基本情報技術者

試験対策テキストⅠ【ベーステクノロジ編】

無料体験入学者用

Ver.4.2

TAC

はじめに

基本情報技術者試験は，2009年春の情報処理技術者試験の制度改革により，従来のシステム開発技術者を対象とした試験から，システム開発技術者・利用者を問わずITにかかわるすべての人材を対象とした試験に衣替えしました。これに伴い，試験で問われる知識範囲も整理・拡充され，現代のIT社会に必要な幅広い知識を問うようになっています。

本書は基本情報技術者試験の出題範囲として体系化された，テクノロジ系，マネジメント系，ストラテジ系の3系統のうち，テクノロジ系の基礎となる情報の基礎理論やハードウェア，ソフトウェア等に関する分野の知識を網羅しています。そして，IT分野について初心者の方でも無理なく学習が行えるよう，基礎的な用語や考え方を分かりやすく解説するように心がけました。

本書により，読者のみなさんが基本情報技術者試験に合格されることを願ってやみません。

2018年1月
TAC情報処理講座

本書の利用法

本書は，各テーマのタイトル欄の右下に重要度が表記されています。重要度は★～★★★の3段階で設定されており、★★★がもっとも重要度が高い内容です。学習する際の目安としてください。特に、重要度の高いテーマについてはしっかりと基礎を理解するよう努めましょう。

また、各テーマのタイトルの下に、そのテーマの学習内容を記載していますので、それを踏まえて各項目の学習に当たりましょう。なお、重要な知識項目は青字で表記されています。しっかりと理解していくことが大切です。

それでは学習を進めてまいりましょう！

目　次

Part 1

基礎理論

1-1 2進数と基数変換

重要度 ★★★

人間の世界では，数を10進数で表します。これは，1桁(けた)に0～9までの10種類の数を用いる方法です。9の次は桁が上がって10となります。
コンピュータの世界では2進数を使います。ここでは2進数の性質や10進数との変換を説明します。

2進数とは

コンピュータの世界では，数は0と1しかありません。つまり，"2"種類の数(0と1)を用いてさまざまな数値を表さなければならないのです。このような方法を「2を基数とする数値」または単に**2進数**といい表します。

2進数では"0"，"1"，と数えていき，その次ですぐ上の桁へ繰上りが起きて"10"となります。この"10"が10進数の2に相当します。"10"の次は"11"，その次はさらに上の桁へ繰上りが起きて"100"となります。"11"は10進数の3に，"100"は10進数の4に該当する表現です。

10進数
1桁に0～9までの
10種類の数字を用いる

2進数
1桁に0，1の
2種類の数字を用いる

10進数	2進数
0	0
1	1
2	10
3	11
4	100

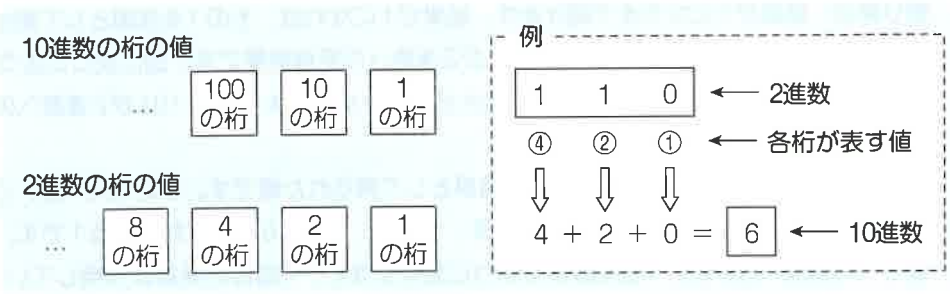
繰上り

2進数であることを明示する表記法はいくつかありますが，本書では主に，$(100)_2$のように基数を右下に付した形を用います。

2進数から10進数への変換

「2進数で表された数を10進数に変換する」というような，基数を変えて数値を表現しなおすことを，**基数変換**とよびます。

2進数から10進数への基数変換は，2進数の各桁が表す値をもとに簡単に変換できます。



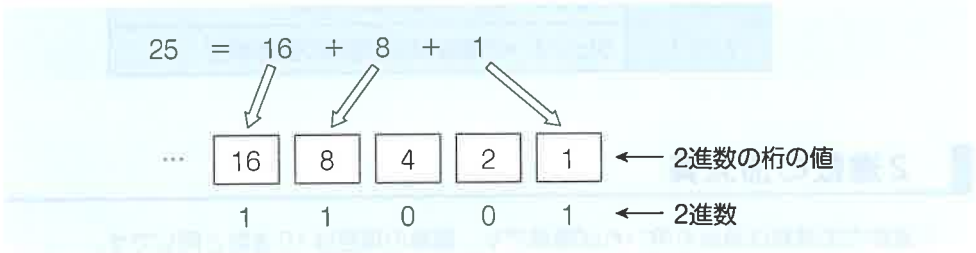
10進数の桁の値は，1(＝10^0)の桁，10(＝10^1)の桁，100(＝10^2)の桁，…というように，各桁の表している値の重みが10倍ずつ変化していきます。これに対し，2進数は1(＝2^0)の桁，2(＝2^1)の桁，4(＝2^2)の桁，…と2倍ずつ変化します。たとえば2進数“110”は，上位から「4の桁が1，2の桁が1，1の桁が0」という数です。つまり，10進数では，

4 + 2 + 0 = 6　← 4の桁と2の桁は「ある」，1の桁は「ない」

と表されるのです。

10進数から2進数への変換

10進数から2進数への変換は，各桁の重みを考えて，もとの数を8や16などの「2のべき乗」に分解するとうまくいきます。



また，もとの数を2で除算(割り算)することを繰り返し，余りを並べていくという方法でも変換結果を得ることができます。

割り算は，結果が1になるまで続けます。結果が1になれば，その1を先頭として剰余を「下から上」に拾いながら並べます。その結果が2進数への変換結果です。図に表したように，10進数29は除算結果の1を先頭に1101と剰余が並びます。つまり，11101が2進数への変換結果となるのです。

最後の除算結果1は，2で4回除算した結果として得られた値です。逆に考えれば，この1は単なる1ではなく，2^4を背負っているのです。つまり，$2^4=16$の桁に対応する1です。

最も下の剰余1は，2で4回除算したように見えますが，4回目の除算は作用していません。なぜならば，4回目の除算ではじき出された余りだからです。この1には2^3が作用しています。つまり8の桁です。

同様に次の1は$2^2=4$の桁，次の0は$2^1=2$の桁，最後の1は$2^0=1$の桁に対応します。

情報の表現

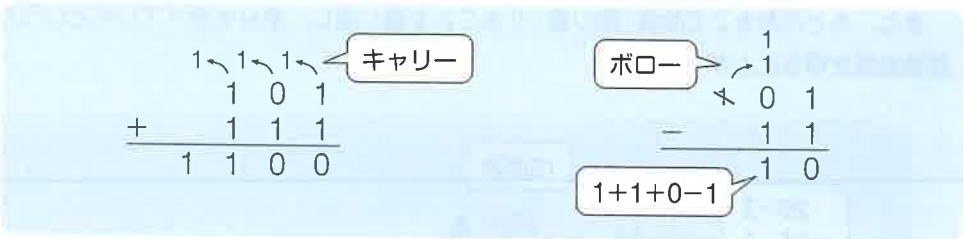
これ以降は2進数の桁とともに，**ビット(bit)**という言葉も用います。ビットはコンピュータの扱う情報の最小単位で「2進数の1桁」を意味します。

1ビットは情報の最小単位ですが，情報の基本単位としては主に8ビットが用いられます。これを**バイト(byte)**とよびます。256バイトの情報は，256×8＝2,048ビットの情報と同じ意味です。

ビット	2進数の1桁。情報の最小単位
バイト	8ビット＝2進数8桁。情報の基本単位

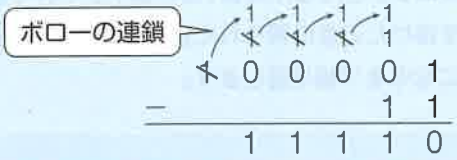
2進数の加減算

2進数の加減算は筆算を用いれば簡単です。筆算の理屈は10進数と同じです。



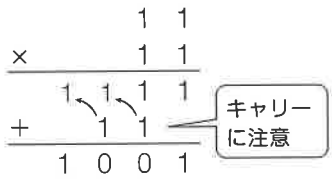
2進数の加算はある意味で10進数より簡単です。ある桁の計算が2以上となる場合には，上位桁への**キャリー**(桁上げ)を行います。下位桁からキャリーがある場合には，それを忘れず計算してください。

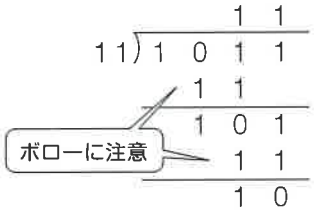
減算の場合には，上位桁から値を借りる**ボロー**(桁借り)が起こります。一つ上位の桁からボローする場合には，上位桁から1を減じ，下位桁に1を二つ加えます。遠くの桁から値を借りる場合には，ボローが連鎖します。



参考：2進数の乗除算

乗除算も，加減算と同じく筆算を用います。





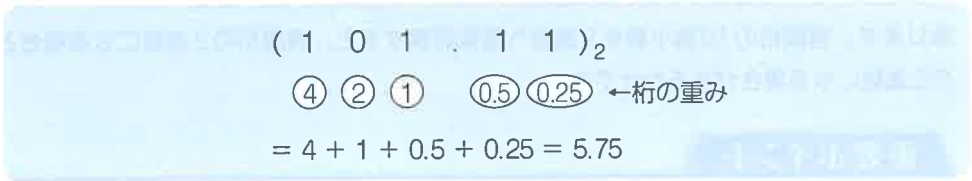
$(11)_2 \times (11)_2 = (1001)_2$　　$(1011)_2 \div (11)_2 = (11)_2 \cdots (10)_2$

乗算の筆算は加算の応用(キャリーに注意)，除算は減算の応用(ボローに注意)です。

2進小数

2進数でも10進数と同様に，小数表現が可能です。整数部分では桁が一つ上がるごとに重みが2倍になりましたが，小数部分は逆に，桁が一つ下がるごとに，重みが1/2倍になっていきます。小数点から一つ下がった桁は1/2＝0.5の桁，さらに下がれば$1/2^2 = 1/4 = 0.25$の桁になるわけです。

桁の重みを考えれば，小数を含む2進数を10進数に簡単に変換できます。



10進小数から2進小数への変換

$(0.1)_2$は$(0.5)_{10}$，$(0.01)_2$は$(0.25)_{10}$，… といった各桁の重みが理解できていれば，いくつかの10進小数については，簡単に2進小数へ変換できます。たとえば10進数の0.625であれば，

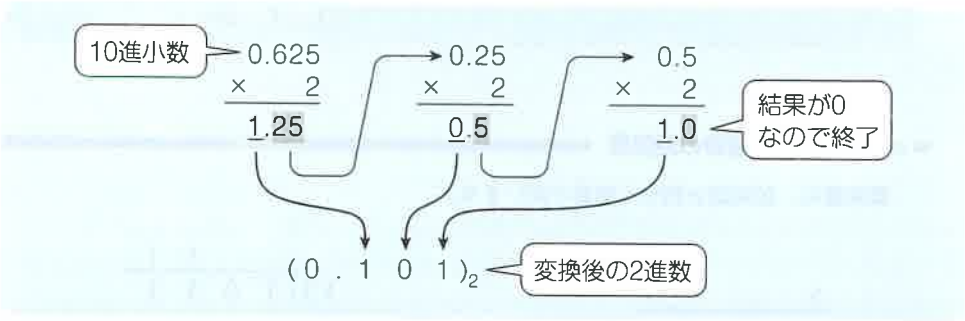
① 0.625　は，0.5＋0.125　に分解できる

② $(0.5)_{10} = (0.1)_2$，$(0.125)_{10} = (0.001)_2$

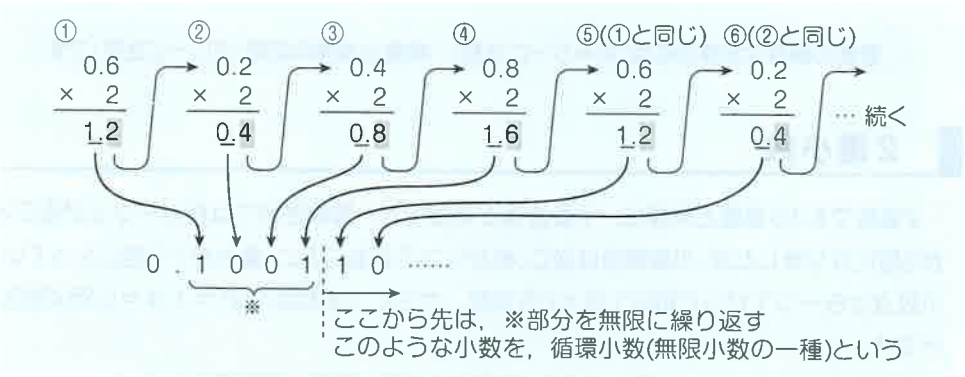
③ よって，$(0.1)_2 + (0.001)_2 = (0.101)_2$

のようになります。

また，元の10進小数に2をどんどん乗算（掛け算）していくという方法でも，変換結果を得ることができます。2を掛けたときに得られた整数部の値が，2進小数の各桁に対応します。この乗算を，小数部が0になるまで繰り返します。



ただし，次に示すように10進小数の0.6を2進小数に変換しようとすると，乗算が終了せずに永遠に続いてしまいます。これは，0.6という有限の桁数の10進数を2進数に変換すると無限の桁（無限小数）になってしまうことを表しています。



このような「有限桁の2進小数に変換できない10進数の小数」には，他に0.1や0.05などがあります。有限桁の10進小数を2進数へ基数変換すると，有限桁の2進数になる場合と無限桁の2進数になる場合があるわけです。

重要ポイント

【2進数】
- **2になったら繰り上がる**
- **1桁上がるごとに，重みが2倍になる（1，2，4…）**
- **1桁下がるごとに，重みが1/2倍になる（0.5，0.25…）**

【ビットとバイト】
- **1ビット＝2進数の1けた分**
- **1バイト＝8ビット**

1-2　8進数と16進数

重要度　★★★

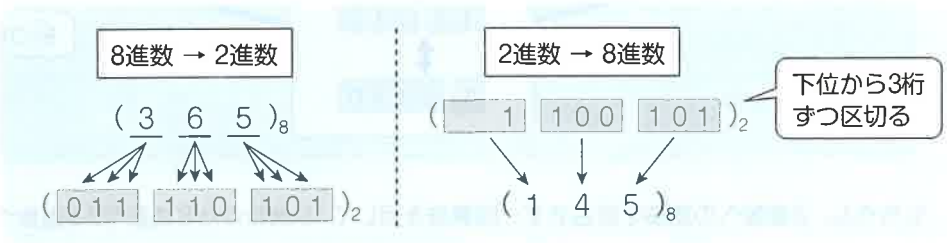
2進数の欠点は桁が多くなりすぎることです。そこで2進数と相性がよく，より桁が少なくなる表現方法が用いられることがあります。その代表例が8進数や16進数です。

なお，これ以降，ある数nの基数kを明示する場合には，$(n)_k$と表現します。

$(n)_2$ … nは2進数　　$(n)_8$ … nは8進数

8進数

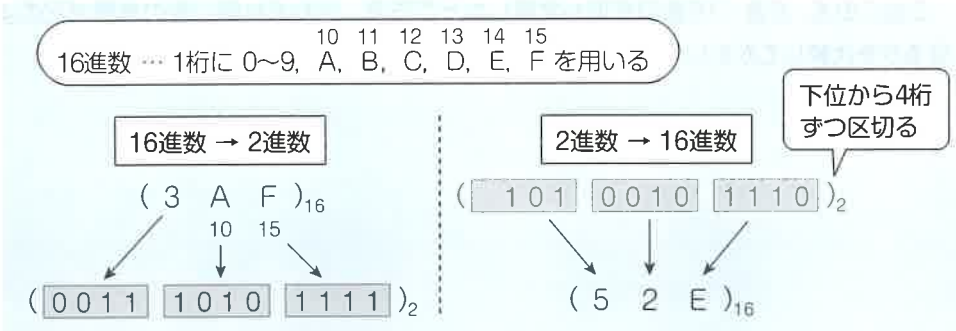
1桁に0～7を用いる方法です。8進数の1桁は2進数の3ビットに対応するため，2進数との変換を簡単に行うことができます。



16進数

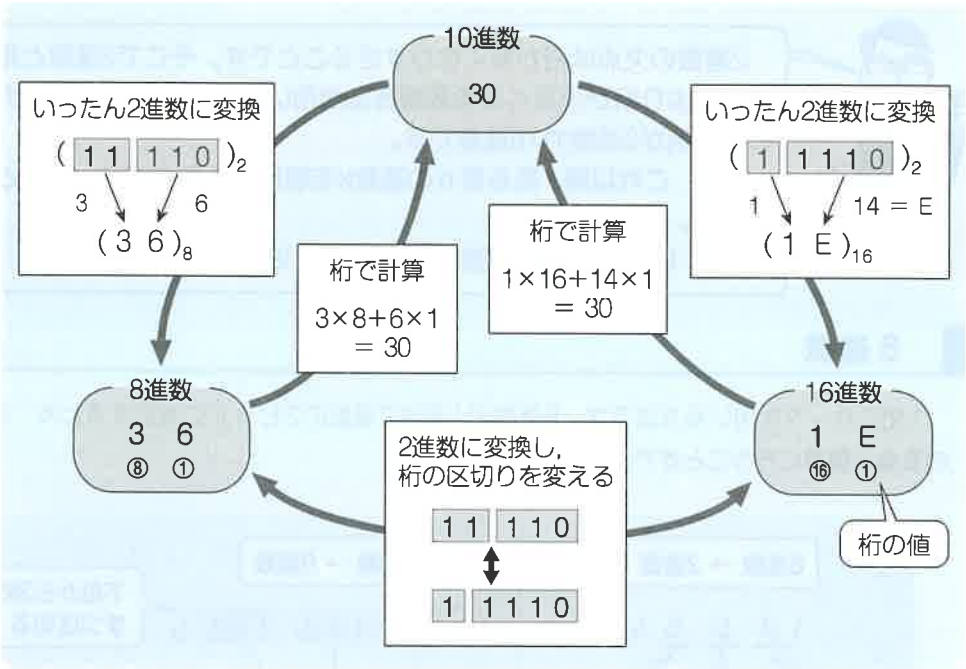
1桁に0～15を用いる方法です。10～15を1桁で表すため，アルファベットのA～Fを用います。

16進数の1桁は2進数の4ビットに対応するため，8進数と同様に2進数へ簡単に変換できます。また，情報の基本単位である1バイトがちょうど2桁の16進数で表されます。

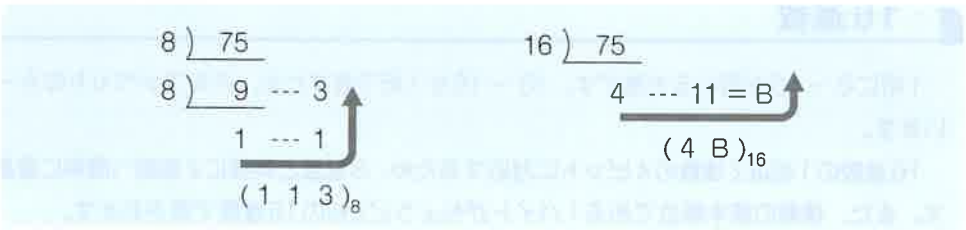


8進数，10進数，16進数の相互変換

10進数で表された数を8進数や16進数に変換するには，いったん2進数に変換するとよいでしょう。逆に，8進数や16進数で表された数を10進数に変換するには，桁を考慮した計算を行います。これらの関係をまとめておきましょう。



もちろん，2進数への変換を経由せず，除算法を用いて10進数から8進数や16進数への変換を行ってもかまいません。



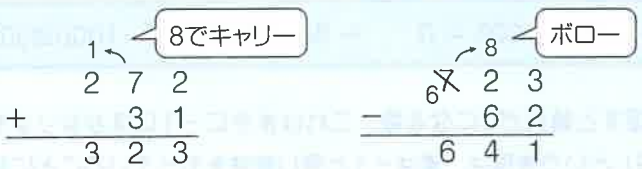
2進に加え，8進・16進の表現も学習したところで，それぞれ同じ数の表現がどのように異なるかを比較してみましょう。

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

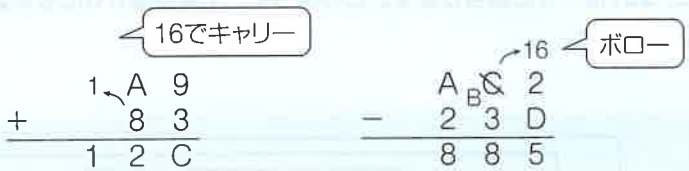
8進数，16進数の加減算

8進数や16進数の加減算は，10進数の筆算と同様の手順で計算できます。キャリーやボローが，8や16ごとだということに注意しましょう。

〔8進数〕



〔16進数〕



16進数の場合はA～Fを10～15に置き換えたほうがイメージしやすいかも知れません。好みの方法を用いてください。

重要ポイント

- **16進数では，10～15に対してA～Fを割り当てる**
- **8進数の1桁は2進数の3桁(000～111)に対応**
- **16進数の1桁は2進数の4桁(0000～1111)に対応**

1-3 負数表現と補数

重要度 ★★★

これまでは正の数の表現だけ考えてきましたが，負の数をどう表すかについても考えなければなりません。2進数で負の数を表現する方法はいくつかありますが，ここではまず，最もポピュラーな方法である，「補数」を使う考え方から説明していきます。

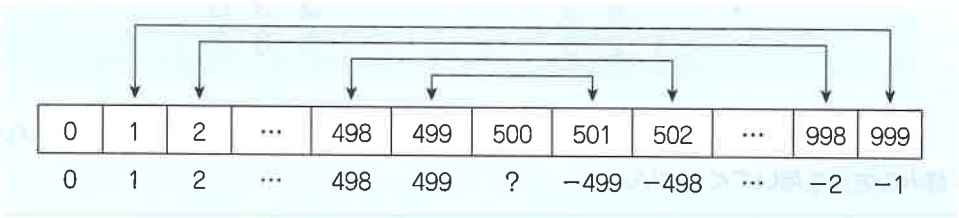
10進数3桁の世界

負数の導入のため，「10進数3桁の世界」を考えてみましょう。この世界で表される数は000(＝0)から999までです。この世界では1000という数は，(4桁目はないものとみなすので)0と同じです。

この世界では，不思議な計算が起こります。1と999を足すと，結果は0になるのです。

1 + 999 = 1000 = 0　← 3桁の世界なので，1000は000と同じ

さて，1を足すと結果が0になる数，これはまさに－1にほかなりません。10進数3桁の世界では，「999」という表現は，実は－1と同じ意味をもっていることになります。同じように，998は－2，997は－3に相当します。このように，加算結果が0になる数をペアにしていくと，次のようになります。



500だけが余りました。この数は＋500を表すのでしょうか，それとも－500を表すのでしょうか？

ここは「－500」としておくのが，この世界のルールです。このようにしておけば，最上位のけたが5以上ならば負数，そうでなければ0以上の数，という具合に正負の切り分けが簡単になるからです。これを「＋500」にしてしまうと，そう簡単にはいきません。

2進数の負数表現

同じような理屈で2進数の負数を表すことができます。8ビットの2進数（の世界）では，

00000001 ＋ 11111111 ＝ 1 00000000 ＝ 0

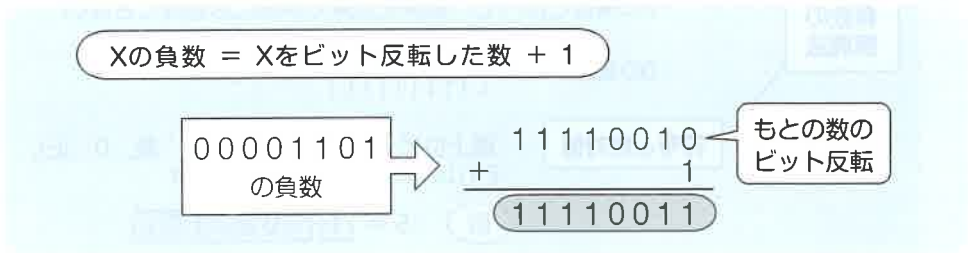
ですから，11111111は－1を表します。同様に11111110は，00000010（＝2）との加算結果が0になるので－2です。

これらの関係を表にまとめてみましょう。

10000000を（128ではなく）－128と決めた理由は，3桁の10進数における500の扱いと同じです。こうすることで「最上位ビットが1の場合は負数」に統一されるのです。この意味で，最上位ビットは**符号ビット**とよばれることもあります。

負数の作り方

ある2進数の負数は，足して0になる数を探せばよいのですが，これが意外に面倒です。そこで，普通は次のように求めます。



ある数とそれを反転した数の加算結果は必ず111…11になります。それに1を加えると，結果が0になるからです。

補数とは

これまでの手順でとった負数を，**2の補数**（ほすう）といいます。2の補数による負数表現は，コンピュータの世界で広く用いられています。というのも，負数を補数で表現することで，**減算処理を加算回路で処理できる**（加算用の回路と減算用の回路を別々に用意する必要がない）からです。

さて，補数とは文字通り「補う数」です。2進数には1の補数と2の補数があります。1の補数は補った結果「すべてのビットが1になる」数で，2（$=(10)_2$）の補数とは補った結果「けた上がりによってすべてのビットが0になる」数を意味します。

2進数	1の補数	もとの数をビット反転した数。加算結果は全ビット1
	2の補数	1の補数＋1。加算結果は全ビット0
10進数	9の補数	加算結果は全けたが9
	10の補数	加算結果は全けたが0

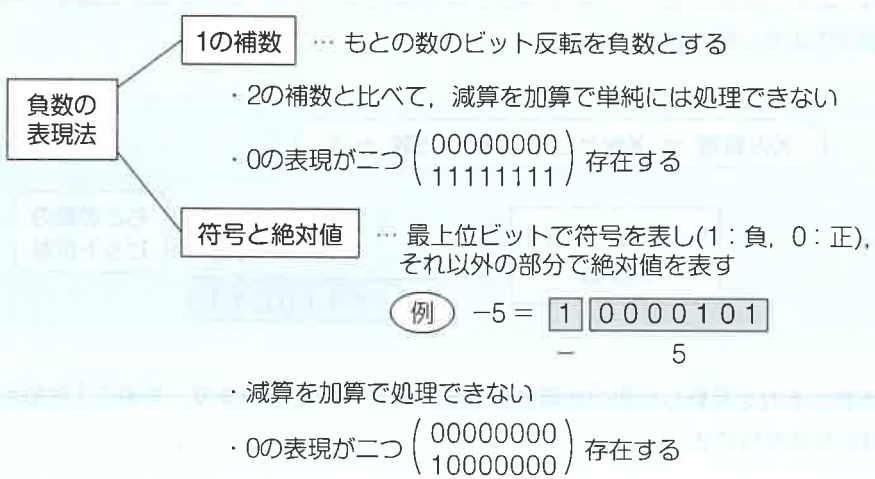
負数を2の補数で表す場合，n桁ならば

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

という範囲の数が表せることになります。たとえば8桁（8ビット）ならば，-2^7から2^7-1まで，すなわち−128～＋127までが，負数表現に2の補数を用いたときの表現範囲になります。

その他の負数表現

2進数の負数表現には，2の補数を用いる方法のほかにも，前述した1の補数を用いる方法や，符号と絶対値を用いる方法があります。



重要ポイント

- **2の補数：2進数の負数表現に用いる**
- **2の補数を作るには，「ビット反転した後に＋1」**
- **負数表現に2の補数を用いると，減算を加算回路で実現できる**
- **nビットで負数表現に2の補数を用いると，表現範囲は$-2^{n-1} \sim 2^{n-1}-1$**

1-4　小数の表現

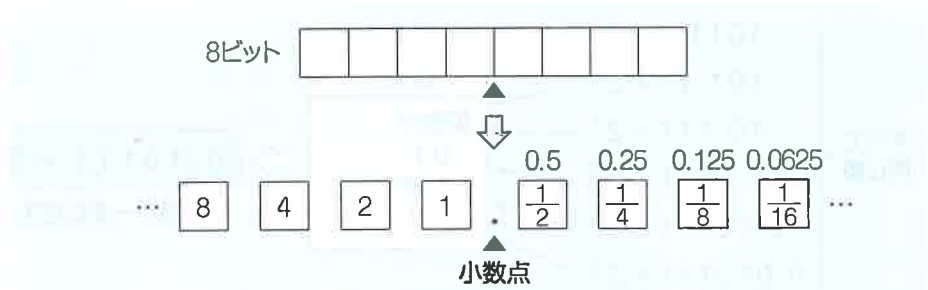
重要度　★★

コンピュータで数を表現する場合，ほとんどは，16ビットや32ビットといった「限られたビット数(桁数)の枠組み」を用いることになります。この枠組みの中で数を効果的に表現するため，いくつかの方法が考案・利用されています。

固定小数点数

固定小数点数とは，あるビット列のどこかの位置に小数点を固定する方式です。たとえば8ビットの中央に小数点を固定したなら，01010101は0101.0101＝$(5.3125)_{10}$を意味することになります。

まず，小数点を固定した場合の桁の大きさを示します。

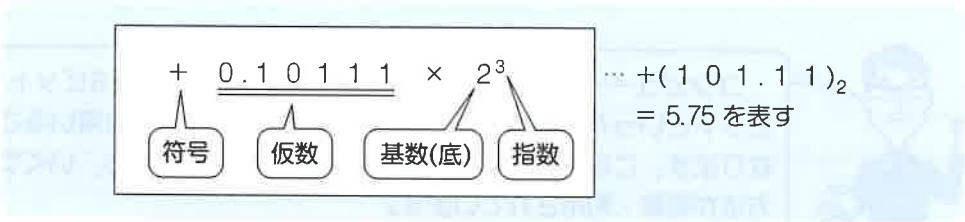


2進数の場合は桁が上がれば桁の値(重み)も2倍ずつ増加します。逆に桁が下がれば重みは半分になっていきます。

固定小数点数は，主に整数データの表現に用いられます。この場合，小数点の位置は右端になります。

浮動小数点数

固定小数点数はわかりやすいのですが，非常に大きな数や非常に小さな数を表す用途には適しません。そのような用途には，**浮動小数点数**を用います。浮動小数点数は主に，小数部も含んだ広い範囲の実数を表現するのに用いられます。

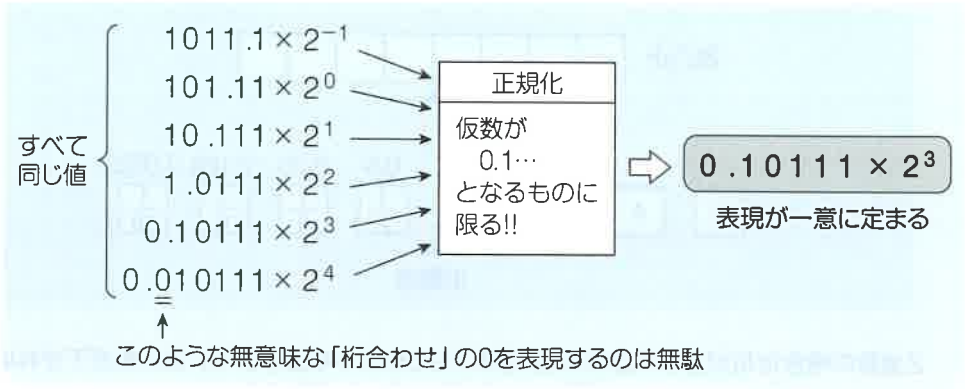


浮動小数点数は，**符号**，**仮数**（かすう），**基数**（底），**指数**を使って

（符号）仮数×基数^{指数}

の形式で数を表します。2進数であれば基数は必ず2です。

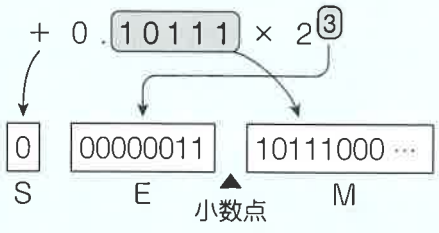
さて，この形式では一つの数に対してさまざまな表現が生まれてしまい，場合によっては無意味な「桁合わせ」の“0”が付くことによって，有効数字（意味をもつ数字の部分）の桁数が少なくなってしまうことも考えられます。そこで**正規化**を行います。



正規化は表現を一意に定め，**有効数字の桁数を確保する**ための操作です。正規化のために小数点を移動させることがあります。小数点を左に1桁移動させた場合には，指数を1増加させて調整します。逆に右に1桁移動すると指数は1減少します。

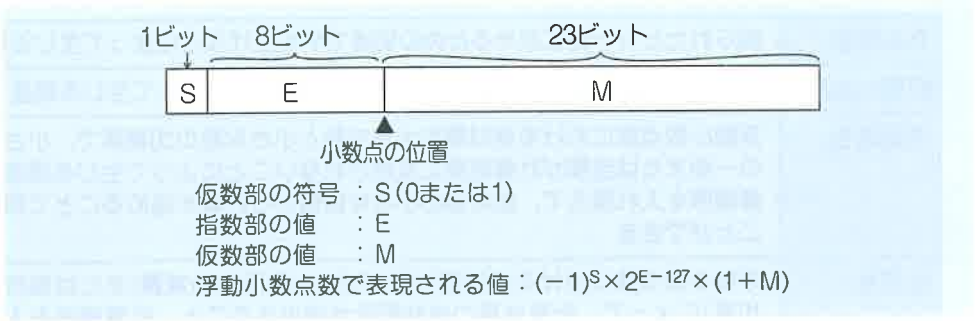
次に浮動小数点数の各要素を，規則に従って，定められた形式にあてはめます。

形式と規則	
符号部 S	1ビット 1：負　0：正または0
指数部 E	8ビット 負数は2の補数
仮数部 M	23ビット



IEEEによる浮動小数点数

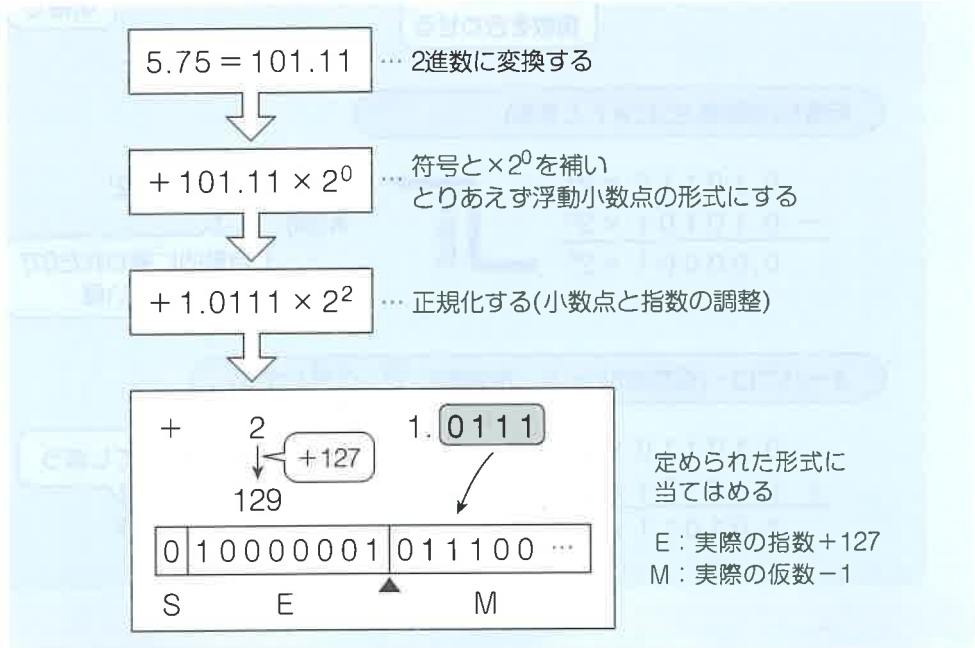
浮動小数点数の規則や形式はさまざまですが，中でも代表的なものがIEEE(アイトリプルイー：米国電気電子技術者協会)による規格(IEEE 754)です。以下は，単精度とよばれる32ビットの規格の様式です(このほかにも，倍精度とよばれる64ビットの規格があります)。



この規格の要点は，次の2点です。

・仮数を「1.…」となるよう正規化し，仮数部には小数点以下のみを設定
　→ 実際の仮数の値は「仮数部に設定された値＋1」となる
・指数部に補数表現を用いず，指数＋127を設定する
　→ 実際の指数の値は「指数部に設定された値－127」となる

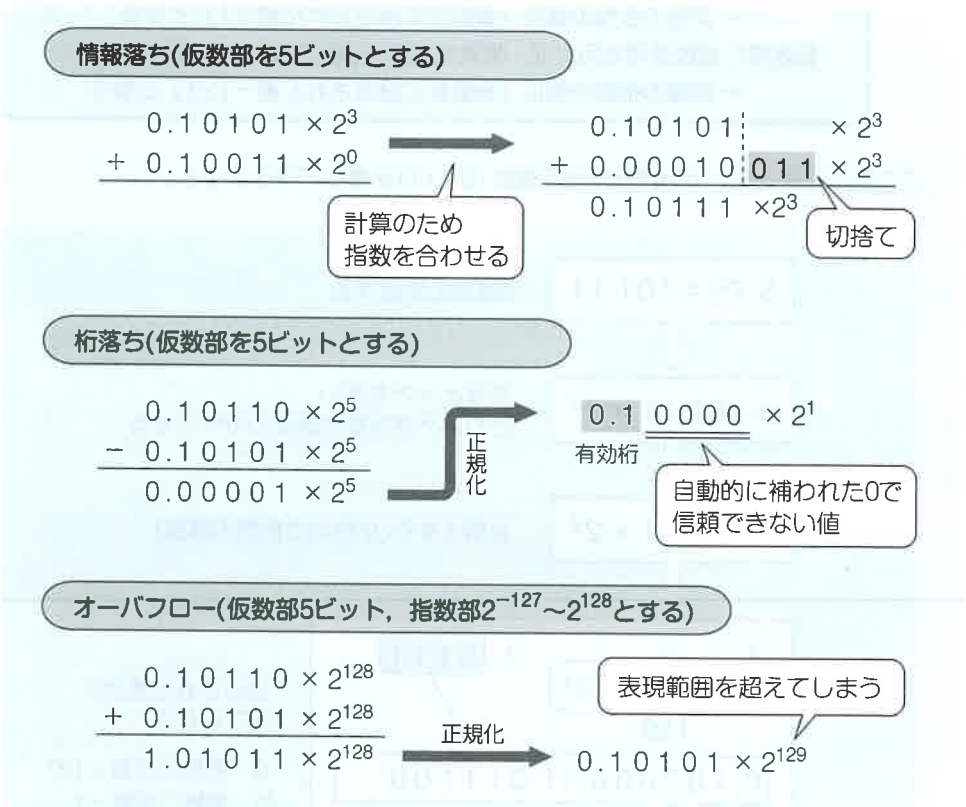
この形式に従って，10進数5.75(2進数101.11)を表してみましょう。



誤差

コンピュータは限られたビット数で数値を表さなければなりません。ところが数値は必ずしもビット数の範囲に収まっているとは限りません。つまり，表現した値は誤差を含んでいる可能性があるのです。

丸め誤差	限られたビット数に収めるための切捨てや切上げなどによって生じる誤差
打切り誤差	無限に続くような計算をある範囲で打ち切ることによって生じる誤差
情報落ち	浮動小数点数における絶対値の大きな数と小さな数の加減算で，小さな数の一部または全部が計算結果に反映されないことによって生じる誤差。計算順序を入れ替えて，絶対値の小さな数値から計算を進めることで抑えることができる
桁落ち	浮動小数点数におけるほぼ絶対値が等しい数同士の減算(または異符号の加算)によって，計算結果の有効桁数が減少すること。計算順序を入れ替えることで抑えることができる
オーバフロー	演算結果の絶対値が大きくなりすぎ，表現できる数の範囲を超えてしまう



なお，有効桁数とは，有効数字（意味のある数字）の桁数のことです。たとえば，先の図の桁落ちの例において，正規化後の数値のうち意味のある数字は“0.1”だけです。下4桁の0は「単に桁合わせのために付けられた位取りの数字」であり，意味のある数字ではありません。

また，誤差を数値として表現するときは，何を基準とするかによって次の2種類に分けることができます。

絶対誤差	真値(本当の値)と計算値(近似値)との差
相対誤差	真値に対する絶対誤差の割合(絶対誤差／真値)

参考：アンダフロー

オーバフローとは逆に，除算などによって演算結果の絶対値が小さくなりすぎ(0に非常に近い値となり)，表現できる数の範囲を超えてしまうこともあります。これをアンダフローといいます。

重要ポイント

【浮動小数点数】

- **(符号) 仮数 × 基数^{指数} の形で数を表現(多くの場合，基数＝2)**
- **仮数の有効桁数を確保するため，正規化を行う(表現をそろえる)**
- **正規化時の注意：仮数部の小数点をn桁左に移動したら，指数をn増やして調整**

【誤差】

- **情報落ち：絶対値の大きな数と小さな数の加減算で，小さい数の情報が反映されなくなる**
- **桁落ち：絶対値のほぼ等しい二つの数で，同符号の減算や異符号の加算の結果，有効桁数が減少してしまう**
- **オーバフロー：演算結果が表現できる数の範囲を超える**
- **丸め誤差：四捨五入や切上げ・切捨てで生じる誤差**
- **打切り誤差：計算処理を途中で打ち切ったために生じる誤差**

1-5 その他のデータ表現関連知識

重要度 ★

ここでは2進化10進数という，やや特殊な数値表現について説明します。また，文字を表現するための文字コードという概念についても紹介します。

BCDコード

1桁の10進数を表すには，以下のように4ビットあれば十分です。これを用いて，「1桁＝4ビット」に固定して数値を表すコードを，**BCD(2進化10進)コード**とよびます。

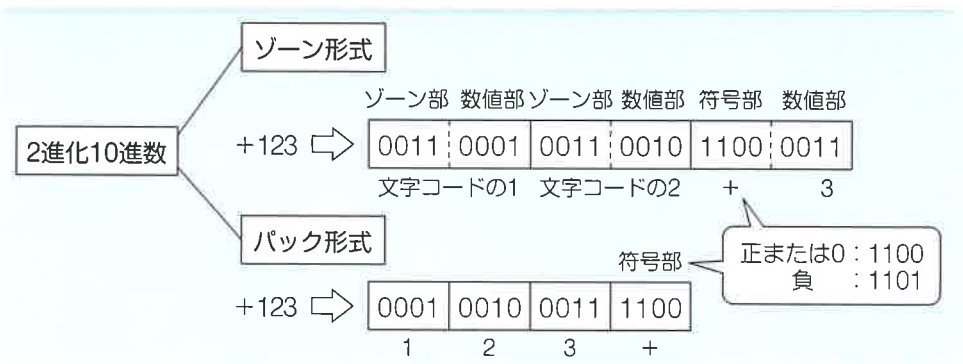
10進数の数値	0	1	2	3	4	5	6	7	8	9
BCDコード	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

BCDコードは，金額や数量を扱う事務処理系のプログラムでよく用いられる表現形式です。このような方法は，データ量が多くなる，処理に時間がかかるという欠点をもつものの，次のような利点があります。

・桁数(文字数)に制限がなく，どんな大きい(小さい)値も表現できる
・計算値を誤差なく表すことができる

ゾーン形式とパック形式

BCDコードを基本として表した数値を**2進化10進数**とよびます。具体的な方式には，**ゾーン形式**と**パック形式**があります。



ゾーン形式は，各BCDコードの前にゾーン部とよばれる4ビット(0011)を加えたものです。ただし，最後の桁はゾーン部ではなく符号部とします。符号部には，あらかじめ定めておいた2種類の値のうち，あてはまる方を設定します。

パック形式は，ゾーン形式からゾーン部を取り払った形式です。符号は数値の最後に付け加えます。

文字コード

コンピュータ内部では，すべてのデータは2進数で記録されます。文字データも例外ではありません。コンピュータ内部では“ABC…”といった文字ではなく“Aを表すビット列，Bを表すビット列，…”が記録されています。

このような「文字を表すビット列」を**文字コード**といいます。次に，文字コード(情報交換用符号)の割当て表の一例を示します。

ビット番号 b_7	b_6	b_5	b_4	列 / 行	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_3	b_2	b_1	b_0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	NUL	(TC_7)DEL	SP	0	@	P	、	p				ー	タ	ミ		
0	0	0	1	1	(TC_1)SOH	DC_1	!	1	A	Q	a	q			。	ア	チ	ム		
0	0	1	0	2	(TC_2)STX	DC_2	"	2	B	R	b	r			「	イ	ツ	メ		
0	0	1	1	3	(TC_3)ETX	DC_3	#	3	C	S	c	s			」	ウ	テ	モ		
0	1	0	0	4	(TC_4)EOT	DC_4	$	4	D	T	d	t			、	エ	ト	ヤ		
0	1	0	1	5	(TC_5)ENQ	(TC_8)NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
0	1	1	0	6	(TC_6)ACK	(TC_9)SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
0	1	1	1	7	BEL	(TC_{10})ETB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
1	0	0	0	8	FE_0(BS)	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
1	0	0	1	9	FE_1(HT)	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
1	0	1	0	10	FE_2(LF)	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
1	0	1	1	11	FE_3(VT)	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
1	1	0	0	12	FE_4(FF)	IS_4(FS)	,	<	L	¥	l	—			ャ	シ	フ	ワ		
1	1	0	1	13	FE_5(CR)	IS_3(GS)	−	=	M	]	m	}			ュ	ス	ヘ	ン		
1	1	1	0	14	SO	IS_2(RS)	.	>	N	^	n	~			ョ	セ	ホ	゛		
1	1	1	1	15	SI	IS_1(US)	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

機能符号(未定義)　機能符号(未定義)　国字符号部分　国字符号部分

機能コード(特殊な機能を表すための文字)

この文字コードを用いれば，アルファベットやカタカナをビット列で表現できます。文字コード表の上部にある4ビットが文字の上位ビット，左側にある4ビットが下位ビットです。たとえば，Aという文字は上位4ビットが0100，下位4ビットが0001，つまり，

文字A：01000001

で表されています。

補助単位

コンピュータは，時には百万バイトを超える情報を処理します。そこで，さまざまな補助単位（接頭語）が用いられます。

単位記号	読み方	意味
k	キロ	$10^3 = 1{,}000$
M	メガ	$10^6 = 1{,}000{,}000$
G	ギガ	$10^9 = 1{,}000{,}000{,}000$
T	テラ	$10^{12} = 1{,}000{,}000{,}000{,}000$

補助単位は“1,000”倍ずつ増加しますが，記憶容量（メモリの大きさ）などを表す場合には“1,024（$=2^{10}$）”倍ずつ増加することもあります。その場合は2kバイトは2,048バイトの情報となります。

小さくなる単位も紹介しておきましょう。主に時間を表すさいに用いられます。

単位記号	読み方	意味
m	ミリ	$1/10^3 = 1/1{,}000$
μ	マイクロ	$1/10^6 = 1/1{,}000{,}000$
n	ナノ	$1/10^9 = 1/1{,}000{,}000{,}000$
p	ピコ	$1/10^{12} = 1/1{,}000{,}000{,}000{,}000$

重要ポイント

【BCDコード】
- **10進数の1桁分を4ビット固定で表現**
- **1バイトに1桁分を格納するゾーン形式と，2桁分を詰め込むパック形式がある**

【文字コード】
- **各文字に固有のコード値（符号）を割り当てる**

【補助単位】
- **1,000倍するごとにk(キロ)，M(メガ)，G(ギガ)，T(テラ)**
- **1/1,000倍するごとにm(ミリ)，μ(マイクロ)，n(ナノ)，p(ピコ)**

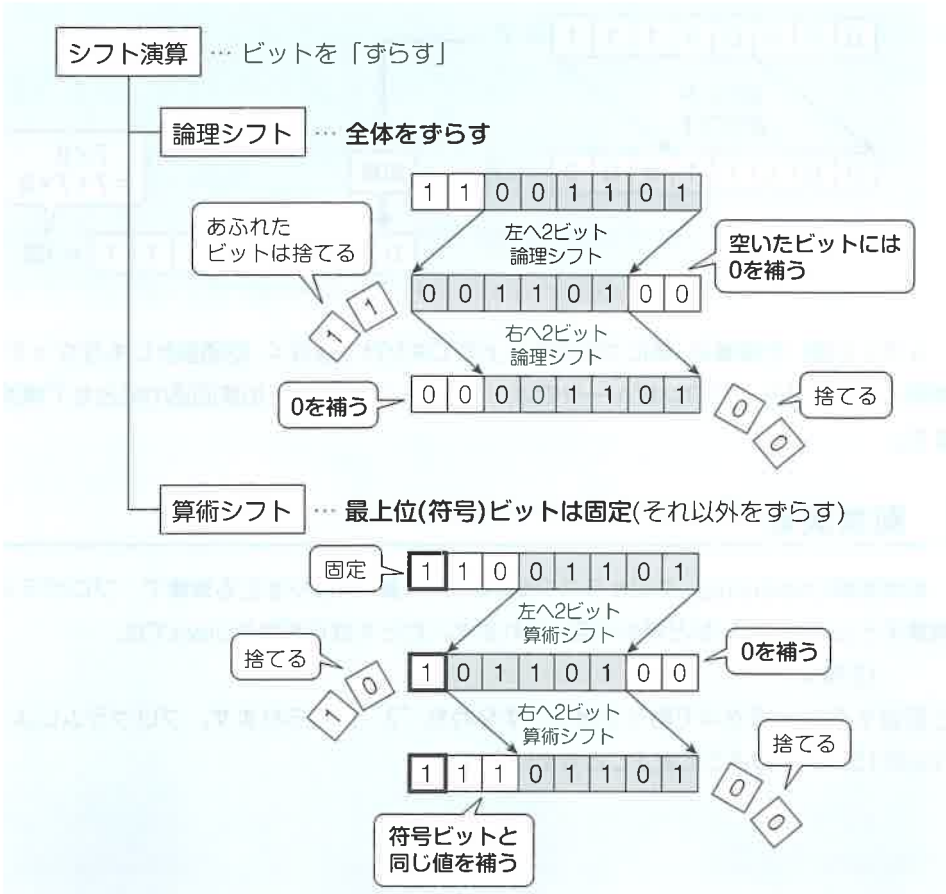
1-6　演算の関連知識

重要度　★★★

数値や文字コードなどを表すデータに対しては，加減乗除(＋－×÷)といった一般的な四則演算のほかにも，さまざまな演算を施すことができます。それらの演算を紹介するとともに，演算に伴って発生する現象についても説明します。

シフト演算

シフト演算は対象をビット列としてとらえ，指定したビット数だけ桁を「ずらす」演算です。ビット全体をずらす**論理シフト**と，最上位ビット(符号ビット)はずらさない**算術シフト**に分かれます。また，ずらす方向によって**左シフト**と**右シフト**に分類できます。



原則として，シフトによってあふれたビットは捨てられ，空いたビットには0が補充されます。ただし，算術右シフトに限っては空いたビットには「符号ビットと同じビット」を補充します。

なお，次の条件を満たす限り，左へ1ビットシフトした結果は元の2倍の値となり，右に1ビットシフトした結果は元の1/2倍の値となります。

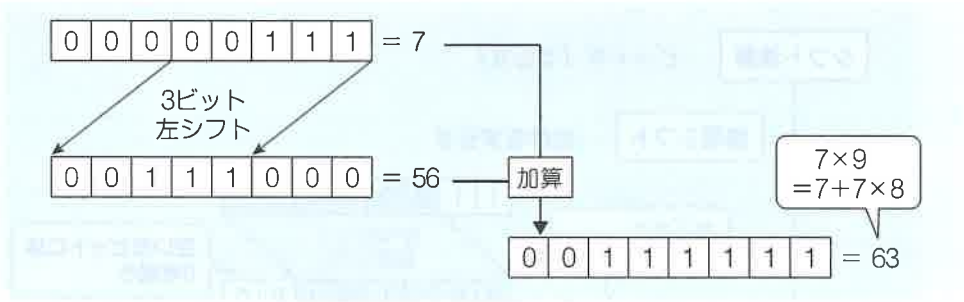
論理右/左シフト	ビット1があふれないこと
算術右シフト	ビット1があふれないこと
算術左シフト	正または0の場合　ビット1があふれないこと 負の場合　ビット0があふれないこと

シフト演算は，乗算・除算を行うためにも用いられます。たとえば7×9という乗算を計算する場合，加算だけで計算しようとすると

$$7 \times 9 = 7+7+7+7+7+7+7+7+7 = 63$$

のように加算を何回も行わねばならず，非効率です。

そこで，シフト演算を用います。左に1ビットシフトさせると，結果は元の数の2倍になることを思い出してください。つまり，左に3ビットシフトすると，結果は元の$2^3=8$倍になるのです。これを用いると，7×9は7×8＋7×1なので，シフト1回，加算1回で計算できます。



シフトを用いた乗算は，単にプログラム上の工夫だけではなく，回路設計にも役立っています。実際に乗算回路をもつコンピュータでは，それをシフト回路と加算回路の組合せで構築しています。

剰余演算

剰余演算（modulus，モジュラス）は除算（割り算）の余りをとる演算で，プログラム言語の演算子としては“%”などがよく用いられます。たとえばC言語やJavaでは，

15 % 4

と記述すると，15を4で割った余り，すなわち“3”が得られます。プログラムによっては，“mod(15，4)”のように表すこともあります。

重要ポイント

【シフト演算】
- ビット列の内容を左右に移動する（ずらす）
- 論理シフト：全体をシフト，空いた部分は0で補う
- 算術左シフト：符号ビット以外をシフト，空いた部分は0で補う
- 算術右シフト：符号ビット以外をシフト，空いた部分は符号ビットと同じ値で補う
- 左にnビットシフトすると2^n倍，右にnビットシフトすると2^{-n}倍

【剰余演算】
- 割り算の余りをとる
- 表現は“X % Y”，“mod(X，Y)”など

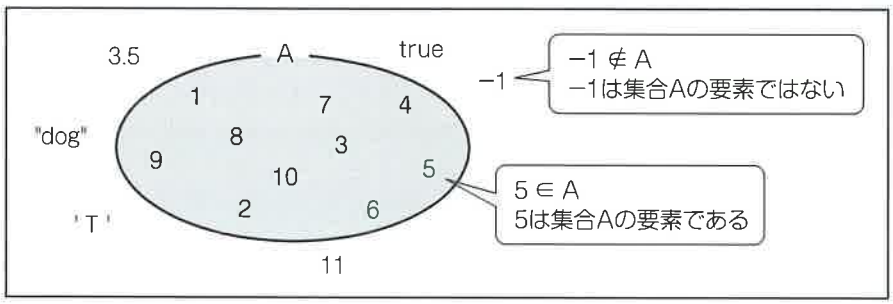
1-7 集合論

重要度 ★

集合論は，コンピュータに関するさまざまな分野で応用されています。集合論が応用されているものの例として，データベースなどが挙げられます。また，プログラムのアルゴリズムも，多くの場面で集合論を応用しています。

集合とは

集合とは，ある特定の性質をもった要素の集まりです。集合Aが「10以下の正の整数」であれば，1，2，7，10などはすべて集合Aの要素です。



集合を表現する方法には次の二つがあります。

①その集合の要素をすべて列挙する … A = {1，2，3，4，5，6，7，8，9，10}

②その集合の要素の特徴となる性質を示す … A = {x | xは10以下の正の整数}

集合の要素であるという関係は記号“∈”を，要素でないという関係は記号“∉”を用いて表します。

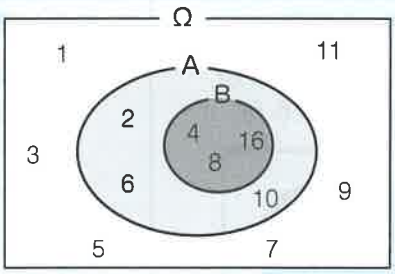
全体集合，空集合，部分集合

集合には，**全体集合**，**空集合**（くうしゅうごう）という特別な集合があります。

特別な集合
- 全体集合 … 対象世界そのものを表す(Ω)
  - 例　整数すべてが対象　Ω＝{ x | xは整数 }
  - 日本人すべてが対象　Ω＝{ x | xは日本人 }
- 空集合 … 要素をもたない集合(Φ)
  - Φ＝{　}

全体集合は対象とする世界そのもので，すべての集合は全体集合に含まれます。空集合は要素をもたない集合です。

ある集合の一部分で構成される集合を，**部分集合**とよびます。



上図の集合Ｂは集合Ａの部分集合であり，“Ａ⊃Ｂ”とも表現します。どのような集合も，全体集合(Ω)の部分集合です。また，どんな集合でも，空集合を部分集合としてもちます。

Ω＝{x | xは整数}　Ａ＝{x | xは2の倍数}　Ｂ＝{x | xは4の倍数}

であれば，Ω ⊃ Ａ ⊃ Ｂ ⊃ Φ という関係が成立します。

ベン図

円や四角を用いて集合の関係を表す表記法を，**ベン図**とよびます。

全体集合の要素のうち，集合Ａに属する要素は集合Ａの円の中に入り，集合Ｂに属する要素は集合Ｂの円の中に入ります。そして，ＡとＢに共通する要素は，二つの円が交差する部分に入ります。

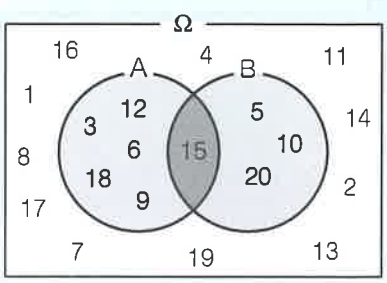
図の例は，

Ω＝{x | xは20以下の正の整数}

Ａ＝{x | xは3の倍数}

Ｂ＝{x | xは5の倍数}

です。



集合の演算

集合に演算を施すことによって，新たな集合を得ることができます。ここでは，それらの演算から**和集合**，**積集合**，**差集合**，**補集合**を取り上げます。

和集合 … 複数の集合のうち，いずれかに含まれる要素の集合

積集合 … 複数の集合のすべてに含まれる要素の集合

差集合 … 一方の集合には含まれ，もう一方の集合には含まれない要素の集合

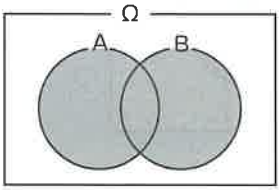
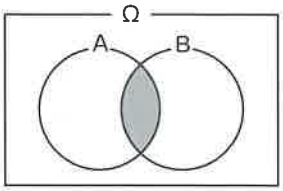
補集合 … ある集合に含まれない要素の集合

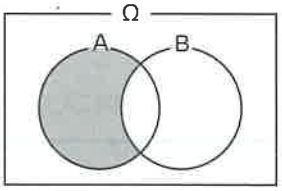
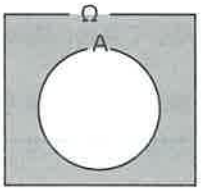
次の集合を例に，各演算のイメージをつかみましょう。

$\Omega = \{x \mid x$は0～9の整数$\} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$A = \{x \mid x$は2の倍数$\} = \{0, 2, 4, 6, 8\}$

$B = \{x \mid x$は3の倍数$\} = \{0, 3, 6, 9\}$

和集合　$A \cup B$	積集合　$A \cap B$
$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ $= \{0, 2, 3, 4, 6, 8, 9\}$ 	$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ $= \{0, 6\}$ 

差集合　$A - B$	補集合
$A - B = \{x \mid x \in A \text{ and } x \notin B\}$ $= \{2, 4, 8\}$ 	$\overline{A} = \{x \mid x \notin A\}$ $= \{1, 3, 5, 7, 9\}$ 

基本的法則

集合演算の基本的な法則は，以下のとおりです。

①同一の法則

(a)　$A \cup A = A$

(b)　$A \cap A = A$

②交換の法則

(a)　$A \cup B = B \cup A$

(b)　$A \cap B = B \cap A$

③結合の法則

(a)　$A \cup (B \cup C) = (A \cup B) \cup C$

(b)　$A \cap (B \cap C) = (A \cap B) \cap C$

④分配の法則

(a)　$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

(b)　$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

⑤吸収の法則

(a)　$A \cup \Phi = A$

(b)　$A \cap \Omega = A$

(c)　$A \cup \Omega = \Omega$

(d)　$A \cap \Phi = \Phi$

(e)　$A \cup (A \cap B) = A$

(f)　$A \cap (A \cup B) = A$

⑥否定の法則

(a)　$A \cup \overline{A} = \Omega$

(b)　$A \cap \overline{A} = \Phi$

⑦ド・モルガンの法則

(a)　$\overline{A \cup B} = \overline{A} \cap \overline{B}$

(b)　$\overline{A \cap B} = \overline{A} \cup \overline{B}$

⑧全体集合と空集合

(a)　$\overline{\Omega} = \Phi$

(b)　$\overline{\Phi} = \Omega$

⑨2重否定

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

重要ポイント

【集合の演算】

- **和集合 $A \cup B$：AとBの少なくともどちらかに含まれている要素の集合**
- **積集合 $A \cap B$：AとBに共通して含まれている要素の集合**
- **補集合 $\overline{A}$：Aに含まれない要素の集合**

【集合演算の法則】

- **分配則：$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$**
  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- **ド・モルガンの法則：$\overline{A \cup B} = \overline{A} \cap \overline{B}$**
  $\overline{A \cap B} = \overline{A} \cup \overline{B}$

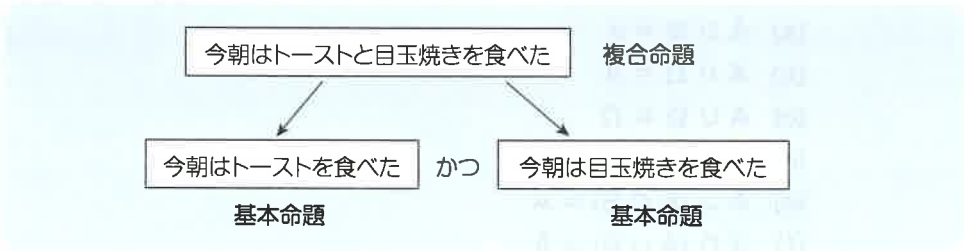
1-8 命題と論理式

重要度 ★★★

論理学のなかでもとりわけ論理代数は，ハードウェアの回路設計やプログラム設計などに重要な役割を果たしています。ここでは，コンピュータ技術に関係する論理の基礎の部分について，説明します。

命題とは

意味のある文章で，その内容が真(True)であるか偽(False)であるかが定まっているものを**命題**とよびます。命題には，**基本命題**と**複合命題**があります。



基本命題は，命題の最小要素で「これ以上分解できない」命題です。複合命題は複数の命題を組み合わせた命題です。

論理演算と論理式

前述の例では，二つの命題を“かつ”で結ぶことで，複合命題が出来上がっていました。このような，一定の規則に従ってある命題から別の命題を導く操作を，**論理演算**といいます。

論理演算は，「P かつ Q」ならば“P∧Q”といったように，特定の記号を用いて表現されます。このときに用いる“∧”などの記号のことを論理記号や論理演算子といいます。論理記号を用いて，論理演算の内容を計算式のように表したものを，**論理式**といいます。

最も基本となる論理演算は，次の三つです。

論理積(AND)：“かつ”を意味する演算。両者がともに真のときだけ真となる
論理和(OR)：“または”を意味する演算。少なくとも一方が真であれば真となる
否定(NOT)：“～でない”を意味する演算。真と偽が逆転する

各演算の具体的な結果を，次の表に整理します。提示した表は，各行が各命題の内容(真か偽)の組合せに対応しています。たとえば論理積の表の1行目は，「Pが真でQが真のとき，P∧Q

は真である」ことを示しています。このように，各命題とそれを用いた論理式の結果の関係を示す表を，**真理値表**といいます。

論理積（∧）（AND）

P ∧ Q：P かつ Q

P	Q	P ∧ Q
真	真	真
真	偽	偽
偽	真	偽
偽	偽	偽

論理和（∨）（OR）

P ∨ Q：P または Q

P	Q	P ∨ Q
真	真	真
真	偽	真
偽	真	真
偽	偽	偽

否定（$\overline{P}$）（NOT）

$\overline{P}$：P ではない

P	$\overline{P}$
真	偽
偽	真

¬を用いて“¬P”のように書くこともある

情報処理技術者試験では，記号の代わりに“AND”などの文字を用いて“X AND Y”や“NOT X”のように記すこともあります。また，以下のような記号を用い，通常の数式と似た表記で表すこともあります。

論理和 …“＋”　　論理積 …“・”

論理式では，論理和よりも論理積の方が優先的に評価されます。たとえば，“A＋B・C”という論理式は，まず“B・C”という論理積部分を評価し，その結果とAとの論理和をとることになります。

前述の基本的な三つのほかにも，さまざまな論理演算が存在します。以下に，試験でよく登場するものを三つほど紹介しておきます。

排他的論理和（eXclusive OR，**XOR**）：「一方が真，一方が偽」のときだけ真になる
否定論理積（Not AND，**NAND**）：論理積の結果の否定をとる
否定論理和（Not OR，**NOR**）：論理和の結果の否定をとる

排他的論理和（XOR）

P	Q	P XOR Q
真	真	偽
真	偽	真
偽	真	真
偽	偽	偽

否定論理積（NAND）

P	Q	P NAND Q
真	真	偽
真	偽	真
偽	真	真
偽	偽	真

否定論理和（NOR）

P	Q	P NOR Q
真	真	偽
真	偽	偽
偽	真	偽
偽	偽	真

参考：述語論理

たとえば“Xは動物である”という記述は，Xの部分に“人間”を入れれば真，“桜”を入れれば偽と評価できます。このように，入力する内容によって真偽が決定する記述のことを述語といい，述語を用いて命題を表現・評価する考え方のことを述語論理といいます。

基本的法則

論理演算においても，次の基本的法則が成立します。“≡”は，二つの論理式が全く等しい真理値をとる(同値である)ことを表します。

①同一の法則

(a) $P \vee P \equiv P$

(b) $P \wedge P \equiv P$

②交換の法則

(a) $P \vee Q \equiv Q \vee P$

(b) $P \wedge Q \equiv Q \wedge P$

③結合の法則

(a) $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$

(b) $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$

④分配の法則

(a) $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

(b) $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

⑤吸収の法則

(a) $P \vee (P \wedge Q) \equiv P$

(b) $P \wedge (P \vee Q) \equiv P$

⑥ド・モルガンの法則

(a) $\overline{P \vee Q} \equiv \overline{P} \wedge \overline{Q}$

(b) $\overline{P \wedge Q} \equiv \overline{P} \vee \overline{Q}$

⑦2重否定

$\overline{\overline{P}} \equiv P$

ビットデータの論理演算

論理式の考え方は，「真＝1，偽＝0」と置き換えれば，ビットデータに対して適用することもできます。たとえば，論理積，論理和，排他的論理和の三つについて，“1”と“0”を使って真理値表を書き直してみると，次のようになります。

P	Q	P AND Q	P OR Q	P XOR Q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

複数ビットからなるデータについても，同様の論理演算が可能です。この場合，各ビット位置ごとに独立して論理演算が行われます。以下は，4ビットのデータ 1010 と 0011 について論理演算を実行してみた例です。

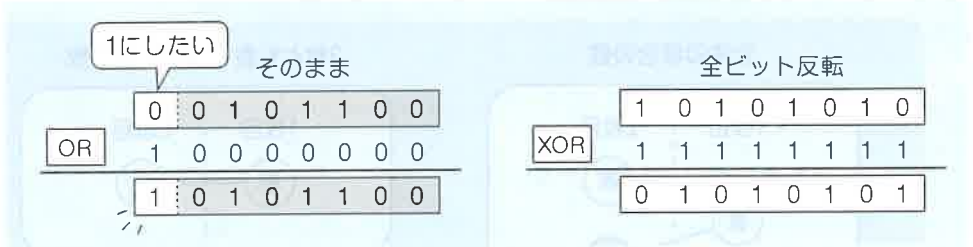
```
      1 0 1 0          1 0 1 0          1 0 1 0
AND   0 0 1 1    OR    0 0 1 1    XOR   0 0 1 1
-------------    -------------    -------------
      0 0 1 0          1 0 1 1          1 0 0 1
```

あるビット列に対し，特別に用意したビット列(**マスクパターン**)との論理演算をとることで，特定ビットを操作する処理を**マスク処理**といいます。使用する論理演算の種類によって，マスク処理の内容は異なります。

演算の種類	マスク処理	マスクパターン
AND	指定ビットを0にする	指定ビットが0，それ以外は1
OR	指定ビットを1にする	指定ビットが1，それ以外は0
XOR	指定ビットを反転する	指定ビットが1，それ以外は0

たとえば，ある8ビットデータの上位4ビットを0とし，下位4ビットはそのままとしたい場合には，データとマスクパターン00001111とのAND演算をとります。

8ビットデータの最上位ビットのみを1に変えたい場合には，マスクパターン10000000との論理和を求めます。8ビットデータのすべてのビットを反転させる場合には，マスクパターン11111111との排他的論理和を求めます。



重要ポイント

【主な論理演算】

- **論理積(AND)：両方とも真(1)のときだけ結果が真(1)**
- **論理和(OR)：少なくとも一方が真(1)ならば結果が真(1)**
- **否定(NOT)：真⇔偽，1⇔0を反転**
- **排他的論理和(XOR)：双方が同じ値ならば偽(0)，異なれば真(1)**

【主なマスク処理】

- **強制的に'0'にしたい部分：'0'と論理積(AND)**
- **強制的に'1'にしたい部分：'1'と論理和(OR)**
- **ビットを反転したい部分：'1'と排他的論理和(XOR)**
- **変化させたくない部分：'1'と論理積(AND)，'0'と論理和(OR)，'0'と排他的論理和(XOR)**

1-9 確率

重要度 ★★

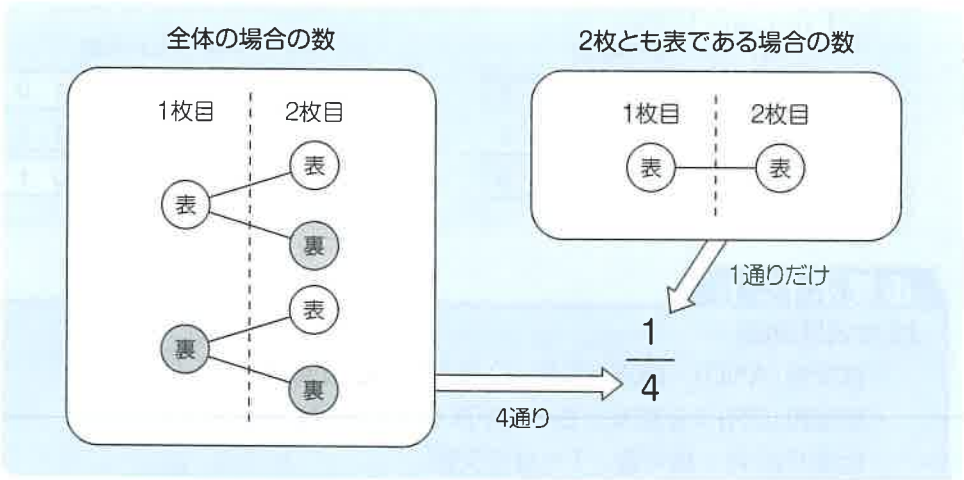
確率は，一見するとコンピュータにはそぐわないように思えますが，実は情報理論には欠かすことのできない知識です。ここでは，確率を計算するための基本的な知識について説明します。

場合の数

場合の数とは，ある事象が起こりえる場合（場面，パターン）をもれなく数え上げた値で，確率はこれを用いて計算できます。すべての場合が同じ確率で起こり得るならば，ある事象（ことがら）が起こる確率は，

その事象に該当する場合の数 ／ 起こり得る場合の数の合計（総数）

で求められます。たとえば，「2枚のコインを投げたとき，2枚とも表である確率」は，次のように求められます。コインに見かけの差はありませんが，別々のモノとして存在しているので，1枚目・2枚目として区別します。



なお，場合の数は計算でも求められます。たとえば，コインを2枚投げる場合の数は，

1枚目の場合の数 × 2枚目の場合の数

です。したがって，全体の場合の数は2×2＝4，2枚とも表である場合の数は1×1＝1なので，2枚とも表である確率は1/4と計算できます。

また，「2枚のコインを投げたとき，表と裏が1枚ずつ出る」確率は1/2です。なぜなら，「表

と裏が1枚ずつ出る」という事象には，

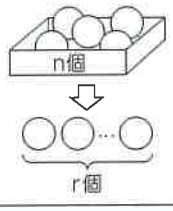
・1枚目が表，2枚目が裏

・1枚目が裏，2枚目が表

という二つの場合が該当するので，確率は 2/4 ＝ 1/2 となります。

順列と組合せ

順列(Permutaion)と**組合せ**(Combination)は，ともに「あるモノの集まりの中からいくつかを選択する」ことをモデルとした場合の数です。順列が並び順を考慮するのに対し，組合せは並び順を考えません。

n個の中から r個を選ぶ 	順列 $_nP_r$	$_nP_r = n(n-1)\cdots(n-r+1) = \frac{n!}{(n-r)!}$
	組合せ $_nC_r$	$_nC_r = \frac{_nP_r}{r!} = \frac{n!}{r!(n-r)!}$

※ $n! = n\times(n-1)\times\cdots\times2\times1$
(nの階乗)

4冊の本(A ～ D)の中から3冊を選ぶことを例にとります。この順列は，

1冊目の場合の数 × 2冊目の場合の数 × 3冊目の場合の数

＝ 4 × 3 × 2 ＝ 24 [通り]

と計算できます。

同じ例でも，組合せは順列とは異なる値をとります。というのも，ある3冊の本(A，B，C)の並べ方は，3×2×1＝6通り存在します。具体的には，

(ABC)，(ACB)，(BAC)，(BCA)，(CAB)，(CBA)

です。順列はこれらを6通りと数えますが，組合せは1通りと数えます。つまり，組合せは，

4冊の中から3冊を選ぶ順列 ／ 3冊の並べ方

＝ 4 × 3 × 2 ／ (3 × 2 × 1) ＝ 4 [通り]

で計算します。

確率の積

ある事象Aが起こる確率をP(A)，Aが起きたという仮定のもとでBが起きる確率を$P_A(B)$と表すとき，「AとBがともに起こる確率」を$P(A\cap B)$と表し，

$P(A\cap B) = P(A)\times P_A(B)$

が成立します。

たとえば，赤球と白球が2個ずつ入った袋から，連続して(取り出した球は戻さずに)2個の球を取り出すとします。このとき，1個目も2個目も赤球である確率は，

・1個目が赤になる確率
　＝2/4＝1/2
・「1個目が赤」という仮定のもとで，2個目が赤になる確率
　＝1/3　（分母，分子ともに1ずつ減っている）

なので，

$$1/2 \times 1/3 = 1/6$$

と求めることができます。

事象XとYの間に依存性がないときは，$P_X(Y) = P(Y)$となるので，単純に各確率を掛け合わせればよいでしょう。たとえば，先のコインの例は「1枚目が表，かつ2枚目が表となる確率」と考えてもよいので，

$$1/2 \times 1/2 = 1/4$$

と求めることもできます。

確率の和

二つの事象AとBがある場合，「AかBのうち，少なくともどちらか一方が起こる確率」を$P(A \cup B)$と表し，

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

で計算できます。AとBが同時に起こることはあり得ない(すなわち$P(A \cap B) = 0$)の場合は，単純に

$$P(A \cup B) = P(A) + P(B)$$

となります。

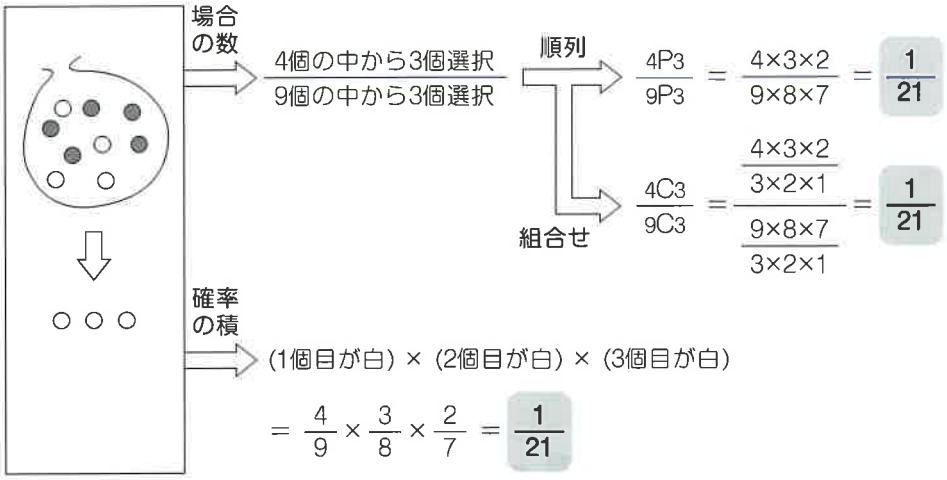
確率の求め方

では，順列や組合せを確率の計算に用いてみましょう。

例題

袋の中に白球が4個，黒球が5個入っている。この袋から3個の球を順番に取り出すとき，3個がすべて白球である確率を計算する。なお，取り出した球は袋には戻さない。

全部で9個の球が入っているので，そこから3個を選ぶ場合の数は「9個から3個を選ぶ場合の数」です。白球は4個なので，白球のみを取り出す場合の数は「4個から3個を選ぶ場合の数」となります。



全部同じ色なので，順列，組合せのどちらを用いても，確率は同じ値で計算されます。

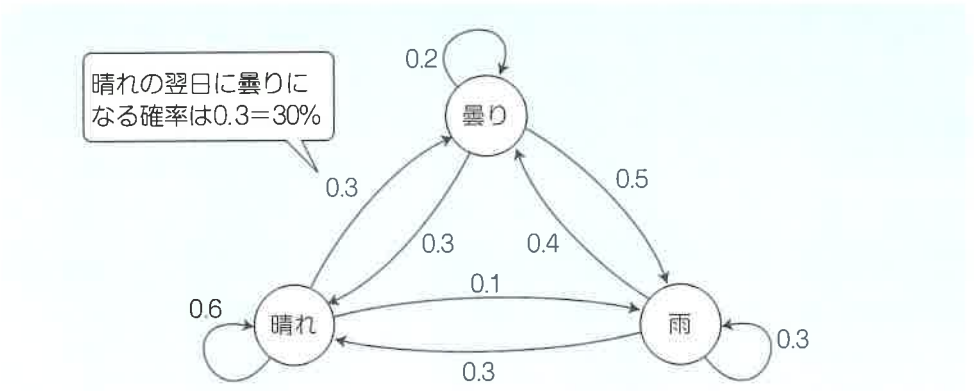
確率の積で直接計算する方法は，順列の応用です。計算式を見れば明らかなのですが，

$(4\times3\times2)/(9\times8\times7) \rightarrow (4/9)\times(3/8)\times(2/7)$

と表現を変えているに過ぎません。「2個目が白」の確率を表す3/8の分母が9ではなく8となっているのは，「1個目を引いて白だった」あとの状況（白球が3個・黒球が5個）を前提にしているためです。

状態の遷移と確率

たとえば一日ごとの天気の移り変わりを考えるような場合，「曇りの日の後には，雨になる確率が高い」といったように，ある状態からある状態へ遷移する（変化する）確率を求めてモデル化することがあります。モデルの表現には，各状態を“○”，状態間の遷移を“→”で表し（状態遷移図という），“→”にはその遷移が起こる確率を付記するという形式がよく用いられます。



このようなモデルを用いて「晴れの日から2日後が雨になる確率」を求める場合は，“晴れ”から2段階の遷移を経て“雨”になるパターンを洗い出し，それぞれの確率を求めて合計を取

ります。前述の例の場合は，

・“晴れ”の翌日に“晴れ”となり，その翌日に“雨”となる確率
　＝ 0.6 × 0.1 ＝ 0.06
・“晴れ”の翌日に“曇り”となり，その翌日に“雨”となる確率
　＝ 0.3 × 0.5 ＝ 0.15
・“晴れ”の翌日に“雨”となり，その翌日に“雨”となる確率
　＝ 0.1 × 0.3 ＝ 0.03

という三つのパターンが考えられるので，合計して0.06＋0.15＋0.03＝0.24が求める答えになります。

参考：単純マルコフ過程

ここで出た天気の例では，明日の天気が何になるかという確率は，「今日の天気が何か」だけで決まるようになっています。2日前や3日前の天気がどうだったかは無関係です。このように状態の移り変わりをシンプルにモデル化したものを，単純マルコフ過程といいます。

重要ポイント

- **事象が起きる確率＝その事象に該当する場合の数／全体の場合の数**
- **n個からr個選ぶときの順列の数：** $_nP_r = n(n-1)\cdots(n-r+1) = \dfrac{n!}{(n-r)!}$
- **n個からr個選ぶときの組合せの数：** $_nC_r = \dfrac{_nP_r}{r!} = \dfrac{n!}{r!(n-r)!}$

1-10　統計

重要度　★★

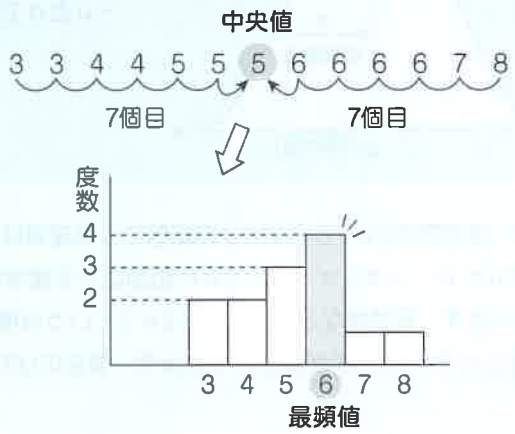
ここでは，統計学における基礎的な事項をいくつか紹介します。

データを代表する値

データの集まりがあったとき，それを代表する値を求めることがあります。テストにおける平均点などがそれにあたります。

平均	データの総和 ÷ データ数
中央値(メジアン)	データを整列させたとき，中央に位置する値 データが偶数個の場合は中央の2値を平均する
最頻値(モード)	最も度数の大きいデータ

中央値と最頻値について，説明を加えておきましょう。



上図のデータ列において中央に位置するデータは5で，これが中央値となります。最も数多く出現するデータは6で，これが最頻値に選ばれます。

データのばらつきを表す値

データのばらつきは，**分散**や**標準偏差**で表します。データを$x_1 \sim x_n$，平均値を$\overline{x}$とするとき，分散と標準偏差は次のように求まります。標準偏差は，一般にσ（シグマ）という記号で表されます。

偏差平方 ＝ (個々のデータ値－平均値)^2

偏差平方和 ＝ 偏差平方の合計 ＝ $(x_1-\overline{x})^2+(x_2-\overline{x})^2+\cdots+(x_n-\overline{x})^2$

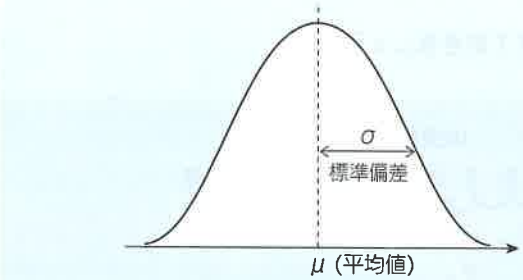
分散 ＝ 偏差平方和÷データ数 ＝ 偏差平方和÷n

標準偏差 σ ＝ $\sqrt{\text{分散}}$

分散は「各データと平均値との距離の２乗」を平均した値で，標準偏差はその平方根です。それらの値が大きいほど平均値から離れた値のデータが多く，小さいほど平均値付近に固まって分布していることになります。

正規分布

多くのデータ分布は，平均付近にデータが集まり，平均から離れるほど疎になっていくような形をとります。これを近似する代表的な分布に，**正規分布**があります。



・**μを中心に左右対称**
・**μとσで形状が定まる**

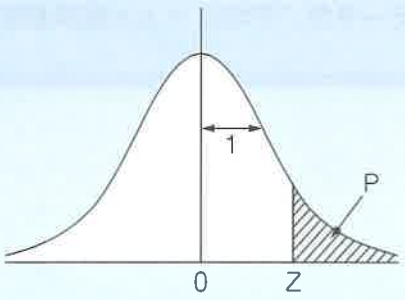
正規分布では，データが平均（μ）から$\pm k\sigma$の範囲に入る確率は，μやσによらず一定です。たとえば，データが$\mu\pm\sigma$，$\mu\pm2\sigma$，$\mu\pm3\sigma$の範囲に入る確率は，それぞれおよそ68.3%，95.4%，99.7%となります。製造業などで「良品率3σ」という目標を掲げることがありますが，これは1,000個製造したとき997個が良品となるよう，製品のばらつきを減少させることを意味します。

正規分布の適用

次の例を用いて，正規分布の適用例を見ることにしましょう。

1,000人の学生を対象にあるテストを実施したところ，平均値 $\mu = 60$，標準偏差 $\sigma = 10$ という結果となった。テスト結果が正規分布に従うと仮定したとき，以下の①，②を検討せよ。なお，表は平均値0，標準偏差1の標準正規分布表（片側）である。

Z	P
1.00	0.1587
2.00	0.0228
3.00	0.0013
1.65	0.05
1.96	0.025
2.33	0.010
2.56	0.005
3.09	0.001



① 50 ～ 70点の範囲に何人の学生が入っていると推測できるか。

② 合格率を95％に設定するとき，合格点をおよそ何点とすればよいか。

正規分布に従う限り確率Pは具体的な μ や σ によらないので，表の値を用いることができます。

①は $\mu \pm \sigma$ に入る人数です。表のZ＝1.00を見ると，確率P＝0.1587を得ます。この確率は片側のものなので，2倍して「$\mu \pm \sigma$ に入らない」確率0.3174を得ます。1からこれを減じれば「$\mu \pm \sigma$ に入る」，すなわち「50 ～ 70点に入る」確率を得ます。その値は0.6826なので，1,000人のうちおよそ683人が50 ～ 70点に入ると推測できます。

②は逆に確率から求めます。不合格率を0.05としたいわけですから，P＝0.05からZ＝1.65を得ます。これは「$\mu + 1.65\sigma$ を上回る」確率が0.05であることを意味します。

正規分布は左右対称ですから，「$\mu - 1.65\sigma$ を下回る」確率も0.05です。これに入る学生を不合格にすれば，合格率は0.95となります。その合格点は，$60 - 1.65 \times 10 = 43.5$ 点となります。

参考：その他の分布

確率統計の分野では，正規分布の他にもさまざまな分布が用いられます。主なものをいくつか紹介しておきましょう。

ポアソン分布 … ランダムに発生する事象の発生回数などに用いられる
指数分布 … ランダムとなるサービス時間の長さなどに用いられる
一様分布 … どの結果も均等な確率で発生する場合の分布

重要ポイント

【統計値】
- 平均：総和÷個数
- 中央値（メジアン）：整列したとき真ん中にいる値
- 最頻値（モード）：度数が一番多い値
- 分散＆標準偏差：データのバラつきを表す（大きいほどバラつきが大きい）

【正規分布】
- 平均値を中心とした，左右対称＆釣鐘型の分布
- データが“平均値 ＋ k×標準偏差”以上（以下）となる確率が決まっている

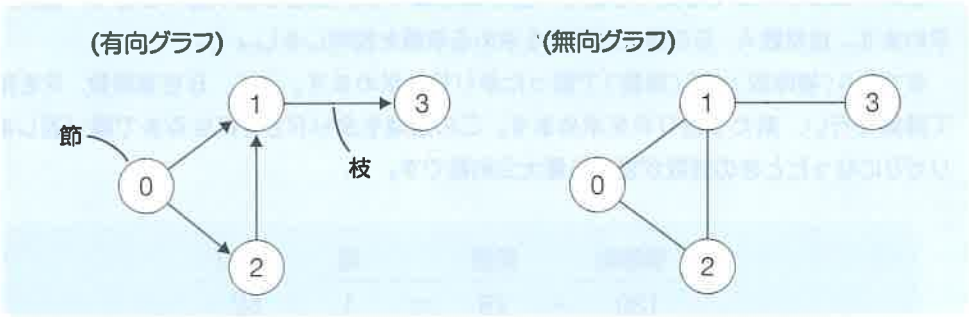
1-11　その他応用数学

重要度　★

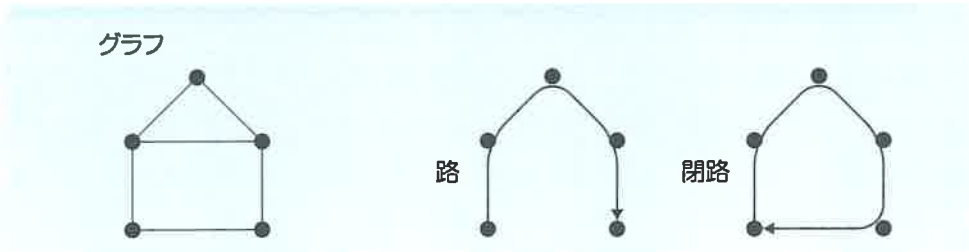
ある種の問題を解くプログラムを作るような場合，理解しておくと便利な数学理論をいくつか紹介します。

グラフ理論

グラフは節と枝から構成される構造で，スケジューリングや資源の配分，情報検索，言語の構文解析などの分野に応用されています。枝に向きを与えたグラフを**有向グラフ**，向きをもたないグラフを**無向グラフ**とよびます。向きの与えられた枝は，その方向にしか進むことはできません。



グラフ上のある節から出発し，枝をたどって複数の節を経由していく道筋のことを，路（小径）といいます。始点と終点が同じ，すなわち出発点に戻ってくる路のことを，**閉路**といいます。



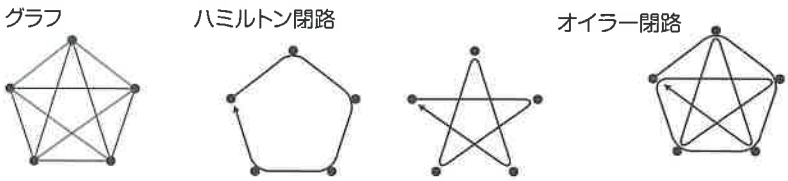
参考：路の種類

路や閉路は性質によって様々に分類することができます。有名なものは，ハミルトン路(閉路)とオイラー路(閉路)です。

ハミルトン路(閉路)：グラフ上のすべての節を一度通過する路(閉路)

オイラー路(閉路)：グラフ上のすべての枝を一度ずつ通過する路(閉路)

ハミルトン閉路をもつグラフをハミルトングラフ，オイラー閉路をもつグラフをオイラーグラフといいます。

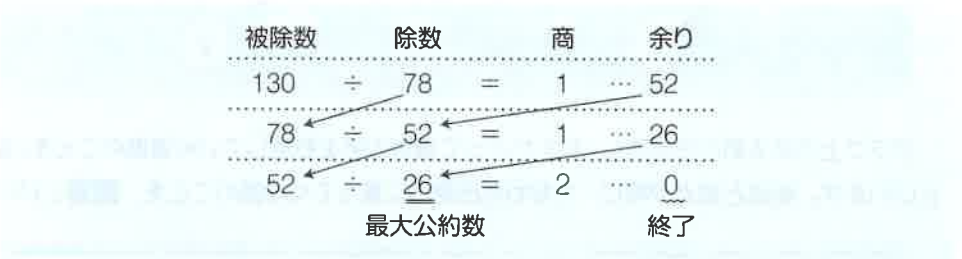


ユークリッドの互除法

ユークリッドの互除法は，二つの自然数の**最大公約数を求める**アルゴリズムです。最大公約数とは，二つの自然数をともに割り切ることのできる数(公約数)のうち，最も大きな数のことです。たとえば，10と15の最大公約数は5です。

このアルゴリズムでは文字どおり「互いに除算する」処理を繰り返すことで，最大公約数を求めます。自然数A，Bの最大公約数を求める手順を説明しましょう。

まず，A(被除数)をB(除数)で割った余りRを求めます。次に，Bを被除数，Rを除数として除算を行い，新たな余りRを求めます。この処理を余りRが0になるまで繰り返します。余りが0になったときの除数が求める最大公約数です。



エラトステネスのふるい

エラトステネスのふるいは，自然数N(＞2)以下のすべての素数を求めるアルゴリズムです。素数とは，1と自分自身以外では割り切れない数のことで，これを「ふるいにかけて」抽出します。

具体的には，次の処理を繰り返して，最終的に表に残った数が素数です。

〔1〕 2～Nを並べた表を用意する。

〔2〕 表を先頭から調べ，「未判定の数の中で最も小さな数」を素数と判定する。

〔3〕 〔2〕で判定した数の倍数を，表から取り除く。

〔4〕 未判定の数が残っている場合には，〔2〕に戻って処理を繰り返す。

(例)　N=20の場合

a)　2～20までの自然数を用意

2　3　4　5　6　7　8　9　10
11　12　13　14　15　16　17　18　19　20

b)　はじめの数2を素数と判定

(2)　3　4　5　6　7　8　9　10
11　12　13　14　15　16　17　18　19　20

c)　2の倍数消去

(2)　3　~~4~~　5　~~6~~　7　~~8~~　9　~~10~~
11　~~12~~　13　~~14~~　15　~~16~~　17　~~18~~　19　~~20~~

d)　素数かどうか未判定の数のうち，最小の3を素数と判定

(2)　(3)　5　7　9
11　13　15　17　19

e)　3の倍数消去

(2)　(3)　5　7　~~9~~
11　13　~~15~~　17　19

f)　d)，e)を繰り返す

g)　最終的に残った数が素数

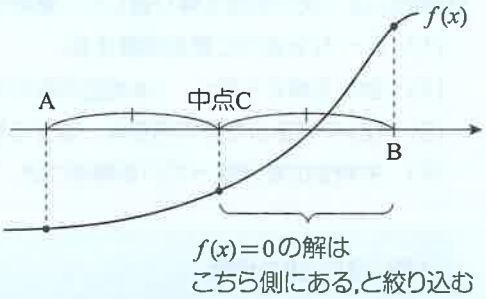
(2)　(3)　(5)　(7)
(11)　(13)　(17)　(19)

数式の解法

関数や方程式の解を求める手法には，以下のようなものがあります。

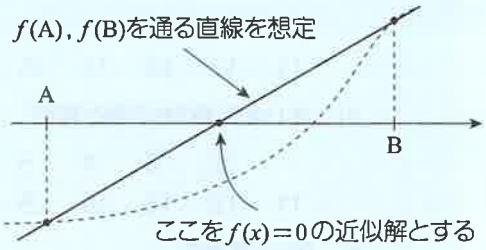
・二分法

区間の中点における関数の値を求め，区間端における関数の値と相互比較することで，解の存在範囲を半分に絞り込んでいく



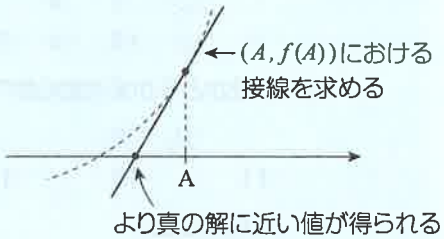
・補間法

複数の点における関数値が分かっている場合，その点を通る簡単な関数(一次関数など)を想定して解を導く



・ニュートン法

導関数(微分結果)を用いて，ある点における接線を求めながら近似解に近づいていく



重要ポイント

【グラフ理論】
- **節(ノード)を枝で結んだもの**
- **閉路：出発点に戻ってくる路(枝の連なり)**

【計算理論】
- **最大公約数を求める手法 … ユークリッドの互除法**
- **近似解を求める手法 … ニュートン法など**