表現できない場合）は、００００ $_{(2)}$ を先頭に挿入することで、８の倍数にする。

パック１０進数はゾーン１０進数に比べてゾーンビットを省略して詰めており、少ないバイト数で多くの桁を表現できる。この省略して詰めていることを**パック**されていると呼ぶ。つまり、ゾーン１０進数をパックしたものがパック１０進数であると言える。また逆にゾーン１０進数はパック１０進数に対してパックされていない(**アンパック**されている)とも言える。そのため、ゾーン１０進数のことをアンパック１０進数とも呼ぶ。

第１部

パック１０進数

ここでは符号ビットは正ならば「１１００$_{(2)}$」、負ならば「１１０１$_{(2)}$」を入れるものとする。

例１）１５４１$_{(10)}$を表現する

	1	5	4	1	
	BCDコード	BCDコード	BCDコード	BCDコード	符号ビット
0000	0001	0101	0100	0001	1100
0	1	5	4	1	C

よって、１５４１$_{(10)}$はゾーン１０進数では０１５４１Ｃ$_{(16)}$と３バイトで表現される。

例２）－１９８$_{(10)}$を表現する

1	9	8	
BCDコード	BCDコード	BCDコード	符号ビット
0001	1001	1000	1101

よって、－１９８$_{(10)}$はパック１０進数では１９８Ｄ$_{(16)}$と２バイトで表現される。

(b)　２進数での数値の表現

（ア）正の数の表現

ビットではｎビットあれば、２^n通りの表現ができる。例えば８ビットならば００００００００～１１１１１１１１の表現が可能になる。このビットの並びを、２進数として考えれば８ビットでは正の整数として０$_{(10)}$～２５５$_{(10)}$の数値が表現できることになる。

つまり、ｎビットある場合、正の整数は０～２^n－１までの値を表現できることになる。

（イ）負の数の表現（符号付き絶対値方式）

ｎビットで負の数を表現する方法には２通りある。１つは、符号をつける方式で、**符号付き絶対値方式**と呼ぶ。

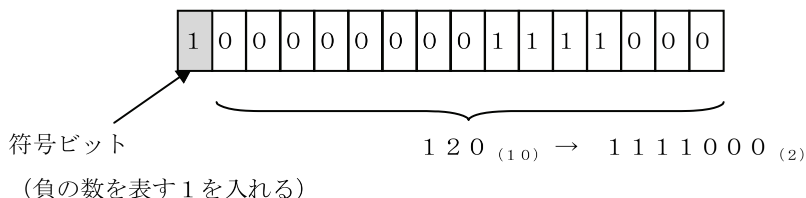
符号付き絶対値方式では一番先頭のビットを符号として扱う。先頭のビットを符号ビットと呼び、正の数ならば０を、負の数ならば１をこのビットに入れ、残りのビットで数値を表す。

符号付き絶対値方式で表すことができる数の範囲は、１ビットを符号ビットとして扱うため、ｎビットの場合$-(2^{n-1}-1)\sim 2^{n-1}-1$までとなる。

符号付き絶対値方式

例）１６ビットで、$-120_{(10)}$を表現した場合

$-120_{(10)} = 1000000001111000_{(2)}$



（負の数を表す１を入れる）

（ウ）負の数の表現（補数を用いる方式）

【補数の定義と２進数の補数表現】

補数とは、とある数に対して加算した場合桁上がりが起きる最小の数のことである。この補数を使うことによって符号を使わず負の数を表現することができるようになる。

補数を使って負の数を表す場合、ある整数の補数がその数の負の数となる。例えば、$+4_{(10)}$という数字がある場合、この＋４の補数Ｘが－４を意味する数値となる。

補数には“基数の補数”と“基数－１の補数”がある。ここでは２進数の“**２の補数**”と“**１の補数**”の２つ補数の求め方と、それを使った計算を説明する。

・１の補数（基数－１の補数）

基数－１の補数は､その桁の数の最大値から減算することによって求めることができる。

２進数では基数－１の補数は、１の補数である。１の補数は減算を使わなくても、各桁のビットを反転（１を０に、０を１に）することで求めることができる。

第１部

１の補数

例）０１１０１０１１$_{(2)}$の１の補数を求める

```
０ １ １ ０ １ ０ １ １
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓        反 転
１ ０ ０ １ ０ １ ０ ０   …（１の補数）
```

・**２の補数（基数の補数）**

基数の補数は、基数－１の補数に１を加算することで求めることができる。

２の補数

例）　０１１０１０１１$_{(2)}$の２の補数を求める

```
０ １ １ ０ １ ０ １ １
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓        反 転
１ ０ ０ １ ０ １ ０ ０   …（１の補数）
                  ＋１   ←　１を加算する
─────────────────
１ ０ ０ １ ０ １ ０ １   …（２の補数）
```

【補数による表現範囲】

２進数で補数を使用すると、負の数は先頭のビットが１になるため、補数を使用した場合の先頭のビットは事実上の符号ビットと同じ扱いになる。

また、１の補数（基数－１の補数）では定義上、０の表現方法が２通り（＋０と－０）でき、演算が複雑になるため、通常は２の補数（基数の補数）を使って負の数を表現する。

従って、８ビットで２の補数を使用して表現できる数値の範囲は、１０進数での－１２８（-2^7）～＋１２７（2^7-1）までとなる。ｎビットある場合の数値の範囲は、次のようになる。

２の補数の範囲

$$-2^{n-1} \sim 2^{n-1}-1$$

【２の補数を使った計算】

補数を使うことで減算を加算で行うことができるようになる。例えば、ａ－ｂという減算を行う場合、ｂの補数を取ることで、ａ+（－ｂ）として加算の形になる。加算で減算が行えるということは、コンピュータが演算を行う機能として加算機能だけあればよく、その分単純な仕組みでよいということになる。

２の補数を使った計算

例１）　０００００１１００$_{(2)}$－０００００１０１$_{(2)}$を求める

```
 12(10) …     00001100
 -5(10) … +   11111011
  7(10) …    100000111
```

（桁上がりは無視する）

０００００１０１
↓
１１１１１０１０　…（１の補数）
↓
１１１１１０１１　…（２の補数）

例２）　０００００１０１$_{(2)}$－００００１１００$_{(2)}$を求める

```
  5(10) …     00000101
-12(10) … +   11110100
 -7(10) …    □11111001
```

００００１１００
↓
１１１１００１１　…（１の補数）
↓
１１１１０１００　…（２の補数）

(c)　小数の表現

ビットでは“０”か“１”の表現しかできないため、“小数点”を表現することができない。よって､ビットで小数を表現するためには､小数点の位置などを考慮した表現が必要となる。少数の表現には“固定小数点数”による表現と、“浮動小数点数”による表現がある。

（ア）固定小数点数

固定小数点数は、小数点の位置が固定的に決められている方式である。使用するビットの長さは１ワードで１６ビット、３２ビット、６４ビットのいずれかとなる。また、負の数は２の補数を使用して表すことが一般的である。

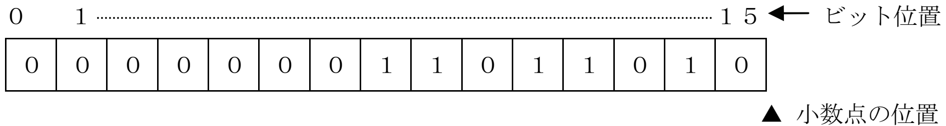
小数点の位置を固定的に決定するが、その位置はコンピュータによって異なる。小数点を最下位桁の次や、最上位桁とその次の桁の間に置く場合が一般的である。

固定小数点数（１）

例）２１８ $_{(10)}$ を１６ビット固定小数点数で表す

２１８ $_{(10)}$ ＝１１０１１０１０ $_{(2)}$

となる。従って、次のように表現できる。



小数点の最下位桁の次に小数点を配置した場合は“固定小数点数”とは呼ぶものの、（ｂ）の（ウ）で説明した２の補数を用いた整数と同じになる。つまり、整数を表現するということは、小数点の最下位桁の次に小数点を配置した固定小数点数による表現である、とも言える。

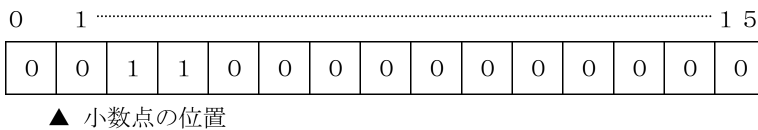
また、小数点を最上位桁とその次の桁の間に置く場合は、次のようになる。

固定小数点数（２）

例）０．３７５ $_{(10)}$ を１６ビット固定小数点数で表す

０．３７５ $_{(10)}$ ＝０．０１１ $_{(2)}$

となる。従って、次のように表現できる。



（イ）浮動小数点数

固定小数点数で表すことのできる範囲はビット数により決まってしまい、使用できるビット数はそのコンピュータで使用するワードによって決まる。そのため、科学技術計算で使用するような非常に大きな数値や、非常に小さな数値を扱うことがワードのビット数の制限によりできなくなる。そこで、小数点を固定しない**浮動小数点数**と呼ばれる方式を使用する。

浮動小数点数では数値を“仮数”と“指数”に分けて表現することで、より広い範囲の数値を取り扱うことができるようになる。

浮動小数点数（１）

浮動小数点数の表現では、数値を次のように表す。

$$X_{(n)} = A \times n^{B}$$

例）５３８．２５$_{(10)}$を浮動小数点数で表現する

$$
\begin{aligned}
538.25_{(10)} &= 538.25 \times 10^{0} \\
&= 53.825 \times 10^{1} \\
&= 5.3825 \times 10^{2} \\
&= 0.53825 \times 10^{3}
\end{aligned}
$$

それぞれ、０．５３８２５ → 仮数、１０ → 基数、３ → 指数と呼ぶ。

０．５３８２５の部分を仮数部、10^{3}の部分を指数部と呼ぶ。

コンピュータでは、仮数を１０進数ではなく１６進数または２進数で表現し、基数は２または１６を使うことが多い。例えば、１６進数の場合の表現は次のようになる。

浮動小数点数（２）

例）２１Ａ．４を１６進数の浮動小数点数で表す

$$
\begin{aligned}
21A.4_{(16)} &= 21A.4 \times 16^{0} \\
&= 21.A4 \times 16^{1} \\
&= 2.1A4 \times 16^{2} \\
&= 0.21A4 \times 16^{3}
\end{aligned}
$$

上図のように、浮動小数点数では小数点が左に１つ移動すると指数が＋１され、逆に小数点が右に移動すると指数が－１される。

また、浮動小数点数をビットで表記するには、エクセス６４形式と、ＩＥＥＥ７５４形式がある。なお、ＩＥＥＥ（電気電子学会：the Institute of Electrical and Electronics Engineers）は通信、電気、電子に関する学会で、様々な規格の標準化を行っている。

この形式で浮動小数点数を表記するには、まず仮数の**正規化**を行う。エクセス６４形式では１６進数で０．ｘｘｘのように１以下の値で、小数点第１位に０以外の数値があるようにする。一方、ＩＥＥＥ７５４形式では、２進数に直した後、１．ｘｘｘのように１以上２未満の、先頭に１がある値にする。

正規化

例）２６．２５ $_{(10)}$ を浮動小数点数にするため正規化する

【エクセス６４】

まず、１６進数に直す。２６．２５ $_{(10)}$ ＝１Ａ．４ $_{(16)}$

１Ａ．４　＝１Ａ．４ × １６^{0}　＝０．１Ａ４ ×１６^{2}

【ＩＥＥＥ７５４】

まず、２進数に直す。２６．２５ $_{(10)}$ ＝１１０１０．０１ $_{(2)}$

１１０１０．０１ × ２^{0}　＝１．１０１００１×２^{4}

浮動小数点数では仮数については正規化を行う。一方、指数は負の数を表現するためバイアスを使用する。エクセス６４、ＩＥＥＥ７５４どちらとも指数は２の補数表示を使わず、特定の値を足して表記する。この特定の値をバイアスと呼ぶ。例えば、エクセス６４では指数に６４を足して表記する。つまり指数が１だった場合は６５、－１だった場合は６３として表記する。

【エクセス６４】

エクセス６４では基数として１６進数を使い、一般的に使用されるエクセス６４は３２ビットで符号ビット１ビット、指数７ビット、仮数２４ビットとなっている。エクセス６４の“６４”は指数に使用するバイアスの値を示す。

エクセス64形式 浮動小数点

０　１ ………… ７　８ ………………………………………… 31

0	1000000	110000000000000　〜　0000
符号	指数（バイアス６４）	仮数

・符号　：　１ビット。正の値のときは０、負の値のときは１である。

・指数　：　７ビット。基数として１６を使用した場合の指数。バイアスとして６４を足した値となる。

・仮数　：　２４ビット。正規化した仮数のうち、０．ｘｘｘｘ$_{(16)}$のｘｘｘｘの部分。

例１）上図の浮動小数点数は

符号…０のため正

指数…１０００００００$_{(2)}$＝６４$_{(10)}$。バイアスとして６４を足しているため０となる

仮数…１１００〜$_{(2)}$＝Ｃ$_{(16)}$。よって、０．Ｃ$_{(16)}$

よって、０．Ｃ$_{(16)}$ × 16^0 ＝ ０．Ｃ$_{(16)}$ ＝<u>０．７５</u>$_{(10)}$

例２）－１２８．５$_{(10)}$を浮動小数点数で表す

－１２８．５$_{(10)}$＝－１００００００００．１$_{(2)}$＝－８０．８$_{(16)}$

＝－０．８０８$_{(16)}$ × 16^2

符号…負のため１

指数…２にバイアスの６４を足して６６。よって、１０００００１０$_{(2)}$

仮数…０．８０８$_{(16)}$の８０８$_{(16)}$＝１０００００００１０００$_{(2)}$

よって、下図のような表記になる。

０　１ ………… ７　８ ………………………………………… 31

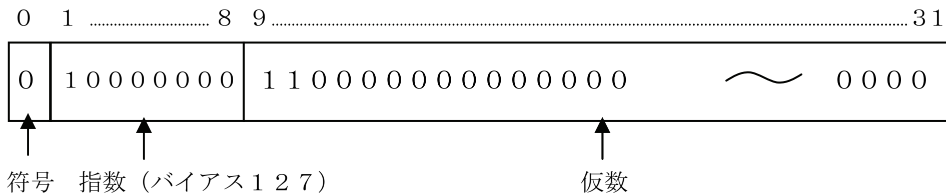
1	1000010	100000001000000　〜　0000

【ＩＥＥＥ７５４】

ＩＥＥＥ７５４での浮動小数点数の形式は、基数として２進数を使う。ビット数は、３２ビット、６４ビット、１２８ビットがある。３２ビットの浮動小数点を**単精度浮動小数点数**、６４ビットを**倍精度浮動小数点数**、１２８ビットを四倍精度浮動小数点数とも呼ぶ。単精度、倍精度、四倍精度では表示できる値の範囲が大幅に異なり、また誤差の範囲も変わる。

ここでは単精度浮動小数点数と、倍精度浮動小数点数を説明する。

ＩＥＥＥ７５４形式 単精度浮動小数点数



・符号　：　１ビット。正の値のときは０、負の値のときは１である。

・指数　：　８ビット。基数として２を使用した場合の指数。バイアスとして１２７を足した値となる。

・仮数　：　２３ビット。正規化した仮数のうち、１．ｘｘｘｘ$_{(2)}$のｘｘｘｘの部分となる。

例）－１３０．５$_{(10)}$を浮動小数点数で表す

－１３０．５$_{(10)}$＝－１０００００１０．１$_{(2)}$

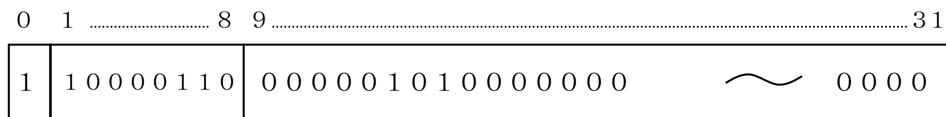
＝－１．０００００１０１$_{(2)}$ × ２^{7}

符号…負のため１

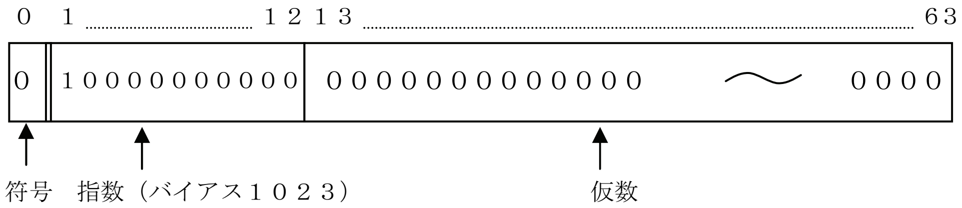
指数…７にバイアスの１２７を足して１３４。よって、１００００１１０$_{(2)}$

仮数…１．０００００１０１$_{(2)}$の小数点以下の部分で０００００１０１$_{(2)}$

よって、下図のような表記になる。



ＩＥＥＥ７５４形式 倍精度浮動小数点数



・符号 ： １ビット。正の値のときは０、負の値のときは１である。

・指数 ： １１ビット。基数として２を使用した場合の指数。バイアスとして１０２３を足した値となる。

・仮数 ： ５２ビット。正規化した仮数のうち、１．ｘｘｘｘ $_{(2)}$ のｘｘｘｘの部分となる。

ただし、小数点をコンピュータで扱う場合には、実際の値と表現された値との間に差異が生じる場合がある。この差異のことを**誤差**と呼ぶ。誤差については、「１．１．３」で説明する。

１．１．３　算術演算と精度

（１）加減乗除の演算

ここでは２進数の加減乗除の演算の方法を説明する。

(a)　加算・減算

２進数の加減算は、加算の場合は「１．１．１（1)」のＮ進数の加算で説明したように、１０進数と同様の加算を行うが､桁の値が２になった場合桁上がりが発生する｡一方減算は、２の補数を使用して減算を加算に直して計算をする。

(b)　乗算・除算

１０進数の乗除算では、基数である１０を掛けるまたは１０で割ると桁が１つ上がる、もしくは下がることになる。これはN進数でも同様で基数であるＮを掛けるまたはNで割ると桁が上下する。

基数を乗算・除算する

【１０進数の場合】

例）３８．２５$_{(10)}$に対して

$$38.25 \times 10 = 38.25 \times 10^{1} = 382.5$$

数字の位置が左へ１つずれ、桁が１つ上がる

$$38.25 \div 10 = 38.25 \times 10^{-1} = 3.825$$

数字の位置が右へ１つずれ、桁が１つ下がる

【２進数の場合】

例）１００１１０．０１$_{(2)}$に対して

$$100110.11 \times 2 = 100110.11 \times 2^{1} = 1001101.1$$

ビットが右へ１つ移動（シフト）する

$$100110.11 \div 2 = 100110.11 \times 2^{-1} = 10011.011$$

ビットが左へ１つ移動（シフト）する

このように基数を乗除する演算は、小数点を中心に考えてみれば、数字の値が右または左に移動している。２進数で考えるとビットが右または左に移動（シフト）することから、この演算を**シフト演算**と呼ぶ。左に移動することを左シフト、右に移動することを右シフトと呼ぶ。

（ア）算術シフト

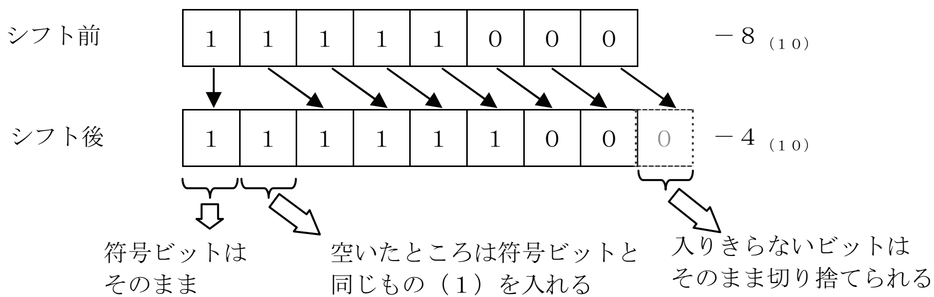
シフト演算のうち、符号ビットや補数を考慮してビットをシフトさせるものを**算術シフト**と呼ぶ。

算術シフト

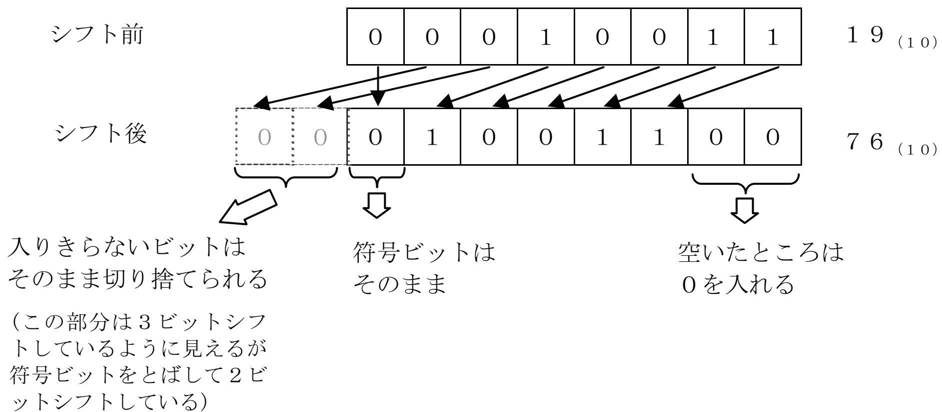
【算術シフトの方法】

・符号ビット以外を右または左に移動させる
・右シフトでは、移動後空いたビットには符号ビットと同じものを入れる
・左シフトでは、移動後空いたビットには０を入れる

例１）１１１１１０００$_{(2)}$を右１ビット算術シフト



例２）０００１００１１$_{(2)}$を左２ビット算術シフト



算術シフトは符号ビットや補数を考慮しているため、乗除の計算に使用する。とある値に２^{N}を乗算する場合に、Ｎが正の数ならば左へＮビット左算術シフト、Ｎが負の数ならばＮビット右算術シフトすることになる。

この算術シフトと、加算を組み合わせれば２進数の乗算が可能になる。

２進数の乗算

例）$110_{(2)} \times 21_{(10)}$ を計算する（$= 6_{(10)} \times 21_{(10)}$）

$110_{(2)} \times 21_{(10)} = 110_{(2)} \times (16+4+1)_{(10)}$

$= 110_{(2)} \times 2^4 + 110_{(2)} \times 2^2 + 110_{(2)} \times 2^0$

$= 1100000_{(2)}$　→ $110_{(2)}$ を左４ビット算術シフト

$+11000_{(2)}$　→ $110_{(2)}$ を左２ビット算術シフト

$+110_{(2)}$

$= 1111110_{(2)} = 126_{(10)}$

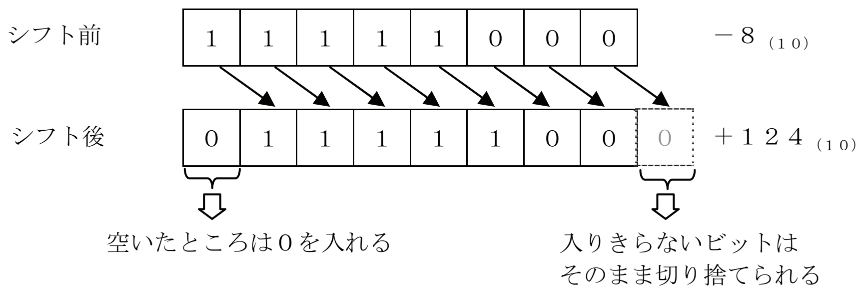
（イ）論理シフト

シフト演算の中で、符号ビットや補数を考慮せず、ただ単にビットを左右に動かす演算を**論理シフト**と呼ぶ。

論理シフト

【論理シフトの方法】
- 符号ビットも含め右または左に移動させる
- 移動後空いたビットには０を入れる

例）$11111000_{(2)}$ を右１ビット論理シフト



空いたところは０を入れる

入りきらないビットは
そのまま切り捨てられる

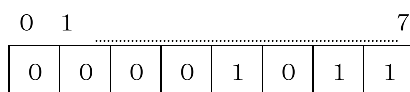
（２）演算の精度

(a)　有効桁数

固定小数点数や浮動小数点数で表現された値と本来の値に誤差ができる場合がある。この時、表現された値のうち、確かに本来の値であると信用できる部分の桁数を**有効桁数**と呼ぶ。

有効桁数

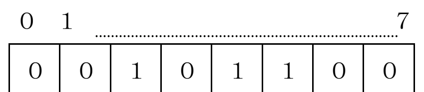
例１）０．０００１０１１１$_{(2)}$を固定小数点数で表現する



▲ 小数点の位置

固定小数点数で表現するため、小数点以下８桁目が切り捨てられて、０．０００１０１１$_{(2)}$となっている。

この値を４倍すると、次のようになる。



▲ 小数点の位置

固定小数点数で表現するために０で埋めている。

この４倍した結果である０．０１０１１００$_{(2)}$と、本来の０．０００１０１１１$_{(2)}$を４倍した値である０．０１０１１１$_{(2)}$とでは０．００００１$_{(2)}$誤差が発生している。

この０．０１０１１００$_{(2)}$の最後の２つの０は固定小数点数で表現するためにつけた０であり、数値として信用できる値ではない。０．０１０１１００$_{(2)}$は１桁目から小数点以下５桁目までの６桁だけが信用できる値となり、有効桁数６桁である。

(b)　精度と誤差

コンピュータでは前述の通り、表現可能な範囲が決まっており、また固定小数点数や浮動小数点数の表現の方式から、本来の数値とコンピュータで保持している数値が異なる場合がある。この２つの数値の差異を誤差と呼ぶ。また、コンピュータが誤差のない状態で保持できる範囲を**精度**と呼び、通常はその表現できる桁数が精度になる。より表現できる桁数が多い表現方式や、より誤差が少なく表現できる表現方式を”精度が高い”と表す。

例えば、ＩＥＥＥ７５４浮動小数点数では、３２ビットの浮動小数点数を“単精度”、その倍のビット数である６４ビット浮動小数点数を“倍精度”と呼び、精度の違いを表している。

また、誤差には“**絶対誤差**”と“**相対誤差**”がある。絶対誤差は、真の値からどれぐらいずれているかを表す。一方、相対誤差は真の値からのずれの割合で、絶対誤差を真の値で割ったものになる。例えば、真の値が１０で、表現した値が９ならば、絶対誤差は１となり、相対誤差は（１０－９）÷１０の０．１（１０％）となる。その誤差には次のものがある。

（ア）丸め誤差

コンピュータで使用できるビット数が有限桁であるため、数値を表現するために切り捨て、切り上げ、四捨五入を行うことによって生じる誤差が**丸め誤差**である。丸め誤差が発生することを“丸める”、“丸められた”と呼ぶことがある。

例えば、１０進数の小数を２進数で表現することによって発生する誤差などが丸め誤差になる。

丸め誤差

例１）０．１$_{(10)}$を８ビット固定小数点数で保存した場合

０．１$_{(10)}$＝０．０００１１００１１００１１……$_{(2)}$（０．０$\dot{0}$０１$\dot{1}$$_{(2)}$）

0	1						7
0	0	0	0	1	1	0	0

▲ 小数点の位置

上図のように、０．０００１１００$_{(2)}$までしか保存できない。

これは、０．０００１１００$_{(2)}$＝ ０．０９３７５$_{(10)}$である。

よって、実際の値の０．１$_{(10)}$と０．００６２５$_{(10)}$の誤差が発生する。

例２）０．１$_{(10)}$を１６ビット固定小数点数で保存した場合

0	1														15
0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0

▲ 小数点の位置

１６ビットでは、

０．０００１１００１１００１１００$_{(2)}$＝０．０９９９７５５８５９３７５$_{(10)}$

となり、誤差は０．００００２４４１４０６２５$_{(10)}$となる。

第１部

（イ）情報落ち

絶対値の非常に大きな数値と非常に小さな数値の演算をする時に、大きな数値の桁に揃えて計算してしまうことになり、小さな数値の方が演算結果に反映されないことによって発生する誤差が**情報落ち**である。

情報落ち

例）０．１０１１１ $_{(2)} \times 2^{15}$と０．１０１１１ $_{(2)} \times 2^{-15}$を加算する

```
  １０１１１００００００００００.
＋                          ０．０００００００００００００００１０１１１
─────────────────────────────────────────────
  １０１１１００００００００００.０００００００００００００００１０１１１
```

１．０１１１０００００００００００００００００００００００００１０１１１ $_{(2)} \times 2^{14}$
となる。

これをＩＥＥＥ７５４単精度浮動小数点数で表すと、

０	１……８	９……３１
０	１０００１１０１	０１１１０００００００００００　〜　００００

である。これは１．０１１１００００００００００００００００００００ $_{(2)} \times 2^{14}$であり、
０．１０１１１ $_{(2)} \times 2^{15}$のことで、加算された０．１０１１１ $_{(2)} \times 2^{-15}$が情報落ちしており、計算結果に反映されていないことになる。

情報落ちを少なくするためには、複数の値を加算する時に絶対値の小さい数値の方から順に加算していくなどの方法が必要となる。

（ウ）桁落ち

桁落ちとは絶対値のほぼ等しい２つの数値を演算すると、有効桁数が急激に減ることがあるために発生する誤差のことである。

桁落ち

例）有効桁数が８桁の２つの数値の０．１１１１０１１ $_{(2)}$ －０．１１１１０１０ $_{(2)}$ を計算する

```
  ０．１１１１０１１
－０．１１１１０１０
――――――――――
  ０．００００００１
```

この演算結果を、浮動小数点数とするため正規化すると、１．０ $_{(2)}$ ×２ $^{-7}$ となり、有効桁数は１桁となってしまう。

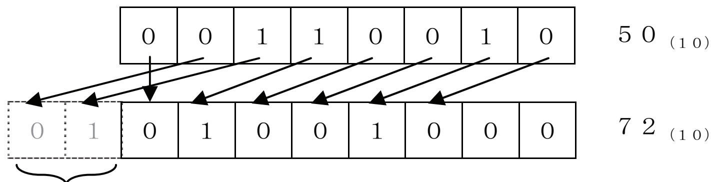
第1部

（エ）オーバフロー／アンダフロー

コンピュータでは使用するビット数により表現できる数値の範囲は決まっている。そのため、演算の結果、この数値の範囲を超えてしまうことによって発生する誤差をフロー（あふれ）と呼ぶ。表現の最大値を超えてしまうことを**オーバフロー**、表現の最小値を下回ってしまうことを**アンダフロー**と呼ぶ。

オーバフロー

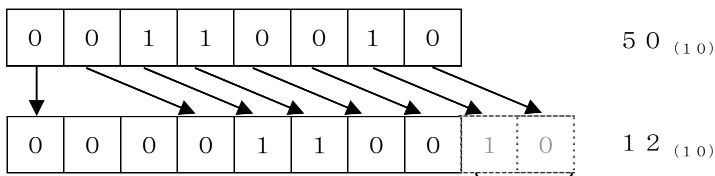
例）２の補数表示の８ビット表示で、００１１００１０$_{(2)}$＝５０$_{(10)}$を４倍する。
（左２ビット算術シフト）



この部分が入りきらずフローしている

アンダフロー

例）２の補数表示の８ビット表示で、００１１００１０$_{(2)}$＝５０$_{(10)}$を４で割る。
（右２ビット算術シフト）



この部分が入りきらずフローしている

１．１．４　集合

（1）集合

集合とは“ものの集まり”のことで、例えば“身長が１７０㎝以上の人の集まり”のように、その集合にいる・いないが明確に判別できるような集まりのことである。この集合の中に含まれるものを**要素**と呼ぶ。

(a)　集合の表現

集合と要素は次のように表現する。通常は集合をＡ、Ｂ、Ｃ……、要素をａ、ｂ、ｃ……で表すことが多く、また、要素のない集合を空集合と呼び、φ（ファイ）という記号で表す。

集合の表現

集合　＝｛要素，要素，要素，……｝

・要素ａが集合Ａの要素である　… $a \in A$

・集合Ａに要素ａが含まれている　… $A \ni a$

・空集合　… ϕ

(b)　部分集合・全体集合・補集合

（ア）部分集合

２つの集合ＡとＢがあり、Ａの要素のすべてがＢの要素に含まれているとき、ＡはＢの**部分集合**であると表現する。

また、集合の関係を図示するために使用されるのが**ベン図**（ベン・オイラー図）である。ベン図では集合を円で表す。

第1部

部分集合

集合Ａ＝｛ａ，　ｂ，　ｃ，　ｄ｝

集合Ｂ＝｛ａ，　ｂ，　ｃ，　ｄ，　ｅ，　ｆ｝

この場合、Ａの要素はＢにすべて存在するため、ＡはＢの部分集合である。部分集合は次のように表す。

・集合Ａは集合Ｂの部分集合である　…　Ａ⊆Ｂ

集合Ｃ＝｛ａ，　ｂ，　ｃ，　ｄ｝

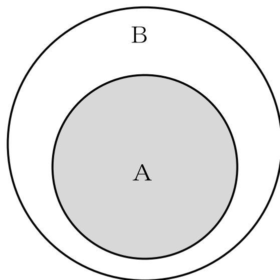
集合Ｃは集合Ａと全く同じ集合である。この場合、“集合Ａの要素すべてが集合Ｃに含まれている”という部分集合の条件を満たしてもいるので、Ａ⊆Ｃでもあり、Ａ＝Ｃでもある。

さらに、上の集合Ｂは集合Ａにない要素を含んでいる。この場合、集合ＡはＢの真部分集合であるとも言う。つまり、Ａ≠ＢかつＡ⊆Ｂの状態が真部分集合である。

・集合Ａは集合Ｂの真部分集合である …Ａ⊂Ｂ

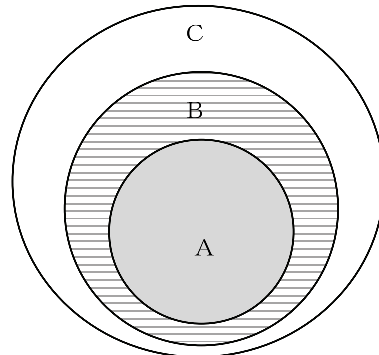
ベン図

Ａ⊂Ｂ



※グレーの部分が集合Ａ

Ａ⊂Ｂ、Ｂ⊂Ｃ



※グレーの部分が集合Ａ
横線の部分とグレーの部分が集合Ｂ

（イ）全体集合・補集合

複数の要素を持つ集合があるとして、その集合の部分集合を作ることを考える時、その元の集合のことを**全体集合**と呼ぶ。

全体集合からとある部分集合を抜き出した場合、その部分集合に含まれない集合を**補集合**と呼び、“￣”を使って表現する。

全体集合・補集合

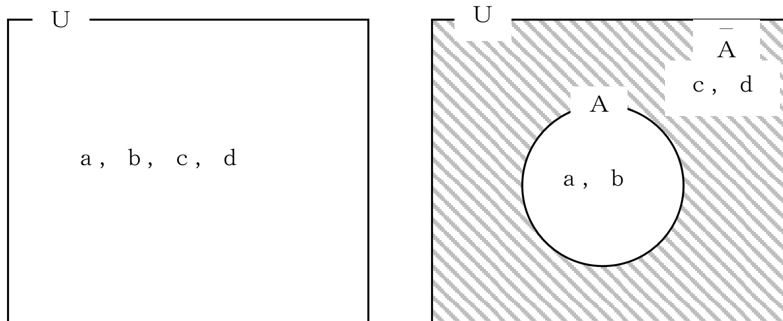
集合Ｕ＝｛ａ，ｂ，ｃ，ｄ｝がある時、この集合Ｕの部分集合を作ると、

｛ａ，ｂ，ｃ，ｄ｝、｛ａ，ｂ，ｃ｝、｛ａ，ｂ，ｄ｝、｛ａ，ｃ，ｄ｝、｛ｂ，ｃ，ｄ｝、｛ａ，ｂ｝、｛ａ，ｃ｝、｛ａ，ｄ｝、｛ｂ，ｃ｝、｛ｂ，ｄ｝、｛ｃ，ｄ｝、｛ａ｝、｛ｂ｝、｛ｃ｝、｛ｄ｝、φ がある。

これらの部分集合に対して、集合Ｕは全体集合である。

全体集合Ｕの中から、部分集合Ａ＝｛ａ，ｂ｝を考えると、全体集合Ｕのうち部分集合Ａに含まれない集合｛ｃ，ｄ｝をＡに対する補集合と呼び、$\overline{A}$と表現する。

ベン図で表現すると次のようになる。



(c)　集合の演算

複数の値に対し、加減乗除などの演算を行うと結果として１つの値が求まる。集合でも複数の集合に対して、演算を行いその結果として１つの集合を求めることができる。

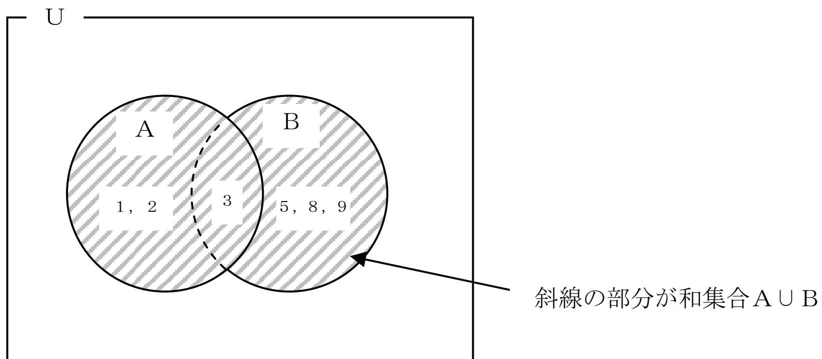
（ア）和集合

全体集合Ｕの部分集合である、２つの集合ＡとＢがある時、ＡとＢのどちらか、またはＡとＢの両方に属するすべての要素の集合を**和集合**と呼ぶ。和集合は記号“∪”を使って表現する。例えば、ＡとＢの和集合は“Ａ∪Ｂ”と記述される。

和集合

例）集合Ａ＝｛１，２，３｝、集合Ｂ＝｛３，５，８，９｝がある

和集合Ａ∪Ｂは、Ａ∪Ｂ＝｛１，２，３，５，８，９｝となる。
ベン図



（イ）積集合

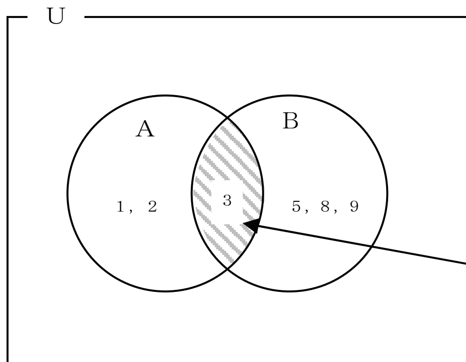
全体集合Ｕの部分集合である、２つの集合ＡとＢがある時、ＡとＢの両方に属するすべての要素の集合を**積集合**と呼ぶ。積集合は記号“∩”を使って表現する。例えば、ＡとＢの積集合は“Ａ∩Ｂ”と記述される。

積集合

例）集合Ａ＝｛１，２，３｝、集合Ｂ＝｛３，５，８，９｝がある

積集合Ａ∩Ｂは、Ａ∩Ｂ＝｛３｝となる。

ベン図



斜線の部分が積集合Ａ∩Ｂ

（ウ）差集合

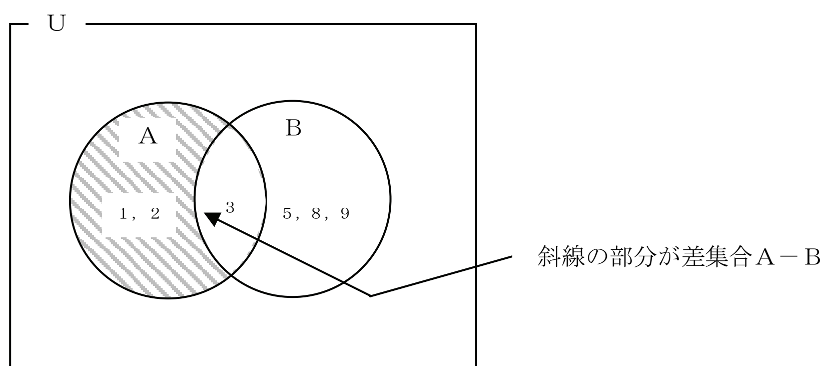
全体集合Ｕの部分集合である、２つの集合ＡとＢがある時、ＡからＡとＢの両方に属する要素を除いた集合を**差集合**と呼ぶ。差集合は記号“－”を使って表現する。例えば、ＡとＢの差集合は“Ａ－Ｂ”と記述される。

差集合

例）集合Ａ＝｛１，２，３｝、集合Ｂ＝｛３，５，８，９｝がある

差集合Ａ－Ｂは、Ａ－Ｂ＝｛１，２｝となる。

ベン図



（２）命題論理

(a)　命題とは

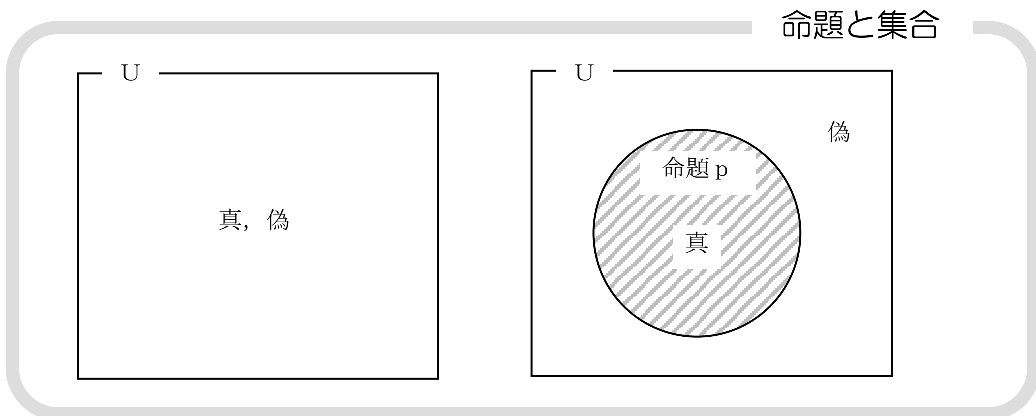
集合と密接な関係にあるものが**命題**である。命題は何かしらの対象の事柄について述べたもので、その述べたことが“**真**”であるか、“**偽**”であるかの判定を行うことである。その命題が“真”であるということはその命題が正しいということで、“偽”であるならばその命題が正しくないということになる。命題は真または偽以外にはなりえない。命題は一般的にｐ、ｑ、ｒなどの文字で表現される。

例えば、“人間は動物である”という命題があるとすると、この命題は正しいので“真”である、ということが言える。

(b)　命題論理

命題は真または偽のどちらかになる。つまり、命題は真と偽の値を持つ集合のうち、真または偽の値を持つ部分集合である、とも言える。

全体集合Ｕ＝｛真，偽｝があるとして、真である命題ｐがある時、命題ｐは全体集合Ｕの中の部分集合ｐ＝｛真｝ということになる。よって、集合と同様にベン図で表現することができる。



集合では集合同士の演算ができたように、命題同士での演算もできる。これを命題論理と呼ぶ。命題論理を使うことで、命題同士の問題である複合命題を推論できるようになる。

（ア）否定（ＮＯＴ）

命題の内容を反対にするもので、“￣”で表現する。命題ｐが真ならば命題$\bar{p}$は偽になり、命題ｐが偽ならば命題$\bar{p}$は真となる。集合で言うならば$\bar{p}$はｐの補集合のことである。

命題ｐが“人間は動物である”で真となり、命題$\bar{p}$は“人間は動物でない”となり偽になる。

（イ）連言（ＡＮＤ）

２つの命題を「かつ」でつなぐもので、“∧”で表す。ｐとｑの命題がともに真のときのみ、ｐ∧ｑが真となり、どちらか一方または両方が偽ならばｐ∧ｑは偽になる。集合で言うならばｐ∩ｑの積集合のことである。

（ウ）選言（ＯＲ）

２つの命題を「または」でつなぐもので、“∨”で表す。ｐとｑの命題が少なくともどちらか一方が真のときｐ∨ｑが真となり、両方が偽ならばｐ∨ｑは偽になる。集合で言うならばｐ∪ｑの和集合のことである。

１．１．５　論理演算

（１）真理値と論理演算

論理演算とは“真”または“偽”の値だけを持つ集合での演算のことである。論理演算は**ブール代数**とも呼ばれる。論理演算でとりうる値である真と偽は、**真理値**（または**ブール値**：boolean-value）と呼ばれ真を１、偽を０で表すこともできる。集合での演算や、複合命題での命題論理も論理演算を使うことができる。

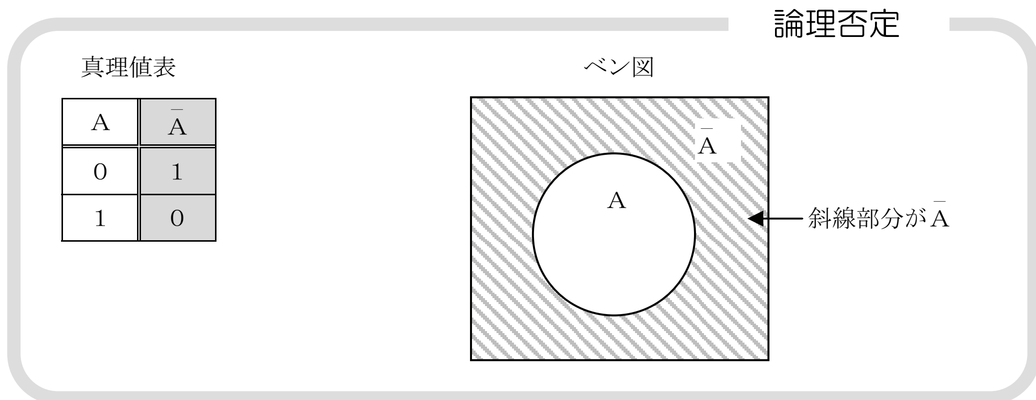
論理演算ではベン図または**真理値表**と呼ばれる表を用いる。真理値表は論理演算で取りうる真と偽のパターンを網羅した表である。

（２）論理演算

論理演算では論理否定、論理和、論理積の３つの演算が存在する。またこの３つを使った否定論理和、否定論理積、排他的論理和なども存在する。

(a)　論理否定（ＮＯＴ）

論理否定はとある集合Ａに対して行われる演算で、“￣”、“¬”、“c”などを演算子として使い、$\overline{A}$、¬Ａ、A^cと表現する。Ａが真の時$\overline{A}$は偽、Ａが偽の時$\overline{A}$は真となる。



A	$\overline{A}$
0	1
1	0

(b)　論理和（ＯＲ）

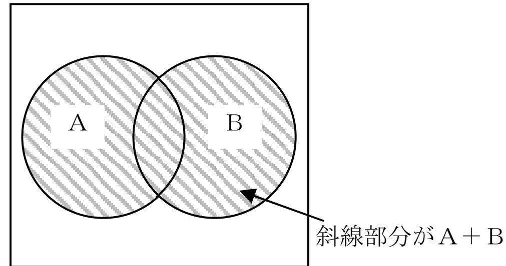
論理和は、とある集合ＡとＢに対して行われる演算で、“＋”、“∪”、“∨” などを演算子として使い、Ａ＋Ｂ、Ａ∪Ｂ、Ａ∨Ｂと表現する。ＡまたはＢの少なくとも一方が真の時Ａ＋Ｂは真となり、ＡとＢの両方が偽の時Ａ＋Ｂは偽となる。

論理和

真理値表

A	B	A＋B
0	0	0
0	1	1
1	0	1
1	1	1

ベン図



(c)　論理積（ＡＮＤ）

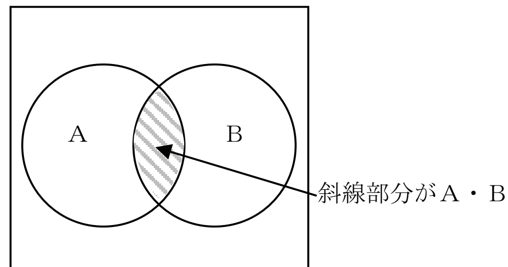
論理積は、とある集合ＡとＢに対して行われる演算で、“・”、“∩”、“∧” などを演算子として使い、Ａ×Ｂ、Ａ∩Ｂ、Ａ∧Ｂと表現する。ＡまたはＢの両方が真の時はＡ・Ｂは真となり、ＡとＢの少なくともどちらか一方が偽の時Ａ・Ｂは偽となる。

論理積

真理値表

A	B	A・B
0	0	0
0	1	0
1	0	0
1	1	1

ベン図



(d)　否定論理和（ＮＯＲ）

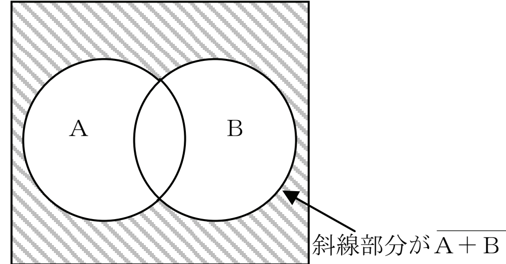
否定論理和は、とある集合ＡとＢに対して行われる演算で、ＡとＢの論理和の論理否定になる。$\overline{A+B}$、￢（Ａ＋Ｂ）のように表現する。

否定論理和

真理値表

A	B	A＋B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

ベン図



(e)　否定論理積（ＮＡＮＤ）

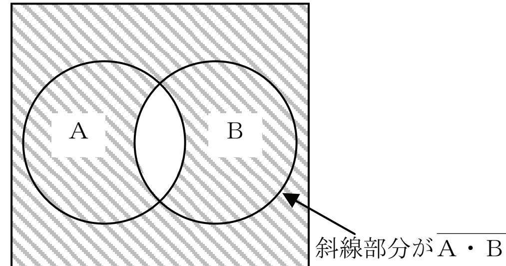
否定論理積は、とある集合ＡとＢに対して行われる演算で、ＡとＢの論理積の論理否定になる。$\overline{A \cdot B}$、￢（Ａ・Ｂ）のように表現する。

否定論理積

真理値表

A	B	A・B	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

ベン図



否定論理和、否定論理積のどちらも、論理和と論理積の論理否定であり、“ＡとＢそれぞれの論理否定”の論理和、論理積ではないことに注意する。

(f)　排他的論理和（Exclusive OR：ＥＯＲまたはＸＯＲ）

排他的論理和は、とある集合ＡとＢに対して行われる演算で、“⊕”を演算子として使い、Ａ⊕Ｂと表現する。ＡとＢのどちらか一方のみが真の時は真、ＡとＢの双方の値が一致した時は偽となる。論理和と論理積と論理否定で表現すると、Ａ⊕Ｂ＝（$\overline{A}$・Ｂ）＋（Ａ・$\overline{B}$）などになる。

排他的論理和

真理値表

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

ベン図

A
B
斜線部分が
A⊕B

A	B	$\overline{A}$	$\overline{A}$・B	A	$\overline{B}$	A・$\overline{B}$	（$\overline{A}$・B）＋（A・$\overline{B}$）
0	0	1	0	0	1	0	0
0	1	1	1	0	0	0	1
1	0	0	0	1	1	1	1
1	1	0	0	1	0	0	0

※ＥＯＲは（$\overline{A}$・Ｂ）＋（Ａ・$\overline{B}$）以外でも表現できる

（３）論理演算の法則

(a)　演算の順序

複数の論理演算子を組み合わせて演算する場合、加減乗除の四則演算と同じように演算に優先順位が決まっている。優先順位は次の順番となる。

１）論理否定
２）論理積
３）論理和、排他的論理和

例えば、$\overline{A}$・Ｂ＋Ｃという演算があった場合、まず$\overline{A}$の論理否定を求め、その結果とＢの論理積の演算を行った後、その結果とＣの論理和をとることになる。演算の順序を変えたい場合や明記したい場合は、四則演算と同様にカッコを使い、カッコ内の演算を優先させる。

(b)　演算の法則

（ア）基本的な法則

論理演算では次のような基本的な法則がある。

論理演算の法則

Ａ・１＝Ａ

Ａ	１	Ａ・１
０	１	０
１	１	１

Ａ＋１＝１

Ａ	１	Ａ＋１
０	１	１
１	１	１

Ａ・０＝０

Ａ	０	Ａ・０
０	０	０
１	０	０

Ａ＋０＝Ａ

Ａ	０	Ａ＋０
０	０	０
１	０	１

【排中則】

$A \cdot \overline{A} = 0$

Ａ	$\overline{A}$	$A \cdot \overline{A}$
０	１	０
１	０	０

$A + \overline{A} = 1$

Ａ	$\overline{A}$	$A + \overline{A}$
０	１	１
１	０	１

【復元則】

$\overline{\overline{A}} = A$

Ａ	$\overline{A}$	$\overline{\overline{A}}$
０	１	０
１	０	１

【べき等則】

Ａ・Ａ＝Ａ

Ａ	Ａ	Ａ・Ａ
０	０	０
１	１	１

Ａ＋Ａ＝Ａ

Ａ	Ａ	Ａ＋Ａ
０	０	０
１	１	１

【交換則】

Ａ・Ｂ＝Ｂ・Ａ

Ａ	Ｂ	Ａ・Ｂ
０	０	０
０	１	０
１	０	０
１	１	１

Ｂ	Ａ	Ｂ・Ａ
０	０	０
１	０	０
０	１	０
１	１	１

Ａ＋Ｂ＝Ｂ＋Ａ

Ａ	Ｂ	Ａ＋Ｂ
０	０	０
０	１	１
１	０	１
１	１	１

Ｂ	Ａ	Ｂ＋Ａ
０	０	０
１	０	１
０	１	１
１	１	１

（イ）吸収則

ＡとＢの論理和を求めたものにＡの論理積を求めると、Ａになる。論理和と論理積を入れ替えて、ＡとＢの論理積を求めたものにＡの論理和を求めてもＡになる。

吸収則

（Ａ＋Ｂ）・Ａ＝Ａ

Ａ	Ｂ	Ａ＋Ｂ	Ａ	（Ａ＋Ｂ）・Ａ
０	０	０	０	０
０	１	１	０	０
１	０	１	１	１
１	１	１	１	１

（Ａ・Ｂ）＋Ａ＝Ａ

Ａ	Ｂ	Ａ・Ｂ	Ａ	（Ａ・Ｂ）＋Ａ
０	０	０	０	０
０	１	０	０	０
１	０	０	１	１
１	１	１	１	１

（ウ）結合則

Ａ、Ｂ、Ｃの論理積を行う場合、ＡとＢの論理積の結果にＣとの論理積を行うのと、ＢとＣの論理積の結果にＡとの論理積を行うのは同じ結果となる。これは論理和でも同じである。

結合則１

（Ａ・Ｂ）・Ｃ＝Ａ・（Ｂ・Ｃ）

A	B	A・B	C	（A・B）・C
0	0	0	0	0
0	0	0	1	0
0	1	0	0	0
0	1	0	1	0
1	0	0	0	0
1	0	0	1	0
1	1	1	0	0
1	1	1	1	1

B	C	B・C	A	A・（B・C）
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	1	0	0
0	0	0	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1

（Ａ＋Ｂ）＋Ｃ＝Ａ＋（Ｂ＋Ｃ）

A	B	A＋B	C	（A＋B）＋C
0	0	0	0	0
0	0	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	1	0	1
1	0	1	1	1
1	1	1	0	1
1	1	1	1	1

B	C	B＋C	A	A＋（B＋C）
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1
0	0	0	1	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

（エ）分配則

ＡとＢの論理積の結果にＣとの論理和をとるのと、ＡとＣ、ＢとＣの論理和の結果同士に論理積をとるのは同じ結果となる。論理積と論理和を入れ替えても同じである。

分配則

（Ａ・Ｂ）＋Ｃ＝（Ａ＋Ｃ）・（Ｂ＋Ｃ）

A	B	A・B	C	（A・B）＋C
0	0	0	0	0
0	0	0	1	1
0	1	0	0	0
0	1	0	1	1
1	0	0	0	0
1	0	0	1	1
1	1	1	0	1
1	1	1	1	1

A	C	A＋C	B	C	B＋C	（A＋C）・（B＋C）
0	0	0	0	0	0	0
0	1	1	0	1	1	1
0	0	0	1	0	1	0
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	1	1	0	1	1	1
1	0	1	1	0	1	1
1	1	1	1	1	1	1

（Ａ＋Ｂ）・Ｃ＝（Ａ・Ｃ）＋（Ｂ・Ｃ）

A	B	A+B	C	（A+B）・C
0	0	0	0	0
0	0	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	1	0	0
1	0	1	1	1
1	1	1	0	0
1	1	1	1	1

A	C	A・C	B	C	B・C	（A・C）＋（B・C）
0	0	0	0	0	0	0
0	1	0	0	1	0	0
0	0	0	1	0	0	0
0	1	0	1	1	1	1
1	0	0	0	0	0	0
1	1	1	0	1	0	1
1	0	0	1	0	0	0
1	1	1	1	1	1	1

（オ）ド・モルガンの法則

ド・モルガンの法則はＡとＢの論理積の論理否定（否定論理積）は、ＡとＢそれぞれの論理否定の論理和になる、またはＡとＢの論理和の論理否定（否定論理和）は、ＡとＢのそれぞれの論理否定の論理積になるという法則のことである。

ド・モルガンの法則

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

A	B	A・B	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

A	B	A＋B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0