コンピュータの原理（1）

～なぜ電気信号で計算できるのか～

Ver. 1.1

リナックスアカデミー矢越昭仁

2012/02/11

コンピュータは２進数を使っているといいますが、実際に電子回路を使ってどうやって計算しているか、数字や文字はどうやって表しているかという点を解説します。

目次

はじめに

コンピュータは 2 進法を使って動いている事は多くの人が知っていると思いますが、なぜそれで計算できるのか、プログラムが動くのか。そのメカニズムを普段意識することは少ないと思います。この講座では、コンピュータの動作原理のもっとも基本となる 2 進法とブール代数を手始めに、コンピュータの基本を学びます。

コンピュータの基礎

歴史

コンピュータは古くは「電子計算」や「電子頭脳」と呼ばれていました。単に計算するための機械であれば、古くはソロバン、計算尺などがありました。ソロバンは紀元前から存在しますし、計算尺は 1600 年代に発明されています。まだ同じころパスカルによって歯車式の計算機が発明されました。
より高速に大量の計算が必要とされるようになると、機械式では速度が追いつかなくなります。歯車の回転ではなく、電気式に計算しようとう試みが数多く生み出され、第二次大戦末期には幾つかの電子式計算機が誕生します。黎明期では ENIAC のように 10 進法を採用した電子計算機もありましたが、回路が複雑で故障が多く、計算速度も遅いことから、構造が単純な 2 進法を採用した電子計算機に淘汰されていきました。
さらに単純な計算だけでなく、条件判断や繰り返しといったプログラム機能を搭載するようになりました。
いわゆる計算機とコンピュータの最大の違いはプログラミングできるか否かだといえます。
数学の問題に当てはめるとイメージしやすいでしょう。

数字を使った計算問題	→	数値演算
文章題のように解法や証明を行う問題	→	論理演算

コンピュータに使用される部品もリレー（継電器）、真空管からトランジスタ、IC、LSI へと発展し、電子工学や物性物理といった分野とも影響しています。

ノイマン型コンピュータ

現在のコンピュータのほとんどは、ジョン・フォン・ノイマン(1903-1957、米国)が提唱した仕組みを応用していることから、「ノイマン型コンピュータ」と呼ばれています。

特徴は以下の通りです

- CPU（制御装置、演算装置）とメモリはバスで接続されている
  これはバスの速度が遅ければ、処理全体に影響が及ぶ事になります。この事を「フォン・ノイマン・ボトルネック」と呼びます。

- メモリはアドレスが割り振られた直線的な記憶装置
  メモリは一次元配列となっています。アドレスは連続していて、特定の場所を参照したり、書きかえる子事ができます。

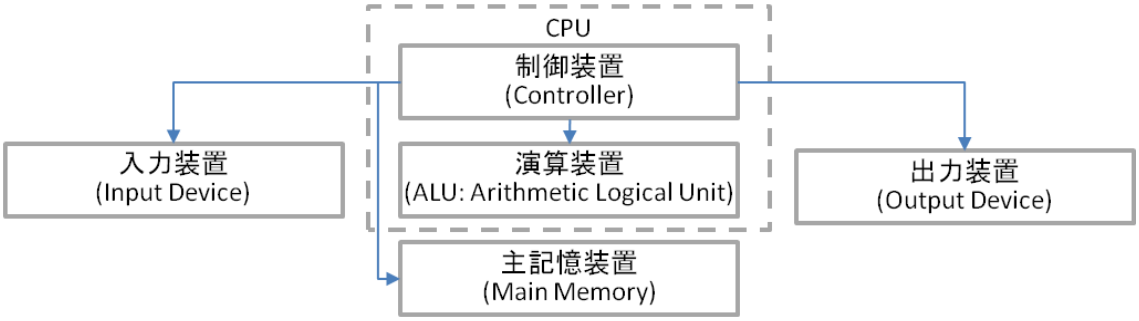
- データとプログラムは区別なくメモリに格納される（ストアドプログラミング方式）
  メモリ上の任意の場所を修正できるため、プログラムの改変も可能です。つまりウイルスに感染する可能性があります。

- 2 進法を採用している
  必須ではないですが、当時ノイマンが所属していた EDVAC プロジェクトが 2 進法を採用していたため、前提と考えられる事が多いです。

余談）黎明期の計算機
1946 年 ENIAC (Electronic Numerical Integrator and Computer=電子式数値積分・計算機、米国)
1949 年 Manchester Mark I、英国
1949 年 EDSAC (Electronic Delay Storage Automatic Calculator=電子遅延記憶式自動計算機、英国)
1951 年 EDVAC(Electronic Discrete Variable Automatic Computer＝電子式離散変数自動計算機、米国)

5 大装置

現在、流通しているコンピュータは以下の構成となっています。各装置の間でデータを受け渡す通路をバスと呼んでいますが、それは割愛しています。



1. 制御装置
   以下に解説する各装置の動作を制御します。多くのコンピュータでは、クロックと呼ばれる内部時計が発する信号に同期を合わせて、処理を行います。

2. 演算装置
   日本語では演算ですが、実際には算術演算（計算、Arithmetic）と論理演算（比較、条件判断、Logical）を合わせた機能を持ちます。プログラムとデータを制御装置が調整するタイミング（リズム）に合わせて読み書きします。

3. 主記憶装置
   プログラムを実行している間、プログラムとデータを保持しています。ほとんどのシステムでは電源を切ると記録は失われる（揮発性メモリ）ため、HDD、Flash Memory、DVD といった補助記憶装置が用意されています。

4. 入力装置
   処理に必要となるデータや、プログラムの動作を指示するパラメータなどを与える装置です。キーボード、マウスといった人が直接する物や、GPS、傾き検出(加速度)といったセンサー類も含まれます。

5. 出力装置
   処理結果を提供する装置。ディスプレイ、プリンタ、スピーカなどの人向けの装置以外にも、自動車のアクセル、ブレーキや他のコンピュータといった機械を含みます。

今回は、このうち演算装置の基本構造について解説することになります。

2 進法と電子回路

ENIACでは 10 進法を使っていましたが、実際の物理現象では 2 値を持つ事が多く存在します。例えばON/OFF、閾値より大きい小さい、磁気のN極・S極といったものです。こうした 2 つの状態を基本とすることで、メカニズムが単純になり処理が高速になります。その為現在のコンピュータは 2 進法を基本として演算を行っています。
2 進法により必要な演算ができることは、1800 年代にブールによって証明されており、現在もコンピュータの基本概念となっています。

ブール代数

ブール代数(Boolean algebra)は、19 世紀中ごろにジョージ・ブール（1815-1864、英国）によって考案されたといわれます。記号論理学(いろいろな命題を記号で表現し、矛盾なくその変位を表す学問)の基礎となっています。「物の考え方」たとえば三段論法などを記号で表現し、客観的・抽象的に表す事ができます。コンピュータのプログラムを実現するには非常に便利な考え方です。

算術演算の加減乗除に相当する記号（演算子）としては、A⇒B（AならばB、論理包含）、A∧B（AかつB、論理積、交わり）、A∨B（AまたはB、論理和、結び）、¬A（Aでない、否定、補集合）があります。これらを論理式と呼びます。
ブール代数自体はいくつかの要素からなる集合A、B、∀（全て、母集団）を扱います。集合を表すベン図に代表的な論理式を表すと以下のようになります。

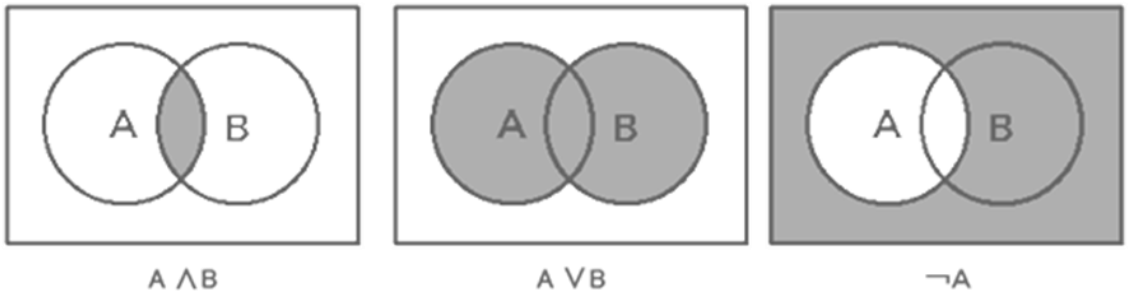


図 1:基本的な論理式とベン図

コンピュータでは集合の要素が0または1に単純化されています。2 進数を当てはめると、上記の組み合わせは非常に簡単なものとなります。
2 進法による論理式を表す例として、簡単な二つのスイッチをもつ電気回路の図を用意しました。スイッチを押すとON（1）、離すとOFF（0）とします。

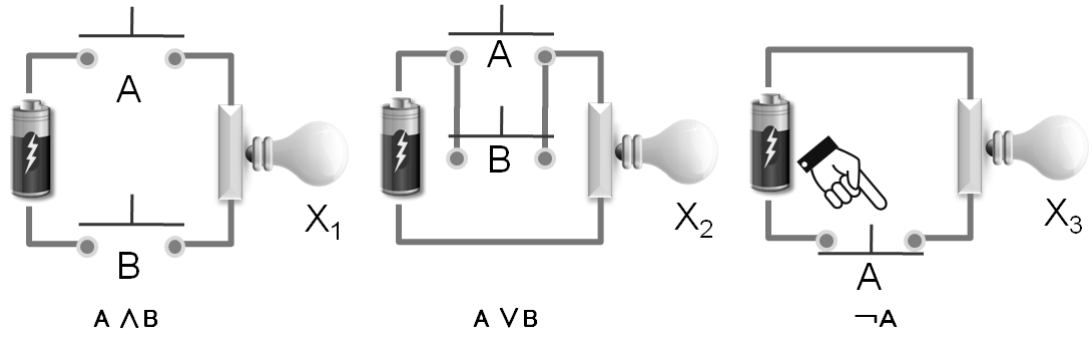


図 2:基本的な電子回路

上図において、スイッチAとスイッチBを押す・離すと、電球Xがどうなるかを考えます。「スイッチAおよびBを押した場合」「スイッチBだけを押した場合」は、それぞれ「A=1、B=1」「A=1、B=0(スイッチを押さない限り Off の場合)」として表現されます。

先の回路図で「スイッチAがON、かつ、スイッチBがON、ならば電灯Xが点灯する」とう命題を論理式で表すと

$$A \wedge B \Rightarrow X$$

となります。
この資料では読みやすくするために、「∧」のかわりに「and」、「∨」のかわりに「or」、否定は「not」とします。そして、変数A、Bと論理演算結果の総当たり表の事を「真理値表(Truth table)」と呼びます。

表 1:基本的な電子回路(論理式)の真理値表

A	B	A and B(X_1)	A or B(X_2)	not A(X_3)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

この and, or, not の 3 要素があれば、全ての論理式を作る事が可能です。つまりコンピュータの論理回路に用いられる電子回路はこの 3 要素の組み合わせだけで作る事が可能です。

ブール代数では、これら論理式を組み合わせて 6 つの法則(恒等式)からなります。

1. 幂等律(べきとうりつ):同じ要素に対し and や or を行っても、値は変わらない。
   X and X →X
   X or X → X

2. 交換律:論理演算の要素を入れ替えても値は変わらない。
   X and Y → Y and X
   X or Y → Y or X

3. 結合律:カッコ()を使って同じ演算の順序を変えても値は変わらない。
   (X and Y) and Z → X and (Y and Z)
   (X or Y) or Z → X or (Y or Z)

4. 吸収律:二項演算の結果に、含まれる要素と別の演算を行うと後の要素と等しくなる。
   (X and Y) or X → X
   (X or Y) and X → X

5. 分配律:三項演算は、先の二項と三項の and/or を反転させた演算結果と、三項めの反転させた演算と等しくなる。
   (X or Y) and Z → (X and Z) or (Y and Z)
   (X and Y) or Z → (X or Z) and (Y or Z)

6. 補元律:二項演算の結果が「∀ すべて」または「⊥ 無」になる。
   X or notX → ∀
   X and notX → ⊥

表 2:分配律(5)の真理値表

X	Y	Z	XorY[*1]	(*1)and Z	X and Z[*2]	Y and Z[*3]	(*1)or(*2)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	0	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	1	0	1
1	1	0	1	0	0	0	0
1	1	1	1	1	1	1	1

ド・モルガンの法則

ブール代数の応用として「ド・モルガンの法則(De Morgan's laws)」があります。この法則はコンピュータ業界では、特に良く使われる機会が多いことから、情報処理系の試験でも出題されます。「二項演算で全体を否定したものは、要素ぞれぞれを否定し、逆の演算を行ったものと等しい」というものです。

not (X or Y) → not X and not Y
not(X and Y) → not X or not Y

この式を覚えていれば、複雑な if 文や電子回路を、より簡単に表現する事ができます。

表 3:ド・モルガンの法則真理値表

X	Y	XorY*1	not(*1)	notX*2	notY*3	(*2)and(*3)
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

たとえば、

『「私は日本人で、かつ男性です」という人以外の人物』(not(X and Y))を表す文章は
「私は日本人ではい、または男性ではない」(not X or not Y) となる事を表しています。

排他的論理和

論理和(or)の応用で、排他的論理和(exclusive or/ X-or)という演算子もコンピュータの世界ではよく使われます。xor は、XとYの値が異なるとき、1 になります。

(notX and Y) or (X and not Y)
(X or Y) and (notX or not Y)
(X or Y) and not(X and Y)

無理に回路図にすると、3 路スイッチでしょうか。よく民家の階段にある電灯のスイッチなどに使われます。On/Off ではなく、階段の上と下で切り替えると、電灯が On/Off になるあれです。
スイッチの状態を表す処に若干の違和感がありますが、スイッチA／Bが同じ状態だと Off、違う状態だと On になる動作となります。

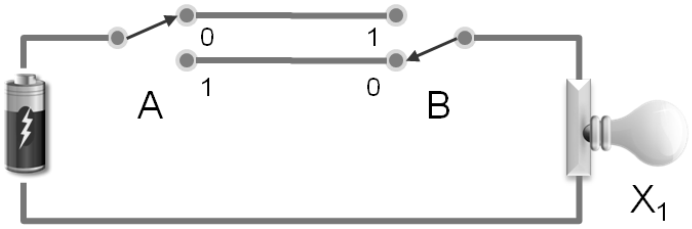


図 3:3 路スイッチ概念図

XOR も 2 進数の総当たり表を作ると、次のようになります。

表 4:XORの真理値表

X	Y	notX*1	notY*2	(*1)andY	Xand(*2)	XxorY
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

2 進数の値A、Bに対し、xor を計算した結果に、再度AまたはBで xor を計算すると、もう片方の値を再現できるという特性があります。

表 5:XORを用いた計算例

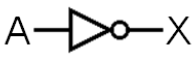
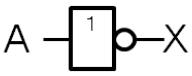
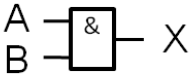
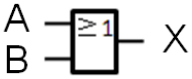
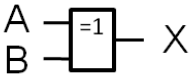
Aの値	0	1	0	1	1	0	1	0	0	1	
Bの値	1	0	1	0	1	0	1	0	1	0	
A xor B*1	1	1	1	1	0	0	0	0	1	1	
(*1) xor B	0	1	0	1	1	0	1	0	0	1	←A と同じ
(*1) xor A	1	0	1	0	1	0	1	0	1	0	←B と同じ

高速に処理できることから、RAIDディスクのチェックサム(パリティ)や、SSL暗号化、画像の重ね合わせ(カーソル)処理などに用いられています。また算術演算でも重要な役割を持ちます。

電子回路

今まで解説した and, or, not, xor は最も基本的な論理計算の演算子で、数値演算の＋、－、×、÷に相当します。その為、電子回路を設計する場合にもこれら 4 つの部品(素子)を用います。ハードウエアを設計する場合に用いる記号は以下のようになります。

表 6:基本的な論理回路素子

	MIL[1]規格	JIS/IEC 規格
NOT素子		
AND素子		
OR素子		
XOR素子		

実際の電子回路では、On/Off ではなく、電圧が高い Hi と、低い Low が用いられます。さらに Hi/Low も 2 値的に動作するわけではなく、若干のずれが生じます。信号が入力され、出力が安定するまでのタイムラグを閾値(threshold)と呼びます。

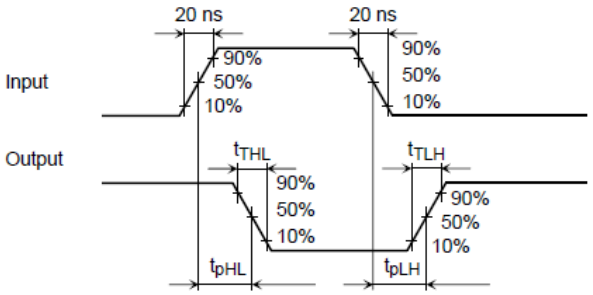


図 4:スイッチング特性例(東芝 TC4001BP/BF/BFT)

[1] Military Standard： 米国防総省が規定する調達品の規格

1つの論理素子を作るのに、数個のトランジスタとダイオードや抵抗などの電子部品が必要となります。これらを一まとめにしたものが IC(integrated circuit、トランジスタは数十個～数百程度)と呼ばれ、特に論理素子の集合を TTL(Transistor-Transistor-Logic)と呼んでいました。現在ではさらに集積度の高い LSI(Large Scale Integration 、トランジスタ数は 1,000～数十万個)、VLSI(Very Large Scale Integration 、数千万以上)と呼ばれる半導体があります。どれも基本は IC ですが、その集積度から呼び名が変わり、時代とともに搭載されるトランジスタ数も飛躍的に増大しています。

また自由に回路を設計できる LSI は ASIC(Application Specific Integrated Circuit)とよばれ、今日では組込機の主流になっています。昔のように TTL を何個も半田付けで繋いで、論理回路を作る(ワイヤードロジック)は、ほとんど見られなくなりました。

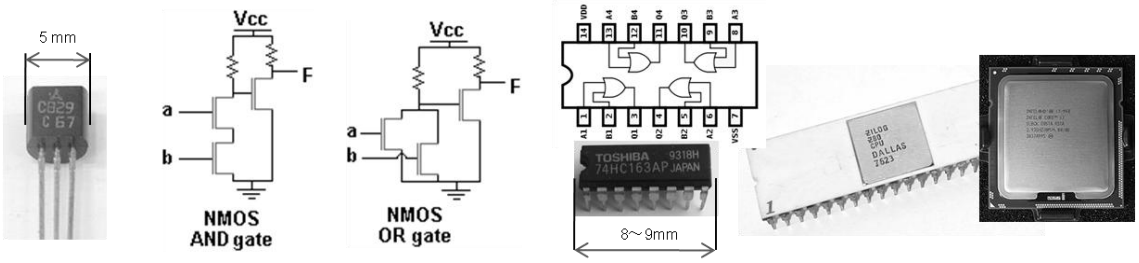
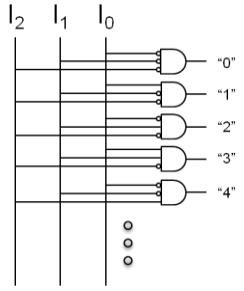


図 5:半導体の歴史

さて電子回路設計で、どんな処理ができるのか。非常に簡単な例として 10 進→2 進変換回路は以下のようになります。



I_2	I_1	I_0	各出力論理式
0	0	0	not I_2 and not I_1 and not I_0
0	0	1	not I_2 and not I_1 and I_0
0	1	0	not I_2 and I_1 and not I_0
0	1	1	not I_2 and I_1 and I_0
1	0	0	I_2 and notI_1 and not I_0
1	0	1	not I_2 and I_1 and not I_0
1	1	0	not I_2 and not I_1 and I_0
1	1	1	I_2 and I_1 and I_0

図 6:2 進 10 進 3 ビットデコーダ(一部)

この回路では、入力信号 I1～3 の状態に応じて、"0"～"7"の 8 個の出力のうち、どれか一つが 1 となる設計です。紙面の都合で"4"までしか図示していませんが、出力の論理式を参考にしてください。
また、AND素子の入力直前で否定(反転)していますが、この設計方法を「負理論」と言います。

図 7:NAND(Not AND)回路の例

半加算器と全加算器

長々と解説しましたが、AND と XOR を組み合わせると、2 進数 1 桁同士の足し算を行う事ができます。
例えば A,B を入力とした場合の答えをXとした場合、算術演算であれば

A ＋ B → X

となりますが、論理式では出力桁も常に一桁なので、Xを C,S の 2 つの値とします。

表 7:1 進数一桁同士の足し算

入力		出力	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

論理式

A and B → C

A xor B → S

A,B の和の結果について、2 桁目 C(and)は、桁溢れを意味します。これで一桁の足し算は可能なのですが、桁上がり処理が中途半端に終わっています。この回路は半加算器(Half Adder)と呼ばれます。

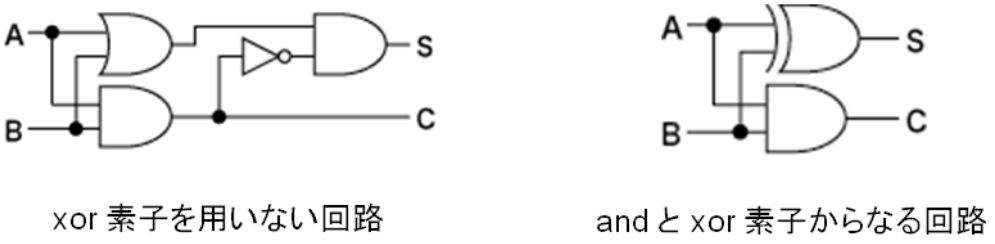


図 8:半加算器回路図

半加算器を 2 台連結し、桁上がり処理を取り込むよう修正したのが全加算器(Full Adder)です。

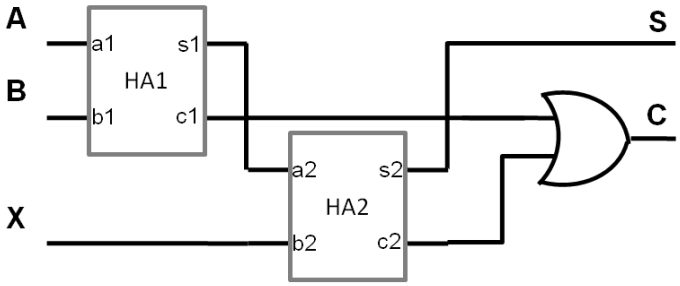


図 9:全加算器回路図

全加算器は半加算器 2 つからなり、下位の計算結果で生じた桁上がりを取り込むXが増えた形になっています。

表 8:全加算器の真理値表

入力			HA1		HA2		出力	
A	B	X	s1	c1	s2	c2	C	S
0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	1
1	0	0	1	0	1	0	0	1
1	1	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0	1
0	1	1	1	0	0	0	1	0
1	0	1	1	0	0	0	1	0
1	1	1	0	1	1	1	1	1

全加算器を連結することで算術演算を行う事ができます。この算術演算に用いる回路をアキュムレータ(Accumulator)と呼び、加算器の数が一度に計算できる 2 進数の桁数になります。

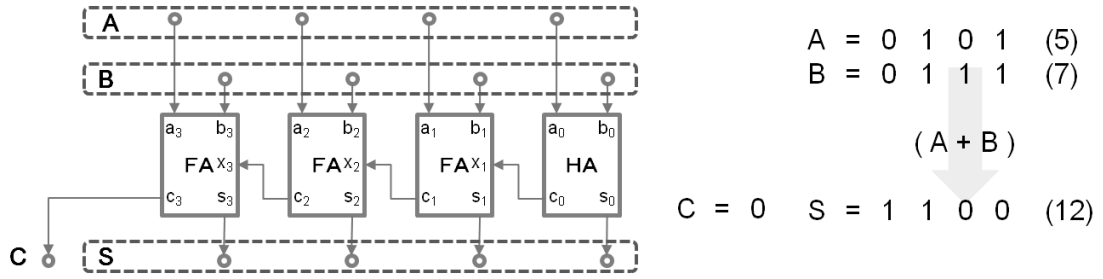


図 10:4bit アキュムレータ

上図において、a0,b0 など最下位の桁を LSB(least significant bit)、a3,b3 など最上位の桁を MSB(most significant bit)と呼びます。

アキュムレータの桁数を CPU の語長(ワード、Word length)とよび、性能を現す評価軸のひとつになります。初代ファミコンが 8bit、スーファミは 16bit、Windows が動く PC が 32bit、Windows 7 用 PC やプレステ 3 などは 64bit で、最近普及しはじめています。
桁数が多いほど、一度に計算できる数字も大きくなります。ただし、その分メモリ消費量も多くなるとう欠点があります。メモリ量が大きくなれば、当然実行形式プログラムのファイルサイズや、ネットワークでやりとりする再のダウンロード時間なども長くなります。
なお現在のコンピュータは、その殆どが CPU とメモリ間のやり取りを Byte 単位で行います。それに対し上記の CPU 語長単位でやり取りするコンピュータをワードマシンと呼びます。バスやメモリ回路が特殊になるため最近では殆ど見ることはできません。

2 進法の計算

基本的にコンピュータは先の加算機だけですべての計算を行います。より高速に計算させるために、特別に計算だけをする回路や別部品（FPU）を組み込んだものも一般的になっています。
では、どうやって四則演算を行うのか、概要は以下の通りです。

- 引き算
  引く数を負にする。たとえばNOT回路を使って、マイナスに変換してから引かれる数と加算。
  2 進数における負の表記は後述。

- 掛け算
  桁送り機能があれば、一桁左へ移動させれば 2 倍となります。よって 3 倍なら元の数を左へ1桁分ずらした後、元の数を加算します。この桁送りの事をシフトと呼びます。

- 割り算
  掛け算とほぼ同じですが、右へのシフトを行います。
  コンピュータの演算にはここに問題があります。整数は正しく表現できますが、少数は正しく表現できません。実はコンピュータで正しく 0.1 は表現できないのです。

データ型

コンピュータは情報をすべて 2 進数の並び(ビット列)で表しますが、数字だけでなく文字や絵、音楽といったものまで扱う事ができます。ビット列にデータの属性を割り当てたものをデータ型といい、プログラミング言語で割り当て(宣言、Declare)します。
以下に主なデータ型と、実際のビット列の表現方法を解説します。

整数(Integer)

最も基本的なデータ型で、ビット列をそのまま 10 進数として扱います。たとえば 4bit アキュムレータでは、0101(b)は 5 というように割り当てます。正の数だけを扱うものは特に「符号なし整数(unsigned integer)」と呼びます。
符号なし整数の最大値は、$2^{(\text{アキュムレータの 2 進桁数})} - 1$ となります。例えば 4bit なら $2^4 - 1 = 15$ です。

負の数を表すには、符号を意味するビットを用意し、0 を正(+)、1 を負(-)というように割り当てます。非常に簡単で、0 111(b)は 7、1 011(b)は-3 となります。しかしこの方法では、マイナスゼロとプラスゼロという数字が発生してしまいます。
今日のコンピュータでは負の数は「2 の補数(Two's Complement)」と呼ばれる表記を用います。求め方は簡単で、以下のようになります。

1 の補数	ビット列の反転(not)
2 の補数	(1 の補数) + 1

例えば、-5 は、0101(5) —1 の補数→ 1010 —2 の補数→ 1011(-5)となります。よって 7-5 は加算でよく 7 + (-5) = 2 です。

```
    0111(b) 7
+)  1011(b) -5
--------------------
    0010(b) 2
```

表 9:4bit アキュムレータで表現できる整数

ビット列	符号なし	2 の補数	ビット列	符号なし	2 の補数
0000	0	0	1000	8	-8
0001	1	1	1001	9	-7
0010	2	2	1010	10	-6
0011	3	3	1011	11	-5
0100	4	4	1100	12	-4
0101	5	5	1101	13	-3
0110	6	6	1110	14	-2
0111	7	7	1111	15	-1

COBOL で採用されている BCD(Binary Corded Decimal、2 進化 10 進符号)では、ビット列のうち 0~9 だけを使用し 10 進数として扱います。処理速度とメモリ使用効率は低くなりますが、正確に 10 進数を扱う事ができます。

例) 14 → 0001 0100 (BCD)

また実際のデータをメモリへ配置する際に、桁の上位から格納するものをビッグ・エンディアン(big endian)、下位からは位置するものをリトル・エンディアン(little endian)と呼びます。この順序の事をバイトオーダー(Byte order)と呼びます。
例えば '12345678(h)'をメモリに配置する場合、ビッグ・エンディアンならそのままの 12, 34, 56, 78 と配置し、リトル・エンディアンは 78, 56, 34, 12 と配置されます。
ネットワーク転送(TCP/IP)では、ビッグ・エンディアンで行われます。

文字(character)

文字もコンピュータの中ではビット列として処理されます。文字の一覧に連番をふり、画面に表示する時に対応する画像に変換しています。この時、文字の一覧表を「文字コード（表） Character set」と呼びます。

文字コードで最も基本となるのは、米国の ASCII(American Standard Code for Information Interchange)で、英字（大文字・小文字）、数字、記号および[Delete][Enter]などの機能文字を含みます。たとえば A は 65 番(31h)、9 は 57 番(39h)となります。この表を格納するには 7bit 必要で、8bit 目は通信エラーを検出するためのパリティとして利用されています。
IBM をはじめ汎用機（レガシーシステム、General purpose computer, Legacy system）では EBCDIC（Extended Binary Coded Decimal Interchange Code）が採用されています。ASCII と違い非連続で、並び順も異なります。たとえば A は 193 番、9 は 240 番で、8bit 必要となっています。

英語圏では上記のように 1Byte あれば必要な文字を表現できたのですが、日本語では圧倒的に数が不足しています。そこで JIS が日本語の文字コードを制定し、それが JIS 漢字コード(情報交換用漢字符号 JIS X 0208 1990 年)と呼ばれます。常用漢字、地名・人名用など第 1 水準から第 4 水準まで分類されています。収容されている文字は 11,233 字で、2Byte で表現されています。
この事から、英語を扱う事を Single Byte、漢字など 2Byte で処理することを Double Byte と呼ぶことがあります。

日本語処理では、漢字はいわゆる倍角文字を扱い、半角は ASCII や JIS X0201「7 ビット及び 8 ビットの情報交換用符号化文字集合」を用いるため、文字の途中で文字コードを切り替えるための特殊な文字を挿入する必要があります。これをエスケープシーケンスと呼びます。

まだ PC が貧弱だったころ、このエスケープシーケンスを節約するため、半角文字で利用されていない連番に漢字を紐づけるよう、漢字表を押しこんだ Shift JIS がマイクロソフトや ASCII などによって提唱され PC の世界でデファクト・スタンダードとなりました。

一方、1980 年代中ごろに UNIX(高性能ワークステーション)用コードとして EUC(Extended Unix Code)が日本語 UNIX システム諮問委員会により制定されました。もともと日本語用でしたが、他言語に対応できるよう EUC-JP として体系化されています。このコードは半角と倍角が混在可能で、倍角文字は全て 8bit 目が 1 になるよう JIS 漢字を再配置しています。

Internet が普及し世界中の人々がコンピュータを利用するようになると、世界中の言葉を網羅した文字コードが必要になりました。そこで新たに Unicode が制定されました。異字体(サロゲート)や、エンコードの方法も複数あり複雑な規格となっています。
2000 年中ごろから普及し、現在では Windows, Linux, Mac OS で扱えるようになっています。プログラム言語 Java でも使われています。なお日本語では UTF-8 と呼ばれるエンコードが使われており、1 文字を表すのに 3Byte が必要です。

漢字コード例 「亜」の通番
3012h（JIS） B0A1h(EUC-JP) 889Fh(Shift JIS) E4BA9Ch(UTF-8)

浮動小数点(Floating point)

実数を扱う場合、コンピュータが計算できる桁数が有限であることから、指数表記を用いています。
指数表記は、仮数、基数、指数からなります。以下の場合、仮数は 1.23、基数は 10、指数は 9 です。

1,230,000,000　　→ 1.23 × 10^9

コンピュータでは、基数は 2 となり、指数部は符号ビットと値から構成されます。一般的には IEEE754 が用いられます。

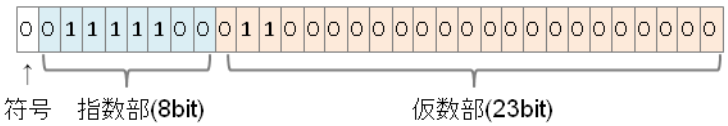


図 11:IEEE754 不動小数点フォーマット

- 符号
  仮数部の符号を表し、0は正、1は負となります。

- 指数部
  指数を表します、基数は 2 で、127 のバイアス表記（下駄ばき）となります。
  図の例では 1111100(b) = 124 で、127 のバイアスがあるので値としては
  124 – 127 = -3
  となります。

- 仮数部
  仮数は 10 進数で 1.xx になるよう調整したうえ、その少数部(.xx)を 2 進数で表します。
  図の例で仮数部は 01100～0(b) = 0.25 + 0.125 = 0.375 ですので、仮数は 1.375 となります。

  結果として図の値は、$1.375 \times 2^{-3} = 1.375 \div 8 = 0.171875$ となります。

不動小数点のデータ長さは精度とよばれ、上記の 32bit 版は「単精度実数」と呼ばれます。「倍精度実数」は符号 1bit、指数部が 11bit、仮数部が 52bit の合計 64bit。「4 倍精度」は符号 1bit、指数部が 15bit、仮数部が 112bit の合計 128bit であらわされます。

また IEEE754 独自の値として、指数部が全て1かつ、仮数部が0の場合は無限大、0 以外の場合は非数（NaN , Not a Number）があります。

IEEE754 以外では、IBM 独自の Excess-64、旧 DEC の VAX Float などがあります。