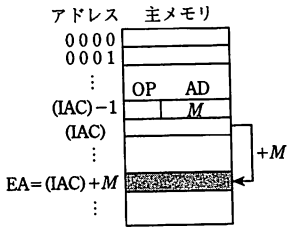
(d) 相対アドレス指定 (relative addressing) 図 2.1.9 に示すように，命令アドレスカウンタ IAC の内容を基準値とし，この内容とアドレス部が示す変位 (displacement) の値を加算して得られた結果が実効アドレスとなる方法を**相対アドレス指定方式**という．ここで，実行中の命令のメモリアドレスを計数するカウンタを命令アドレスカウンタという．なお，現在の命令アドレスカウンタの内容が (IAC) のとき，実行中の命令は (IAC) −1 に記憶されていることに注意されたい．すなわち，命令が読み出されると IAC の値は常に次の命令が入っているアドレスを示す．相対アドレス指定によりアドレス部の長さが短くてよく，メモリのどの部分を使って記憶してもよいプログラムを作成できる．



IAC は命令アドレスカウンタの略．
(IAC) は命令アドレスカウンタの内容（次に実行される命令のアドレス）を示す．

図 2.1.9: 相対アドレス指定

(e) インデックスアドレス指定 (index addressing) 図 2.1.10 に示すようにアドレス部を 2 つに分割し，**インデックスレジスタ** (index register) とよばれるレジスタの番号 I を示す部分と，アドレス M を示す部分に分ける．実効アドレスがインデックスレジスタ I の内容 (I) とアドレス M の和の値となるような方式を**インデックスアドレス指定方式**という．このとき，通常この

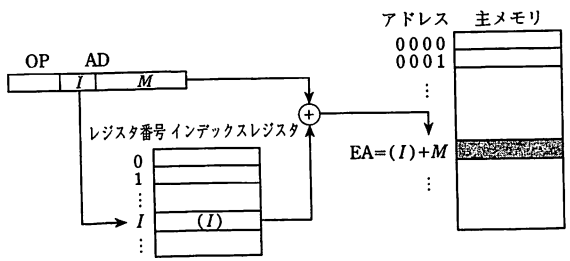


図 2.1.10: インデックスアドレス指定

命令が実行されるごとにインデックスレジスタの内容は自動的に +1 （または −1）される．M を定数とし，アドレス M から始まる一連のメモリ領域のデータにアクセスするようなループ処理に用いられる．間接アドレス方式の一つと考えることもできる．

(f) ベースアドレス指定 (base addressing) [6] 図 2.1.11 に示すようにアドレス部分を 2 つに分割し，**ベースレジスタ** (base register) とよばれるレジスタの番号 B を示す部分とアドレス M を示す部分に分ける．実効アドレスがベースレジスタ B の内容 (B) とアドレス M との和となる方式を**ベースアドレス指定方式**という．ここで，ベースアドレスは**基準値**を，アドレス部は**変位**を示すから，相対アドレス指定方式の一つと考えることもできる．ただし，ベースレジスタの内容はプログラムにより自由に変えることができる．例えば，多数のユーザプログラムを主メモリ上の任意の位置に配置し，ベースアドレスの内容は主メモリ上にロードする先頭番地とすればよく，アドレス部の変更は不用である．ベースレジスタのレジスタ長は主メモリのアドレス空間の任意の位置を指定できる長さをもつ．

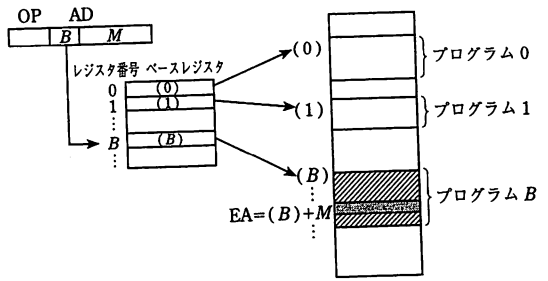


図 2.1.11: ベースアドレス指定

以上述べたアドレス指定方式はいくつか組み合わせて用いられることも多い．

2.2 コンピュータアーキテクチャ (2) データ形式

コンピュータ内部のデータは 2 値のメモリ素子を用いて記憶される．後で述べるように，演算もまた 2 値の論理回路を用いて実行される．ここでは，主として数値データの 2 進数表現について述べる．

[6] インデックスアドレス指定とベースアドレス指定をインデックス修飾とよぶことがある．

2.2.1 2進数と表記法

(1) r 進法

数 A を r 進数で

$$A = (a_l a_{l-1} \cdots a_1 a_0 . a_{-1} \cdots a_{-m})_r \tag{2.2.1}$$

と表すとき，r 進数の数値 A は次式で与えられる．

$$\begin{aligned} A &= a_l r^l + a_{l-1} r^{l-1} + \cdots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \cdots + a_{-m} r^{-m} \\ &= \sum_{i=-m}^{l} a_i r^i \qquad (a_i \in \{0, 1, 2, \cdots, r-1\}) \end{aligned} \tag{2.2.2}$$

ここで，r を**基数**（radix, base）とよぶ．

(2) 基数の選定

まず，r 進数 1 桁を記憶するために r 個の素子が必要であると仮定しよう．r 進数 k 桁を表すために必要なメモリ素子数 L は

$$L = kr \tag{2.2.3}$$

で与えられる．また，r 進数 k 桁で表しうる数値パターンの総数 M は

$$M = r^k \tag{2.2.4}$$

である．式 (2.2.3)，(2.2.4) より

$$L = r \log_e M / \log_e r \tag{2.2.5}$$

となる．M を一定とし，素子数 L を最小とする基数 r を求めると，$\dfrac{\partial L}{\partial r} = 0$ を解いて

$$r = e \fallingdotseq 2.718 \tag{2.2.6}$$

となる．基数 r は整数値をとるから，最適な基数は $r = 3$ である（演習問題 [2.2] 参照）．

一方，実現されているメモリ素子あるいは論理素子のほとんどは 2 つの安定状態をもつ 2 値素子である．回路設計上 2 値論理はブール代数（Boolean algebra）と整合し，また 2 値素子は多値素子と比べ実現しやすいこと，耐雑音性に優れているなどの特徴をもつ．したがって，2 安定素子を使うかぎり r 進数 1 桁は

$$p = \lceil \log_2 r \rceil \tag{2.2.7}$$

を満たす p 個の 2 安定素子で表現できる．ただし，$\lceil x \rceil$ は x 以上の最小の整数である．この結果，式 (2.2.3) は 2 値素子を用いて $L = \lceil \log_2 r \rceil k$ となる．以上より，2 進数（一般的には 2^p 進数）の優れていることがわかる[7]．

(3) 基数の変換

2 進法はコンピュータに好適であるが，日常では 10 進法が使われるため 2 進–10 進変換は頻繁に行われる．また，2^p 進数は 2 進法との変換が容易であり，2 進数の表記法として用いられる（表 2.2.1 参照）．以下では基数の変換について述べる．

(a) s 進–r 進変換 一般に s 進数 B を r 進数 A に変換することを考える．r 進数 A は式 (2.2.1)，(2.2.2) のように表されているとする．すなわち

$$\begin{aligned} (B)_s &= (B_I . B_F)_s \\ &= (A)_r = \sum_{i=-m}^{l} a_i r^i \qquad (a_l \neq 0,\ a_{-m} \neq 0) \end{aligned} \tag{2.2.8}$$

である．ここで，B_I は B の整数部，B_F は B の小数部を示す．

式 (2.2.8) を次のように変形する．

$$\begin{aligned} (A)_r = r(r(\cdots(r(ra_l + a_{l-1}) + a_{l-2}) \cdots + a_2) + a_1) + a_0 \\ + (a_{-1} + (a_{-2} + \cdots (a_{-m+2} + (a_{-m+1} + a_{-m} r^{-1}) r^{-1}) \cdots) r^{-1}) r^{-1} \end{aligned} \tag{2.2.9}$$

この式より，容易に次の漸化式を得る．

$$\begin{aligned} A_i &= r A_{i+1} + a_i \qquad (i = 0,\ 1, \cdots,\ l) \\ r A_{-i} &= A_{-i-1} + a_{-i} \qquad (i = 1,\ 2, \cdots,\ m) \end{aligned} \tag{2.2.10}$$

ここで，$A_0 = B_I$，$A_{-1} = B_F$ であり，演算は s 進法で行う．また，r 進数 A の整数部が $l+1$ 桁，小数部が m 桁のとき，$a_j = 0$ $(j \geq l+1, j \leq -m-1)$ である．

式 (2.2.10) は次のような計算を意味している．整数部については A_i を r で割った剰余を a_i，商を A_{i+1} と順次求めていく．$a_i \in \{0, 1, \cdots, r-1\}$ であるから，必ず有限桁の B_I は有限桁の r 進数でおきかえられる．一方，小数部については A_{-i} に r を乗じた数の整数部を a_{-i}，小数部を A_{-i-1} と順次求めていく．$A_{-i-1} < 1$ であることに注意する．このとき，B_F が有限桁のとき s^{-1} が r 進数

[7] 2 値論理の他，多値論理も研究されている．また，ファジィ論理も実用化されている．

で無限小数でないとすれば，式 (2.2.10) より $rA_{-m} = a_{-m} \in \{1, 2, \cdots, r-1\}$ のときに限り，r 進数 m 桁の有限桁に変換される．

(b) 2 進–10 進変換

(i) 10 進 → 2 進変換 $r = 2$，$s = 10$ とおくと，数値 B の値は

$$(B)_{10} = a_l 2^l + a_{l-1} 2^{l-1} + \cdots + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + \cdots + a_{-m} 2^{-m} \quad (a_i \in \{0, 1\}) \quad (2.2.11)$$

である．10 進数を 2 進数に変換するとき，以下の 10 進数演算を行う．整数部 $(B_I)_{10}$ は

$$(B_I)_{10} = a_l 2^l + \cdots + a_1 2^1 + a_0 2^0 = A_0 \quad (2.2.12)$$

と表せるから，これを 2 で割り

$$A_0 = 2(a_l 2^{l-1} + \cdots + a_1 2^0) + a_0 = 2A_1 + a_0 \quad (2.2.13)$$

を得る．剰余 a_0 をとった商 A_1 をまた 2 で割り，$2A_2 + a_1$ から a_1 を得る．順次 A_i を 2 で割る操作をくり返して 2 進数の各桁の数 a_i が求まる．一方，小数部 $(B_F)_{10}$ は

$$(B_F)_{10} = a_{-1} 2^{-1} + a_{-2} 2^{-2} + \cdots + a_{-m} 2^{-m} = A_{-1} \quad (2.2.14)$$

であるから，これに 2 を乗じ

$$2A_{-1} = a_{-1} + (a_{-2} 2^{-1} + \cdots + a_{-m} 2^{-m+1}) = a_{-1} + A_{-2} \quad (2.2.15)$$

を得る．ここで，$A_{-2} < 1$ より a_{-1} は $2A_{-1}$ の整数部，A_{-2} は $2A_{-1}$ の小数部である．a_{-1} をとった A_{-2} にまた 2 を乗じ，$a_{-2} + A_{-3}$ から a_{-2} を得，順次 A_{-i} に 2 を乗ずる操作をくり返して a_{-i} が求まる．

［例 **2.2.1**］ 10 進 →2 進変換の例
$(13.7)_{10}$ の 2 進表示は図 2.2.1 のようにして求められる．整数部は 1101，小数部は $0.1\dot{0}11\dot{0}$ となり，$(13.7)_{10} = (1101.1\dot{0}11\dot{0})_2$ と変換される[8]． □

(ii) 2 進 →10 進変換 式 (2.2.1)，(2.2.2) で $r = 2$ とおけば，式 (2.2.11) に直接代入して容易に求められる．もちろん，(a) で述べた方法に，$s = 10$，$r = 2$ とし 2 進演算を行っても求められる．

［例 **2.2.2**］ (2 進 →10 進変換の例)
$(11011.1011)_2$ の 10 進表示は $(11011)_2 = 16 + 8 + 2 + 1 = (27)_{10}$，$(0.1011)_2 = 0.5 + 0.125 + 0.0625 = (0.6875)_{10}$ から $(11011.1011)_2 = (27.6875)_{10}$ を得る． □

[8] $\dot{\ }$ と $\dot{\ }$ の間は循環節であることを示す．たとえば，$0.\dot{3} = 0.33\cdots$，$0.\dot{1}23\dot{4} = 0.12341234\cdots$，$1.05\dot{6} = 1.0565656\cdots$ である．

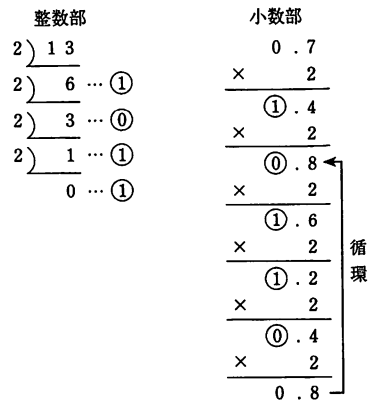


図 2.2.1: 10 進 →2 進変換の例

表 2.2.1 に 10 進-2^p 進数の対応表を示す．

表 2.2.1: 10 進数 -2^p 進数の対応

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	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

2.2.2 データ形式

(1) 数値データ

コンピュータ内部の数値データは 2 値表現を基本とし，2 進法・8 進法・10 進法・16 進法などの表現法がとられる．これらはいずれも式 (2.2.1) が式 (2.2.2) で計算された値を示しており，**位取り記数法** (positional notation) という．アラビア数字によるこの表記法は大きな整数（あるいは小さな小数）を短い長さで表現でき，また四則演算にも適する．ローマ数字などと比べればその優れた性質が理解できる．

(a) 固定小数点データ 固定小数点（fixed point）データは小数点の位置を固定したものである．小数点の位置を最下位ビット（Least Significant Bit : **LSB**）[9] の右に置いて整数を表す．最上位ビットを符号ビットとして用い，正を 0，負を 1 で表す．その結果，固定小数点表示は図 2.2.2 のようになる．なお，最上位ビット（Most Significant Bit : **MSB**）の右に小数点を置き小数を表すものもある．

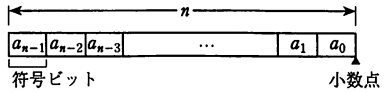


図 2.2.2: 固定小数点データ

固定小数点データの負数の表示法には次の 3 通りがある．

① **符号付き絶対値**（signed magnitude）**表示** ··· MSB を符号ビットとし，残りの $n-1$ ビットで 2 進絶対値を表す．

② **補数**（complement）**表示** ··· 一般に，式 (2.2.1) の r 進数 A の r の**補数** A''，$(r-1)$ の**補数** A' を次式で与える[10]．

$$A'' = r^{l+1} - A \tag{2.2.16}$$

$$A' = (r^{l+1} - r^{-m}) - A \tag{2.2.17}$$

さて，簡単のため n 桁の 2 進正整数 $A = (a_{n-1}, a_{n-2}, \cdots, a_0)_2$ に対する**補数表示**を考える．ただし，a_{n-1} は符号桁である．**2 の補数** A'' と **1 の補数** A' は次式で与えられる．

$$A'' = 2^n - A \tag{2.2.18}$$

$$A' = (2^n - 1) - A \tag{2.2.19}$$

ここで

$$\begin{aligned} A' &= \overbrace{(11\cdots1)}^{n}{}_2 - (a_{n-1}a_{n-2}\cdots a_0)_2 \\ &= \sum_{i=0}^{n-1}(1-a_i)2^i \end{aligned} \tag{2.2.20}$$

[9] 一般に最上位桁を **MSD**（Most Significant Digit），最下位桁を **LSD**（Least Significant Digit）という．

[10] 式 (2.2.1) より整数部は $l+1$ 桁，小数部は m 桁あることに注意のこと．

であるから，1 の補数は符号桁も含め各桁 a_i の 0 と 1 を入れかえればよく，2 の補数は 1 の補数に 1 を加えたものとなる．この方法はまた，式 (2.2.16)，(2.2.17) により，小数点をもつ数に対しても同様に適用できる．

［**例 2.2.3**］ 2 進負数の表示方法の例を表 2.2.2 に示す． □

表 2.2.2: 負数の表示例

A		$-A$		
10 進正数	2 進正数	符号付き絶対値表示	2 の補数表示	1 の補数表示
1	0001	1001	1111	1110
7	0111	1111	1001	1000
0.125	0.001	1.001	1.111	1.110
0.625	0.101	1.101	1.011	1.010

③ **エクセス $2^{\nu-1}$ 表示** ··· あらかじめ $2^{\nu-1}$ を加えて表示する**エクセス $2^{\nu-1}$**（excess $2^{\nu-1}$）**表示**（またはバイアス（bias）表示）とよばれる符号も負数を表すことができる．一般に，$-2^{\nu-1}$～$2^{\nu-1}-1$ を 0～$2^{\nu}-1$ にそれぞれ対応させる．浮動小数点データの指数部に用いられる．

［**例 2.2.4**］ $\nu = 8$，すなわちエクセス 128 表示の例を表 2.2.3 に示す．この場合，MSB は符号桁であるが，0 が負，1 が正を示す． □

表 2.2.3: エクセス 128 表示

10 進数	10 進数エクセス 128	エクセス 128 表示
127	127 + 128 = 255	1 1 1 1 1 1 1 1
126	126 + 128 = 254	1 1 1 1 1 1 1 0
	⋮	⋮
2	2 + 128 = 130	1 0 0 0 0 0 1 0
1	1 + 128 = 129	1 0 0 0 0 0 0 1
0	0 + 128 = 128	1 0 0 0 0 0 0 0
−1	−1 + 128 = 127	0 1 1 1 1 1 1 1
−2	−2 + 128 = 126	0 1 1 1 1 1 1 0
	⋮	⋮
−126	−126 + 128 = 2	0 0 0 0 0 0 1 0
−127	−127 + 128 = 1	0 0 0 0 0 0 0 1
−128	−128 + 128 = 0	0 0 0 0 0 0 0 0

以上の①～③の表示法の例を例 2.2.5 に示す.

[例 **2.2.5**] ($n=\nu=5$ の正負整数) −16～15 の表示を表 2.2.4 に示す. □

表 2.2.4: 2 進数の負数の表示例

10 進数	符号付き絶対値	1 の補数	2 の補数	エクセス 16
15	01111	01111	01111	11111
14	01110	01110	01110	11110
13	01101	01101	01101	11101
⋮	⋮	⋮	⋮	⋮
4	00100	00100	00100	10100
3	00011	00011	00011	10011
2	00010	00010	00010	10010
1	00001	00001	00001	10001
0	00000	00000	00000	10000
−0	10000	11111	—	—
−1	10001	11110	11111	01111
−2	10010	11101	11110	01110
−3	10011	11100	11101	01101
−4	10100	11011	11100	01100
⋮	⋮	⋮	⋮	⋮
−13	11101	10010	10011	00011
−14	11110	10001	10010	00010
−15	11111	10000	10001	00001
−16	—	—	10000	00000

(b) 浮動小数点データ 科学技術計算では実数演算が多く，**浮動小数点** (floating point) データが用いられる．一般に，$A = M \times B^E$ の形の表示方法を**浮動小数点表示**とよぶ．ここで，M は**仮数** (mantissa)，B は**基数** (base, radix)，E は**指数** (exponent) である．通常，仮数は符号付き絶対数，1 の補数，または 2 の補数表示が用いられ，指数はエクセス $2^{\nu-1}$ 表示，基数は 10 または 2 のべき乗が用いられる[11]．一般に，データが n [ビット]，このうち指数が ν [ビット] のとき (図 2.2.4 参照)，表示し得る浮動小数点データ A は仮数部を 2 の補数表示とすると

[11] 通常，$(A)_r = M \times r^E$ $(r^{-1} \le M < 1)$ すなわち，$B = r$ に選ぶ.

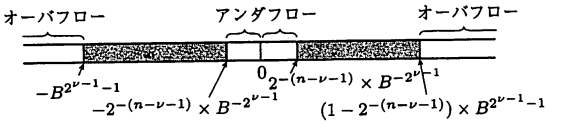
$$
\begin{aligned}
&-B^{2^{\nu-1}-1} \le A \le -2^{-(n-\nu-1)} \times B^{-2^{\nu-1}} \\
&A = 0 \\
&2^{-(n-\nu-1)} \times B^{-2^{\nu-1}} \le A \le (1-2^{-(n-\nu-1)}) \times B^{2^{\nu-1}-1}
\end{aligned}
\qquad (2.2.21)
$$

で与えられる．したがって，図 2.2.3 のように表示し得る数値 A は限られ，同一の数値をもつものを含め，$2^{n-\nu} \times B^{2^\nu}$ 個しか存在しない.

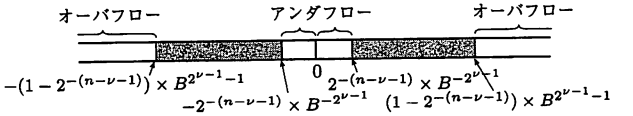
なお，仮数が 1 の補数表示の場合は絶対値表示と同じで，表現し得る A の仮数 M の値は

$$
2^{-(n-\nu-1)} \le |M| \le 1-2^{-(n-\nu-1)}, \quad M = 0 \qquad (2.2.22)
$$

となる．以上より，有限桁で表し得る数値は離散的にしか存在しない．このため演算上，次の点を考慮しなければならない.



(a) 仮数が 2 の補数表示の場合



(b) 仮数が 1 の補数または符号付き絶対値表示の場合

図 2.2.3: 浮動小数点で表現し得る範囲

① **正規化** (normalization) ··· 同一数値に対する複数の表現方法の中から，指数部が最小となるように浮動小数点表示をする．このことを正規化という．すなわち，1 の補数表現のとき，$B^{-1} \le |M| < 1$，および $M = 0$ とするように指数部 E を定める.

② **オーバフロー** (overflow) ··· 図 2.2.3 に示すように，2 つの数の演算結果が表現し得る範囲より大きな絶対値をもつ数値になることをオーバフローという.

③ **アンダフロー** (underflow) ··· 図 2.2.3 に示すように 2 つの数の演算結果が表現し得る範囲より小さな絶対値をもつ数値になることをアンダフローという.

④ **丸め誤差**（rounding error）··· 10進から2進変換などの際，仮数部最小桁以下に誤差を生ずることがある．これを丸め誤差という．誤差を小さくするために2語で一つのデータを表す倍精度が用いられる．このとき通常，仮数部の有効桁が増加する．

その他，仮数部が有効桁のため大きな数値と小さな数値との加減算などの際，0に近い数の有効桁が脱落することがある．これを**桁落ち**という．このため，通常 $A=0$ のときは，$M=0$，$E=0$ で表現する．

［例 **2.2.6**］ IEEE標準形式

図2.2.4にIEEE標準形式を示す．S は符号（$+:0$，$-:1$），M は仮数（絶対値表示），E は指数（エクセス127表示）である．32ビット形式のとき数値 A は

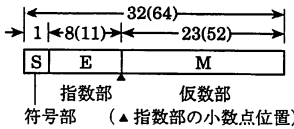


図 2.2.4: 32ビット（64ビット）のIEEE標準形式

$$A=\begin{cases}(-1)^S(1.M)\times 2^{E-127} & (0<E<255)\\ (-1)^S(0.M)\times 2^{-126} & (E=0,\ M\neq 0)\\ (-1)^S\infty & (E=255,\ M=0)\\ (-1)^S 0 & (E=0,\ M=0)\\ \text{非数} & (E=255,\ M\neq 0)\end{cases}\qquad (2.2.23)$$

とする．なお，上式で $(1.M)$ としたのは正規化 $1\le|M'|<2$ $(B=r=2)$ において，仮数 M' のMSBは必ず1になるのでこれを省略することができるからである．M' のMSBを省略したものが M である．これを**けち表現形式**（economical form）といい，$(1.M)$ の1は省略したMSBで，これを**隠しビット**（hidden bit）という．例えば，$(1.6275)_{10}=(1.1011)_2$ から仮数部 $M'=11011$ でMSBは必ず1である．したがって，これを省略し $M=1011$ とすることができる．これより，例えば $A=(-5.5)_{10}=(-101.1)_2$ のとき，$(101.1)_2=(1.011\times 2^2)$ より $E=(127+2)_{10}$ $M'=1011$ である．これをIEEE標準形式に変換すると

S	E	M
1	10000001	$\underbrace{0110\cdots0}_{23\text{ビット}}$

となる． □

(c) 10進数 2進 m 桁を使って10進1桁を表す符号を**10進符号**（decimal code）という．表2.2.5に示すように，いく通りかの表現方法がある．ここで

$$R=\frac{\log_2 10}{m}\qquad (2.2.24)$$

を10進数の**符号化比率**とよぶ．表2.2.5のうち**2進化10進**（Binary Coded Decimal：BCD）符号が最もよく用いられる．特に事務計算では10進数のまま記憶したり演算することもある．また，文字としてデータを転送することもしばしば行われる．図2.2.5のように，1［バイト］に10進2桁を表示する方法を**パック**（pack）**形式**という．一方，次で述べる文字データとして，1［バイト］に10進1桁を表示する方式を**アンパック**（unpack）**形式**という．パック形式では，$R=\log_2 10/4\fallingdotseq 0.83$ であり，アンパック形式では，$R=0.415$ である．当然ながら前者の方が効率が良い．事務計算ではデータが数値だけの場合，パック形式に変換する．このため両形式間の変換命令をもつ．

表 2.2.5: 10進符号

名称 / 重み / 10進数	2進化10進符号 8421	エクセス3符号 なし	5者択2符号 74210	交番2進符号 なし	偶数パリティ検査符号 8421P
0	0000	0011	11000*	0000	00000
1	0001	0100	00011	0001	00011
2	0010	0101	00101	0011	00101
3	0011	0110	00110	0010	00110
4	0100	0111	01001	0110	01001
5	0101	1000	01010	0111	01010
6	0110	1001	01100	0101	01100
7	0111	1010	10001	0100	01111
8	1000	1011	10010	1100	10001
9	1001	1100	10100	1101	10010

*重み付けは例外

表2.2.5に示したエクセス3符号は各桁の0と1を入れかえると互いに9の補数となる．また，10進加算回路（演習問題［3.8］参照）が簡単で，必ず1のビットをもつという利点がある．**5者択2**（2 out of 5）**符号**は5［ビット］を用いて必ず1が2［ビット］あるような構成になっている．**交番2進符号**（reflected binary code, Gray code）は，アナログ量をディジタル量にかえる，いわゆるA–D変換によく用いられる．となり合う数の符号が1［ビット］だ

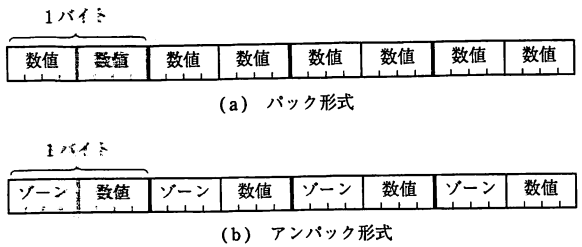


図 2.2.5: 10 進数の表現

け異なる．**偶数パリティ検査**（even parity check）符号は例えば 4［ビット］の 10 進符号に 1［ビット］を付加し，5［ビット］中 1 であるビットの数が偶数であるように構成する．この符号や 5 者択 2 符号では奇数個のビット誤りに対する検査能力をもつ．

純 2 進（pure binary）数は $R = 1$ であり最も効率が良い．しかし，$R < 1$ とすればとり得ないパターンが存在し誤り検出能力が生じる．$1 - R$ を**冗長度**（redundancy）とよび，一般に冗長性が増すに従い誤り検出能力を増すことができる．

(2) **非数値データ**

(a) **文字データ** ビットパターンを用いてコンピュータ内部で文字を表現したものを**文字符号**（character code）といい，文字符号で構成されたデータを**文字データ**という．現在では，ほとんど 1 文字を 8［ビット］，すなわち 1［バイト］で表現する 8［ビット］符号が用いられている．最も広く知られている符号は，国際標準化機構（International Organization for Standardization : ISO）により制定された 7［ビット］で構成された ISO 646 符号である．データ伝送用として 1［ビット］のパリティビットを付加して用いられることがある．**ASCII**（American Standard Code for Information Interchange）符号は，これを 8［ビット］に拡張したアメリカ国内規格符号である．日本においては ISO 646 符号に準拠して制定されたカナ文字を含む 8［ビット］の情報交換用符号 (JIS C6220) が広く使われている．JIS C6220 では，例えば

“A”→ 0 1 0 0 0 0 0 1　　“カ”→ 1 0 1 1 0 1 1 0

“2” → 0 0 1 1 0 0 1 0　　“&” → 0 0 1 0 0 1 1 0

のように，192 個の英文字，カナ文字記号，機能コードのビットパターンが定

められている．このほか，従来の 6 ビット符号を拡張した IBM 社で用いられている EBCDIC（Extended Binary Coded Decimal Interchange Code）がある．

(b) 論理データ ビットごとに論理的な意味をもたせたものを論理データとよび，2 値の論理変数を表す．AND，OR，NOT，EXCLUSIVE OR，COINCIDENCE（表 3.1.2 参照）などの論理演算を実行する．

(3) **固定長データと可変長データ**

命令語の場合と同様，データ語にも語を単位とする**固定長データ形式**と**可変長データ形式**とがある．例えば，固定小数点データ，浮動小数点データは通常固定長データである[12]．一方，事務計算におけるファイルデータにはレコード（record）とよばれる数値列や文字列があり，バイト単位に可変長で構成されるので可変長データとよばれる．

2.3 プロセッサアーキテクチャ

図 1.2.1 に示した演算装置・制御装置の所望機能をどのように実現するかという設計思想が**プロセッサアーキテクチャ**である．当然，ハードウェアとソフトウェアをどのように機能分担させるかでトレードオフが存在する．ハードウェアで実現すれば一般に高速となるが，専用化することになり融通性が減少する．逆に，ソフトウェアで実現すれば汎用性・融通性は増すが速度が落ちる．

ノイマンアーキテクチャではまず，メモリからマシン命令を読み出し・解読（**Fetch**）する．次に，メモリからオペランドを読み出し演算を実行し，メモリにその結果の書き込みを実行（**Execute**）する命令サイクルを繰り返す．1951 年，命令の実行を制御する信号を簡単な制御回路で発生させる**マイクロプログラム制御方式**が M.V.Wilks により提案され，EDSAC II（1958 年），IBM 360（1964 年）などに採用された．命令セットを自由に生成することができ，エミュレーションやファミリー化が可能となった．

一方，命令サイクルにおいて命令の**先回り制御方式**[13]が IBM Stretch コンピュータ（1961 年）に採用されてその後，マイクロプロセッサ R4000（1991 年）のスーパパイプライン方式[14]，Super-SPARC（1991 年）のスーパスカラ方式[15]が実用化された．

なお，プロッセサ（特に制御装置）の動作が理解できればコンピュータ全体を理解するするのはたやすい．それ程，制御アーキテクチャは重要である．

[12]固定長データの長さは，1 語の 1/2 倍，2 倍，4 倍などがあり，それぞれ半語長，倍（語）長，4 倍（語）長とよばれる．
[13]命令の実行中にメモリから次の命令をとり出す方式．
[14]さらに細かいステップで先読みをする方式．
[15]複数パイプラインにより複数の命令を同時に実行する方式．

2.3.1 演算アーキテクチャ

(1) 算術論理演算ユニット

算術論理演算ユニット（Arithmetic Logic Unit : ALU）をどのように構成するかを考える．ALU は通常，図 2.3.1 のように画く．2 語入力 1 語出力（n [ビット/語]）で，ALU の出力の一部（例えば，演算結果の正負符号，オーバフローフラグなど）が状態レジスタ（SR）に入力される．ALU は算術（四則）演算器，論理演算器，補数化器，（巡回）シフト器などで構成される．図 2.3.2 にハードウェアで実現した算術演算器を示す．固定小数点の演算では，右に記載した方法ほど専用のハードウェア演算器を必要とする．構成法は負数表示が符号付絶対値表示，1 または 2 の補数表示，エクセス w 表示のいずれかであるかにより異なる．浮動小数点演算は固定小数点演算器と正規化器を用いて実現できる．

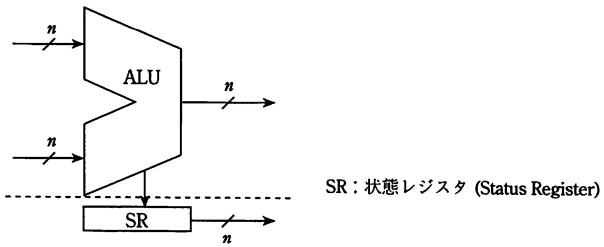


図 2.3.1: 算術論理演算ユニット

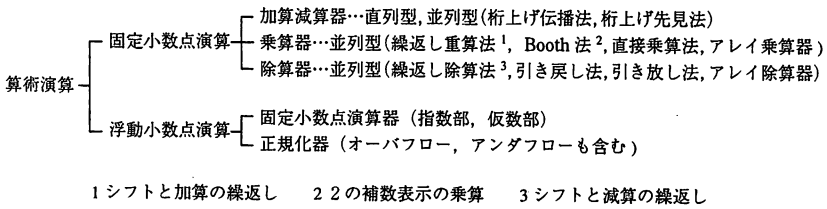


図 2.3.2: 算術演算器の実現方法

(2) 2 進算術演算

2 の補数表示による固定小数点演算法を示す．

(i) 加　算　表 2.3.1 に 2 進数 x と y 各 1 桁の加算規則を示す．ここで，s は和（sum），c は**桁上がり**（carry）である．この規則に従う回路を**半加算器**

（Half Adder : HA）とよぶ．半加算器では 2 進数 2 桁目以上の加算はできない．下の桁から桁上げ c_O を考慮した加算器を**全加算器**（Full Adder : FA）とよぶ．上の桁への桁上げを c_I とすると，演算規則（真理値表）は表 2.3.2 で与えられる．なお，小数点は 10 進演算と同様に保存される．

表 2.3.1: 2 進 1 桁の算術演算規則

		加算 $x+y$		減算 $x-y$		乗算
x	y	c	s	b	d	$x \cdot y$
0	0	0	0	0	0	0
0	1	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	0	0	0	1

表 2.3.2: 全加算器の真理値表

入力			出力	
c_I	x	y	c_O	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

［例 **2.3.1**］　加　算

10 進数	2 進数
5	0 0 1 0 1
+) 9	+) 0 1 0 0 1
1 4	0 1 1 1 0

(ii) 減　算　減算には 2 進直接減算法（表 2.3.1 参照）と**補数化加算法**がある．前者は減算器を必要とするが，10 進演算と同様である．後者は負数の補数表示により減算が加算器により実行される．

いま，$A, B > 0$ とし，A, B とも符号を含め 2 進 n 桁で表現されているとする．2 の補数表示を用いたとき，式 (2.2.18) より $B - A = B + (-A)$ であるから $-A$ を 2 の補数表示した A'' より

$$B - A = B + (2^n - A) - 2^n = B + A'' - 2^n \qquad (2.3.1)$$

となり，減算は 2^n の項を除き補数の加算におきかえられる．これは，次の例 2.3.2 で 2^n の項を無視することに相当する．なお，1 の補数の場合もほぼ同様であるが，**循環桁上げ**（end around carry）が必要である．

［例 **2.3.2**］ 減 算

10進数	2進数（2の補数）	2進数（1の補数）
10	0 1 0 1 0	0 1 0 1 0
+) (−3)	+) 1 1 1 0 1	+) 1 1 1 0 0
7	1 0 0 1 1 1	1 0 0 1 1 0
	桁上がり → 無視する	→ +) 1 → 0 0 1 1 1

なお，2進直接減算法は**半減算器**だけでは実行できない．加算器と同様**全減算器**が必要でその演算規則（真理値表）を表2.3.3に示す．ここで，$x-y$を計算するときb_I, b_Oは借り，dは差である．

表 2.3.3: 全減算器の真理値表

入力			出力	
b_I	x	y	b_O	d
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

(iii) 乗 算 表2.3.1に2進数xとy各1桁の乗算規則を示す．符号および小数点以下は10進演算と同様である．負数の補数表示による場合は，次のように符号部を分けて計算する．

いま，$A, B > 0$とし，A, Bとも符号を含め2進n桁で表現されているとする．2の補数表示を用いたとき，$C = B \times (-A)$は次のように計算される．$-A$の2の補数表示の符号1を除くと，$A'' - 2^{n-1}$となるから

$$C = B \times [(2^n - A) - 2^{n-1}] - B \times 2^{n-1} = -AB \qquad (2.3.2)$$

となり，$-B \times 2^{n-1}$を補正すればよい．

［例 **2.3.3**］ 乗 算

10進数	2進数	10進数	2進数（2の補数）
			符号桁 ↓
3	0 0 0 1 1	3	0 0 0 1 1
×) 5	×) 0 0 1 0 1	×) (−5)	×) 1 1 0 1 1
15	0 0 0 1 1	−15	0 0 0 1 1
	0 0 0 1 1		0 0 0 1 1
	0 0 0 0 0 0 1 1 1 1		+) 0 0 0 1 1
			0 0 1 0 0 0 0 1
			+) 1 1 1 0 1 …… 被乗数00011の2の補数
			1 1 1 1 1 0 0 0 1 ……(−15)

(iv) 除 算 2進数の除算は10進法と同様にできる．減算をくり返し，例2.3.4のように実行する．

なお，例2.3.1〜2.3.4では簡単のため2進数5［ビット］，内符号1［ビット］で示した．

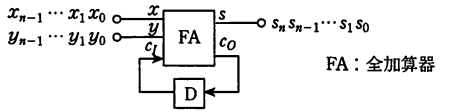
［例 **2.3.4**］ 除 算

10進数	2進数
4	1 0 0 ……商
3) 1 3	0 0 0 1 1) 0 1 1 0 1
1 2	0 1 1
1	0 0 0 0
	0 0 0 0
	0 0 0 0 1
	0 0 0 0 0
	0 0 0 0 1 ……余り

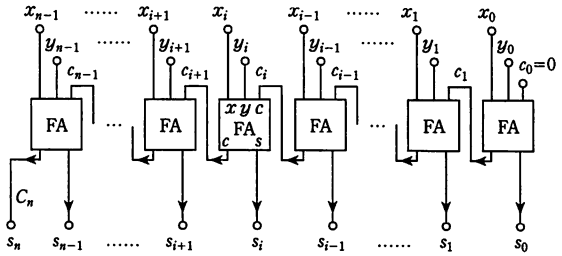
次に固定小数点加算器を例にとり，その具体的な実現方法を示す．図2.3.3に(a) **直列型加算器**，(b) **桁上げ伝播法**による並列型加算器，(c) **桁上げ先見法**による並列型加算器を示す（論理回路については3.1節参照）．図中，入力は$(x_0, x_1, \cdots, x_{n-1})$, $(y_0, y_1, \cdots, y_{n-1})$，出力は$(s_0, s_1, \cdots, s_n)$でありFAは全加算器，Dは遅延（記憶）を示す．

原理的には1［ビット］の(a)直列型加算器（と補数器，シフト器）があればソフトウェアにより四則演算が実行できる．しかし，それでは時間がかかりすぎるので並列化するのである[16]．

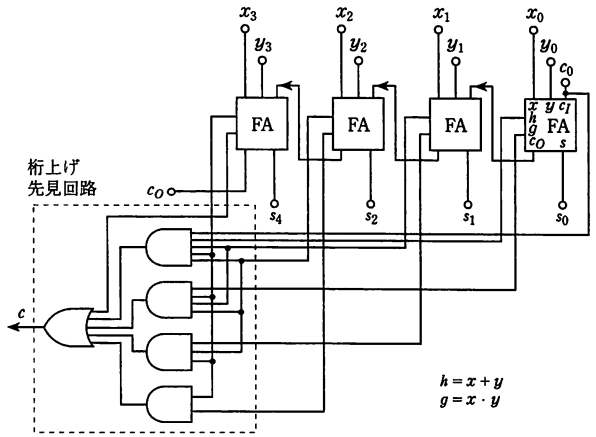
[16](b)では桁上げ伝播時間の遅れは許容する必要がある．



(a) 直列型加算器



(b) 桁上げ伝播法による並列型加算器



(c) 桁上げ先見法による並列型加算器(4ビット)

図 2.3.3: 固定小数点加算器の具体的実現方法

2.3.2 制御アーキテクチャ

(1) プロセッサの構成

プロセッサの単純化した構成例を図 2.3.4 に示す．A バス，B バス，C バスはデータ転送ルートで，アドレス信号・データ信号・制御信号・電源など各部に共通に配られる**共通信号母線**（通常，数十本／組）である[17]．これにより，データ語・命令語・制御信号などを運ぶ．例えば，Pentium II のシステムバスは 64［本／組］で構成されている．CPU をもう少し詳細に記述した例を図 2.3.5 に示す．

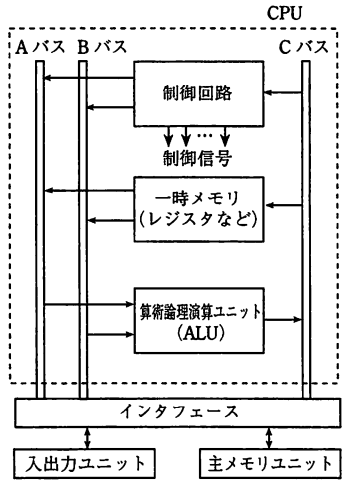


図 2.3.4: プロセッサ（CPU）の構成例

制御ユニットはメモリから読み出し（fetch）てきた命令を解読（decode）し，命令を実行（execute）するため各種制御信号を生成する．制御信号は解読された命令により，主として所定のゲートパルスを供給し，マイクロ動作に相当する処理を実行する．基本的には，制御ユニットは次のもので構成される．

[17]バス（bus）は情報の転送路でありハイウェイ（high-way）ともいう．m 個の装置を互いに結合させるには $\binom{m}{2}$ の信号線の組が必要である．例えば，キーボード，フロッピーディスク，プリンタ，マウス，ディスプレイの入出力装置と主メモリ，CD-ROM，ハードディスクの記憶装置と CPU を互いに結合させるには $\binom{9}{2}=36$ の信号線の組が必要となる．そこで，それぞれの装置に共通な信号線の組（装置によっては不用な信号線も含まれる）をひとたばにし各装置を 1 組の信号線に接続し必要に応じ入力（送信端）と出力（受信端）を切りかえる制御を行うと単純化・低コスト化と同時に拡張性をもたせることができる．これがバス（ハイウェイ）で入力・出力ゲートの制御（バスの使用権）は通常制御ユニットが担う．もちろん，バスは同時には 1 組の装置の転送路しか構成できないから転送速度は落ちる．

［2.3］ 次の10進数を2進数に変換して演算を実行せよ．ただし，レジスタ長は符号も含め10ビットとし，負数は2の補数をとるものとする．

(1) $98+105$ (2) $189-33$ (3) $77-210$ (4) $-16-38$

［2.4］ IEEE 標準形式 (例 2.2.6 参照) において表現しうる数値 A の範囲を示せ．ただし，基数 $B=2$，仮数は1の補数によるものとする．

［2.5］ 図 2.4.5 において，(1) 探索引数が 01011011 のとき，読出しデータは何か．(2) マスクレジスタが 01000101 のとき，読出しデータは何か．

［2.6］ 並列2進加算回路において，2進各桁の桁上り C_{i-1} を知って，高速に桁上げを行う方式について説明せよ．

3 ハードウェア

コンピュータアーキテクチャはひとりでその形態を作り，整えてきたわけではない．むしろ，ハードウェア技術がアーキテクチャを変え，応用を広げ，新しいニーズを生んできたという方が正しい[1]．例えば，コンピュータの世代にみられるように，真空管，トランジスタ，IC，LSI，VLSI とデバイス（素子）の発展がさまざまなアーキテクチャを作り応用分野を広げてきた．現在では，VLSI のチップに数千万から億単位のトランジスタが載り，CPU と大規模なメモリ（場合によってはアナログ回路も）を組み込んだシステム LSI[2]という高機能システムを実現している．

有名な経験則「ムーアの法則」によれば，4倍/3年の速さで集積度が上がる．コンピュータの価格と体積は10分の1/10年で低下する[3]．価格の低下は普及に拍車をかけ，小型化（ダウンサイジング）とともに高速化・低消費電力化・高信頼化をもたらす．一方で，CPU の爆発的な普及は発熱や電力消費という地球環境問題まで引き起している[4]．

ここでは，このような変化の激しいハードウェア技術について，論理（ロジック）・メモリ素子などのデバイスと第2章で述べたアーキテクチャをこれらのデバイスで実現するための論理設計の基礎技術をとり上げる．

なお，これ以外にもコンピュータのハードウェア技術として重要なものにさまざまなものがある．例えば，ハードウェア記述言語（HDL）とロジックの検証・シミュレーション・テストと診断などの設計自動化技術，配線・MCM（Multi Chip Module）・冷却・パッケージなどの実装技術がある．その他に，近年多様化をみせるマルチメディア入出力機器のハードウェア技術がある．

[1] もちろん，データベース，分散処理や並列処理などに関連してソフトウェア技術もアーキテクチャに大きな影響を与えている．

[2] この技術を SoC（System on a Chip）とよぶ．一つの LSI 上に複数の異なる機能の回路を混載し，システムレベルの機能を小型・低消費電力・低価格・高性能に実現したもの．

[3] 1970年頃，性能は価格の2乗に比例するという経験則「グロッシュの法則」も有名である．

[4] ごく最近のニュースによると，消費電力低減のため，従来の 5V 電源を 0.5V で動作するロジックの開発に着手している．消費電力は電圧の2乗に比例するから，一挙に 1/100 にすることができる．

3.1 論理設計

論理設計とは所要の機能（機能仕様）をもつ論理回路を組織的に作り出すことである．コンピュータにおいてアーキテクチャで展開された機能仕様は個々に設計された加算器・レジスタ・ゲート・バスなどを互いに接続することにより実現される．具体的に個々の単位機能を満たす論理回路を作り，これを積み上げていく際の強力な手法がブール代数である．実際には，遅延・回路規模の他，ハザードやレーシングなどの動作の安定性を考慮しなければならない．

論理回路にはCPUの演算回路やバスなどの**組合せ論理回路**と，組合せ論理回路とメモリを用いて実現される**順序論理回路**がある．

3.1.1 ブール代数とブール関数

(1) 論理演算

ブール代数では2値のブール変数をとり扱う．基本的には**論理積**・**論理和**・および**否定**の3つの演算が定義される．以下，ブール変数は論理値 $\{0, 1\}$ をとるものとする．ブール代数の基本公式を表3.1.1に示す．これらはすべてブール代数の公理から導かれる．

表 3.1.1: ブール代数の基本公式

吸収則 (absorption law)	$1+x=1$ $0+x=x$ $x+x\cdot y=x$	$1\cdot x=x$ $0\cdot x=0$ $x(x+y)=x$
交換則 (commutative law)	$x+y=y+x$	$x\cdot y=y\cdot x$
結合則 (associative law)	$x+(y+z)=(x+y)+z$	$x\cdot(y\cdot z)=(x\cdot y)\cdot z$
分配則 (distributive law)	$x(y+z)=xy+xz$	$x+yz=(x+y)(x+z)$
冪等則 (idempotent law)	$x+x=x$	$x\cdot x=x$
相補性 (complementary)	$x+\overline{x}=1$	$x\cdot\overline{x}=0$
二重否定 (double negatin)	$\overline{\overline{x}}=x$	
ド・モルガンの定理 (de Morgan's theorem)	$\overline{x+y}=\overline{x}\cdot\overline{y}$	$\overline{x\cdot y}=\overline{x}+\overline{y}$

(2) ブール関数

まず，2変数のブール関数表を表3.1.2に示す．

表 3.1.2: 2変数ブール関数表

入力ブール変数			ブール関数	名称	記号
	x	0 0 1 1			
	y	0 1 0 1			
出力ブール関数	z	0 0 0 0	$z_0=0$		
		0 0 0 1	$z_1=x\cdot y$	AND（論理積）	図3.2.3
		0 0 1 0	$z_2=x\cdot\overline{y}$	AND NOT y（抑止）	
		0 0 1 1	$z_3=x$		
		0 1 0 0	$z_4=\overline{x}\cdot y$	AND NOT x（抑止）	
		0 1 0 1	$z_5=y$		
		0 1 1 0	$z_6=\overline{x}\cdot y+x\cdot\overline{y}$	EX-OR（排他的論理和）	
		0 1 1 1	$z_7=x+y$	OR（論理和）	図3.2.4
		1 0 0 0	$z_8=\overline{x+y}=\overline{x}\cdot\overline{y}$	NOR（Pierceの演算）	図3.2.7
		1 0 0 1	$z_9=\overline{x}\cdot\overline{y}+x\cdot y$	Coincidence（一致）	
		1 0 1 0	$z_{10}=\overline{y}$	NOT y（否定）	
		1 0 1 1	$z_{11}=x+\overline{y}$		
		1 1 0 0	$z_{12}=\overline{x}$	NOT x（否定）	図3.2.5
		1 1 0 1	$z_{13}=\overline{x}+y$		
		1 1 1 0	$z_{14}=\overline{x\cdot y}=\overline{x}+\overline{y}$	NAND（Shefferの縦棒）	図3.2.6
		1 1 1 1	$z_{15}=1$		

(a) 真理値表とブール関数

u 個の入力ブール変数に対し，1個の出力ブール関数が決定される．このような，与えられた機能をもつ論理回路を求めよう．一般に u 個の入力ブール変数 $x_1, x_2, \cdots, x_u \in \{0,1\}$ のそれぞれに対し論理値を与えると，出力ブール関数 $z=f(x_1, x_2, \cdots, x_u) \in \{0,1\}$ が決まる．ここで，2^u 通りある入力ブール変数のとり得るすべての組合せに対応する出力ブール関数の値を表にまとめた**真理値表**よりブール関数を求める方法を説明する．

(i) 主加法標準形 出力ブール関数 $z=f(x_1, x_2, \cdots, x_u)$ を $x_1=0$ を含む項と，$x_1=1$ を含む項に分けると

$$f(x_1, x_2, \cdots, x_u)=\overline{x_1}f(0, x_2, \cdots, x_u)+x_1 f(1, x_2, \cdots, x_u) \qquad (3.1.1)$$

となる．例えば，$f(x)=\overline{x}f(0)+xf(1)$ である．すなわち，$f(x)$ は1変数 x の関数で表される．また，$f(x_1, x_2)=\overline{x_1}f(0, x_2)+x_1 f(1, x_2)$ となることを確かめれば明らかである[5]．この手順を，$x_2, x_3, \cdots, x_u$ についても行っていくと

$$f(x_1, x_2, \cdots, x_u)=\overline{x_1}\ \overline{x_2}\cdots\overline{x_u}f(0, 0, \cdots, 0)$$

[5] $x_1=0$ と $x_1=1$ の場合に分ければ簡単に証明できる．

$$+\overline{x_1}\,\overline{x_2}\cdots x_u f(0,0,\cdots,1)$$
$$\cdots$$
$$+x_1x_2\cdots x_u f(1,1,\cdots,1) \qquad (3.1.2)$$

と合計 2^u 通りの項に展開される．ブール関数を式 (3.1.2) のような形に展開したものを**主加法標準形**（principal disjunctive canonical form）という．各論理積の部分を**最小項**（miniterm）とよぶ．

式 (3.1.2) で $f(0,0,\cdots,0), f(0,0,\cdots,1),\cdots,f(1,1,\cdots,1)\in\{0,1\}$ であるから $f(\cdot,\cdot,\cdots,\cdot)=0$ の項はすべて式 (3.1.2) から除去される．

この結果，主加法標準形よりブール関数は次のようにして求められる．真理値表より，ブール関数値が 1 をとる変数の組合せだけをとり出す．ついで，それらのブール変数が 0 には $\overline{x_i}$，1 には x_i を対応させる．その論理積を作りそれらの論理和をとれば論理式が求まる．

(ii) 主乗法標準形 $f(x_1,x_2,\cdots,x_u)$ を主加法標準形より展開し，全体の否定をとったものに de Morgan の定理を適用すると

$$\begin{aligned} f(x_1,x_2,\cdots,x_u) =&\{x_1+x_2+\cdots+x_u+f(0,0,\cdots,0)\}\\ &\cdot\{x_1+x_2+\cdots+\overline{x_u}+f(0,0,\cdots,1)\}\\ &\cdots\\ &\cdot\{\overline{x_1}+\overline{x_2}+\cdots+\overline{x_u}+f(1,1,\cdots,1)\} \qquad (3.1.3)\end{aligned}$$

を得る．ブール関数を式 (3.1.3) のような形に展開したものを**主乗法標準形**（principal conjunctive canonical form）という．各論理和の部分を**最大項**（maxterm）とよぶ．

式 (3.1.3) からブール関数は次のようにして求められる．真理値表よりブール関数値が 0 をとる変数の組合せだけをとり出す．ついで，それらのブール変数が 0 には x_i，1 には $\overline{x_i}$ を対応させる．その論理和を作り，それらの論理積をとれば論理式が求まる．両標準形を用いた例を［例 3.1.1］に示す．

［例 **3.1.1**］ 次の表 3.1.3 の真理値表で表される 3 変数のブール関数を主加法標準形と主乗法標準形にそれぞれ展開する．

(i) 主加法標準形

$$\begin{aligned} f(x_1,x_2,x_3) =& f(0,0,0)\cdot\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}+f(0,0,1)\cdot\overline{x_1}\cdot\overline{x_2}\cdot x_3\\ &+f(0,1,0)\cdot\overline{x_1}\cdot x_2\cdot\overline{x_3}+f(0,1,1)\cdot\overline{x_1}\cdot x_2\cdot x_3\\ &+f(1,0,0)\cdot x_1\cdot\overline{x_2}\cdot\overline{x_3}+f(1,0,1)\cdot x_1\cdot\overline{x_2}\cdot x_3\end{aligned}$$

表 3.1.3: 3 変数の真理値表

x	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$+f(1,1,0)\cdot x_1\cdot x_2\cdot\overline{x_3}+f(1,1,1)\cdot x_1\cdot x_2\cdot x_3$$

真理値表から

$$f(0,0,0)=f(0,0,1)=f(1,0,0)=0$$
$$f(0,1,0)=f(0,1,1)=f(1,0,1)=f(1,1,0)=f(1,1,1)=1$$

を代入して

$$f(x_1,x_2,x_3)=\overline{x_1}\cdot x_2\cdot\overline{x_3}+\overline{x_1}\cdot x_2\cdot x_3+x_1\cdot\overline{x_2}\cdot x_3+x_1\cdot x_2\cdot\overline{x_3}+x_1\cdot x_2\cdot x_3 \qquad (3.1.4)$$

を得る．

(ii) 主乗法標準形

$$\begin{aligned} f(x_1,x_2,x_3) =&\{f(0,0,0)+x_1+x_2+x_3\}\cdot\{f(0,0,1)+x_1+x_2+\overline{x_3}\}\\ &\cdot\{f(0,1,0)+x_1+\overline{x_2}+x_3\}\cdot\{f(0,1,1)+x_1+\overline{x_2}+\overline{x_3}\}\\ &\cdot\{f(1,0,0)+\overline{x_1}+x_2+x_3\}\cdot\{f(1,0,1)+\overline{x_1}+x_2+\overline{x_3}\}\\ &\cdot\{f(1,1,0)+\overline{x_1}+\overline{x_2}+x_3\}\cdot\{f(1,1,1)+\overline{x_1}+\overline{x_2}+\overline{x_3}\}\\ =&(x_1+x_2+x_3)\cdot(x_1+x_2+\overline{x_3})\cdot(\overline{x_1}+x_2+x_3) \qquad (3.1.5)\end{aligned}$$

を得る． □

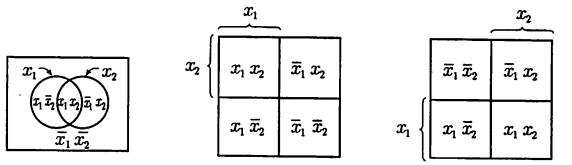
(b) ブール関数の変形

同一の機能をもつ論理式はいく通りも存在する．一方，NAND 回路のみで論理回路を構成することが多い．通常，遅延（論理段数）や回路規模（論理素子数）を考慮し，できるだけ簡単な回路で実現することが望ましい．

(i) ブール代数による方法 ブール代数の公式を用いて簡単化していく方法である．通常，最小項から $x+\overline{x}=1$（例えば，$xa+\overline{x}a=a$ など），$1\cdot x=x$，$x\cdot x=x$ などを用いて項数を減少させることにより行われる．

(ii) 図表による方法 **Venn 図表**，**Veich 図表**，**Karnaugh 図表**など，図表を用いる方法である．例 3.1.2 に 2 変数の場合の 3 つの図表の例を示す（図

3.1.1)．Venn 図表は集合を表すのによく用いられる方法であるが，項の数の増加と共に表現が急速に困難となる．Venn 図表では円で表したものを正方形で見やすく表したものが Veich 図表である．両図において一つの区画は最小項を示している．また逆に，最大項は区別し得る最大数の区画を示す．いずれも 2 変数のとき 4 区画をもつ．Karnaugh 図表は入力 u 変数を $u_1 + u_2 = u$ となるよう 2 つに分割し，2^u 個の区画表を作る．$u_1 = u_2 = 2$，$u = 4$ の場合を図 3.1.2 に示す．各区画は最小項を示す．ここで $\overline{x_i}$ は 0，x_i は 1 と表したとき，隣接した各区画は 1 ビットしか違っていない．すなわち，ハミング距離が 1 になるように配置する．しかも上下・左右が巡回状に隣接している．



(a) Venn 図表　(b) Veich 図表　(c) Karnaugh図表

図 3.1.1: 2 変数の場合の各種図表

x_1x_2 \ x_3x_4	0 0	1 0	1 1	0 1
0 0	0 0 0 0 $(\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4)$	0 0 1 0 $(\bar{x}_1\bar{x}_2x_3\bar{x}_4)$	0 0 1 1 $(\bar{x}_1\bar{x}_2x_3x_4)$	0 0 0 1 $(\bar{x}_1\bar{x}_2\bar{x}_3x_4)$
1 0	1 0 0 0 $(x_1\bar{x}_2\bar{x}_3\bar{x}_4)$	1 0 1 0 $(x_1\bar{x}_2x_3\bar{x}_4)$	1 0 1 1 $(x_1\bar{x}_2x_3x_4)$	1 0 0 1 $(x_1\bar{x}_2\bar{x}_3x_4)$
1 1	1 1 0 0 $(x_1x_2\bar{x}_3\bar{x}_4)$	1 1 1 0 $(x_1x_2x_3\bar{x}_4)$	1 1 1 1 $(x_1x_2x_3x_4)$	1 1 0 1 $(x_1x_2\bar{x}_3x_4)$
0 1	0 1 0 0 $(\bar{x}_1x_2\bar{x}_3\bar{x}_4)$	0 1 1 0 $(\bar{x}_1x_2x_3\bar{x}_4)$	0 1 1 1 $(\bar{x}_1x_2x_3x_4)$	0 1 0 1 $(\bar{x}_1x_2\bar{x}_3x_4)$

図 3.1.2: 4 変数の Karnaugh 図表

さて，いずれの図表を用いた場合も変数の数を減じるためには，共通の変数を括り出さねばならない．Karnaugh 図表においては，まず論理式の各項に対応する区画に印をつける．次いで，印のついた区画を 2 のべき乗個でできるだけ広く括り出す．その結果，例 3.1.3 に示すように論理式が単純化される．

(iii) 計算による方法 Karnaugh 図表を用いてもせいぜい $u = 7$ くらいまでで，さらに変数が増加したときは，一般に **Quine-McCluskey の方法**が

用いられる．この方法は主加法標準形から簡単な積和型ブール形式を求める組織的な方法である．詳細は省略するが，このアルゴリズムを用いてコンピュータにより $u = 100$ 程度の実行が可能である．

［例 **3.1.2**］ 2 変数の場合の図表
図 3.1.1 参照

［例 **3.1.3**］ 表 3.1.4 の $f(x_1, x_2, x_3)$ の真理値表より，図 3.1.3 の Karnaugh 図表を得る．その結果，関数 $f(\cdot)$ は

$$f(x_1, x_2, x_3) = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot \overline{x_2} + x_1 \cdot x_3 = \overline{x_2} + x_1 \cdot x_3 \qquad (3.1.6)$$

と導かれる． □

表 3.1.4: $f(x_1, x_2, x_3)$ の真理値表

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

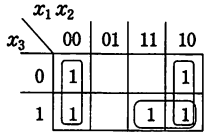


図 3.1.3: $f(x_1, x_2, x_3)$ の Karnaugh 図表

3.1.2 組合せ論理回路

これまで述べた論理回路は u 個の入力変数の組合せに対し，一般には v 個の与えられた出力ブール関数 $(v \leq 2^{2^u})$ を満足するものである．すなわち，図 3.1.4 に示すように

$$z_j = f_j(x_1, x_2, \cdots, x_u) \qquad (j = 1, 2, \cdots, v) \qquad (3.1.7)$$

で与えられる論理回路である．このように出力 $\{z_j\}$ が過去の入力には依存せず現在の入力 $\{x_1, x_2, \cdots, x_u\}$ によってのみ決まる論理回路を u 入力，v 出力の**組合せ論理回路** (combinational logical circuit) という．このような論理回路の設計手順は次のとおりである．

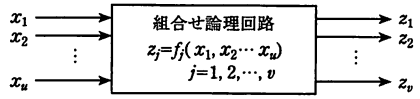


図 3.1.4: 組合せ論理回路のモデル

(i) 与えられた問題から**真理値表**を求める．
(ii) 真理値表から**ブール関数**を求める．
(iii) 求めたブール関数を**簡単化**し，実現しやすい論理式に変換する．
(iv) ブール関数から**論理回路**を求める．

以下に例をあげる．

［例 **3.1.4**］ （半加算器と全加算器[6]）

(a) 半加算器（Half Adder: HA） 半加算器は 1 桁の 2 進数の加算を行う回路で，表 3.1.5 に真理値表を示す．論理式は次式で与えられる．

$$\begin{aligned} s &= \overline{x}y + x\overline{y} = (x+y)(\overline{x}+\overline{y}) \\ c &= xy \end{aligned} \tag{3.1.8}$$

その論理回路を図 3.1.5 に示す．

表 3.1.5: 半加算器の真理値表

入力		出力	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

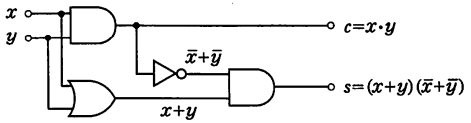


図 3.1.5: 半加算器の論理回路

(b) 全加算器（Full Adder: FA） 全加算器の真理値表は既に表 2.3.2 に示した．論理式は

$$\begin{aligned} s &= xyc_I + x\overline{y}\,\overline{c}_I + \overline{x}y\overline{c_I} + \overline{x}\,\overline{y}c_I \\ c_O &= xy + xc_I + yc_I \end{aligned} \tag{3.1.9}$$

となる．また，NAND 回路のみを用いた全加算器の例を図 3.1.6 に示す[7]． □

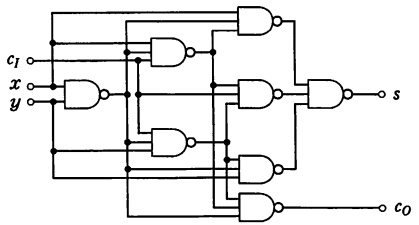


図 3.1.6: 全加算器

[6]以降，特に混乱のない場合は論理積 · は除去して表示する．
[7]これを用いた n ビットの並列加算器の例は既に図 2.3.3(b) に示した．

3.1.3 順序論理回路

ある時刻 t における出力 $\boldsymbol{z}^t = (z_1^t, z_2^t, \cdots, z_v^t)$ がその時刻の入力 $\boldsymbol{x}^t = (x_1^t, x_2^t, \cdots, x_u^t)$ と回路の内部状態 $\boldsymbol{y}^t = (y_1^t, y_2^t, \cdots, y_w^t)$ によって決まる論理回路を**順序論理回路**（sequential logical circuit）という．図 3.1.7 に示すように

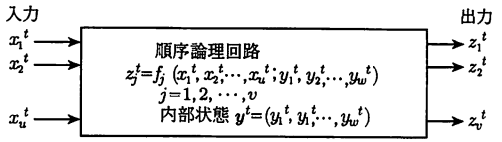


図 3.1.7: 順序論理回路のモデル

$$\begin{aligned} z_j^t &= f_j(x_1^t, x_2^t, \cdots, x_u^t;\, y_1^t, y_2^t, \cdots, y_w^t) \quad (j = 1, 2, \cdots, v) \\ y_k^{t+1} &= g_k(x_1^t, x_2^t, \cdots, x_u^t;\, y_1^t, y_2^t, \cdots, y_w^t) \quad (k = 1, 2, \cdots, w) \end{aligned} \tag{3.1.10}$$

で与えられる論理回路である．ここで，t は時刻を表す．順序論理回路はまた，組合せ論理回路とメモリ素子（または遅延素子）を用いても表現できる．図 3.1.8 に実現方法を示す．

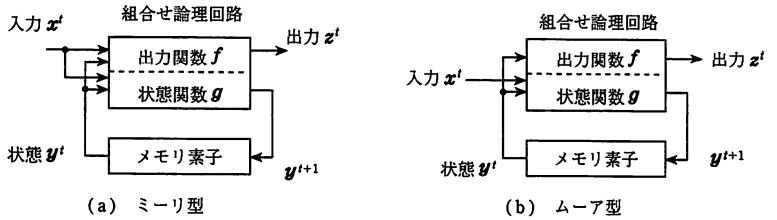
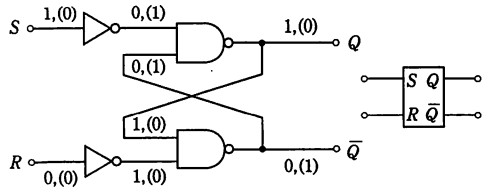


図 3.1.8: 順序論理回路の実現方法

(a) ミーリ型は出力信号を内部状態と入力信号から生成する．回路は簡単であるが入力の雑音に弱い．
(b) ムーア型は出力信号を内部状態のみ（入力信号に無関係）から生成する．回路構成はきれいな形をしており，入力の雑音の影響を受けない．

次に，代表例としてフリップフロップ（Flip Flop: FF）とその応用例を示す．フリップフロップは，2 つの安定状態をもった回路で，1［ビット］のメモリ素子となる．カウンタ，レジスタなどの基本回路として広く用いられている．

(a) RS フリップフロップ RS フリップフロップの真理値表・記号・回路図を図 3.1.9 に示す．論理式は次式で与えられる．



(a) RS-FF の回路　(b) RS-FF の記号

(c) RS-FF の真理値表

R^t	S^t	Q^{t+1}
0	0	Q^t
0	1	1
1	0	0
1	1	*

*：不定

(d) RS-FF の励起表

$Q^t \to Q^{t+1}$		R	S
0	0	ϕ	0
0	1	0	1
1	0	1	0
1	1	0	ϕ

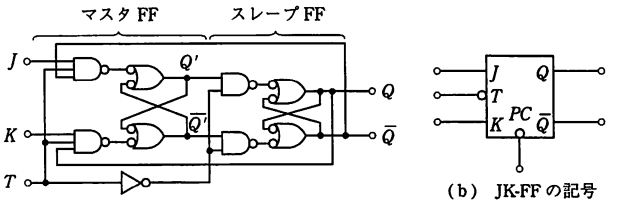
ϕ：0 または 1

図 3.1.9: RS フリップフロップ

$$Q^{t+1} = S^t + \overline{R}^t Q^t$$
$$R^t S^t = 0 \quad (R^t + S^t = 1) \tag{3.1.11}$$

(b) JK フリップフロップ J と K の入力端子のほかにトリガ端子 T にパルスが加えられ，出力が決定されるフリップフロップである．JK フリップフロップの真理値表・記号・回路図を図 3.1.10 に示す．論理式は次式で与えられる．

$$Q^{t+1} = \overline{K}^t Q^t + J^t \overline{Q}^t \tag{3.1.12}$$



(a) JK マスタスレーブ FF の回路　(b) JK-FF の記号

(c) JK-FF の励起表

$Q^t \to Q^{t+1}$		J	K
0	0	0	ϕ
0	1	1	ϕ
1	0	ϕ	1
1	1	ϕ	0

ϕ：0 または 1

(d) JK-FF の真理値表

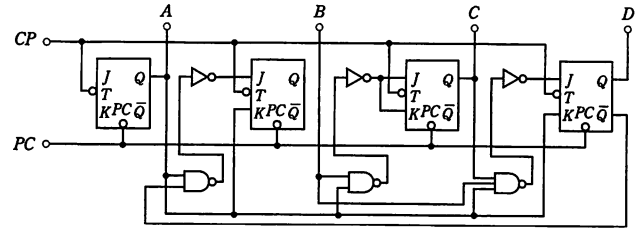
J^t	K^t	Q^{t+1}
0	0	Q^t
0	1	0
1	0	1
1	1	$\overline{Q^t}$

(e) プリクリア (Pre Clear : PC) の真理値表

PC	Q^{t+1}
0	0
1	Q^t

図 3.1.10: JK フリップフロップ

［例 **3.1.5**］ 図 3.1.11 に JK フリップフロップによる 10 進カウンタの例を示す．□



(a) 論理回路図

(b) 10 進カウンタの真理値表

D^t	C^t	B^t	A^t	D^{t+1}	C^{t+1}	B^{t+1}	A^{t+1}
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	×	×	×	×
1	0	1	1	×	×	×	×
1	1	0	0	×	×	×	×
1	1	0	1	×	×	×	×
1	1	1	0	×	×	×	×
1	1	1	1	×	×	×	×

×：不能

DC \ BA	00	01	11	10
00	0	0	[1]	0
01	ϕ	ϕ	ϕ	ϕ
11	×	×	×	×
10	0	0	[×]	×

$J_c = AB$

DC \ BA	00	01	11	10
00	ϕ	ϕ	[ϕ]	ϕ
01	0	0	[1]	0
11	×	×	[×]	×
10	ϕ	ϕ	[×]	×

$K_c = AB$

(c) J_c を求めるための Karnaugh 図表　(d) K_c を求めるための Karnaugh 図表
(J_c, K_c は出力 C の FF の J, K 入力を示す)

図 3.1.11: 10 進カウンタの論理回路

3.2 論理回路

コンピュータ内部でのデータ処理は {0,1} の 2 値による演算が用いられる．この演算は 2 安定状態をもつ**論理素子** (logical element) による**論理演算** (logical operation) によって行われる．これらの演算を行う回路が**論理回路** (logical circuit) である．

3.2.1 論理素子

最も広く使われている論理素子として，ダイオードとトランジスタについて説明する．**ダイオード**は順方向には電流が流れ，逆方向には電流がほとんど流れない．ダイオードの ON・OFF の特性を論理回路に利用する．この性質を図 3.2.1 に示す．トランジスタにはベース (B)・エミッタ (E)・コレクタ (C) の 3 端子がある．(a) バイポーラ npn 型トランジスタではコレクタ側が正電位（例えば，+5 [V]），エミッタ側が 0 電位（0 [V]）となるように電圧をかける．その際，ベースが正電位ならばコレクタからエミッタへ電流が流れ，0 [V] か負電位ならばこの電流は流れない．トランジスタはスイッチ素子として ON・OFF の特性を論理回路に利用する．この性質を図 3.2.2 に示す．なお，同図 (b) に MOS（Metal Oxicide Semiconductor）型のものを示す．MOS 型も基本的な動作はバイポーラ型と同一である．一般に，バイポーラ型に比べ集積度が高く電力消費は小さいが動作速度は遅い．

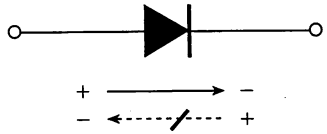


図 3.2.1: ダイオード

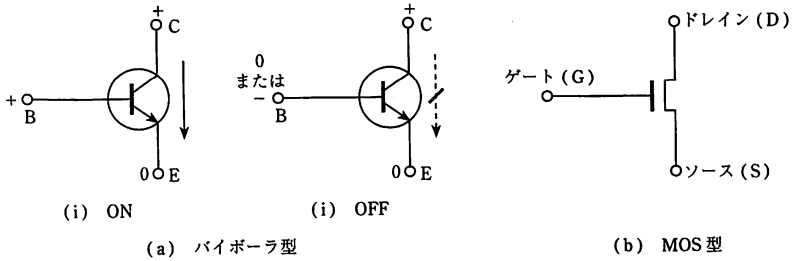


図 3.2.2: トランジスタ

3.2.2 基本論理回路

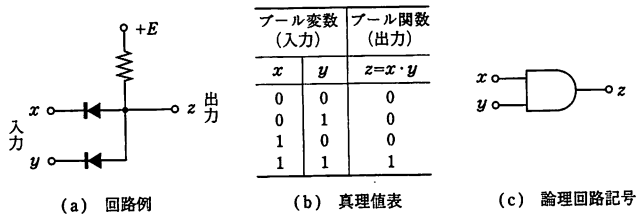
論理素子は**論理値**または**真理値** (truth value) として 2 つの状態，真 (truth) と偽 (false) をもち，通常真は 1，偽は 0 で表す．論理回路においては，例えば +5 [V] と 0 [V] をそれぞれの状態に対応させる．論理値 {0,1} をとる変数が**ブール変数**または**論理変数** (logical valiable) である．

以下の説明において，入力 x, y はブール変数である．出力 z は入力 x, y の関数値でブール関数または**論理関数** (logical function) で，$z = f(x, y)$ などと表される．すべての入力の組合せに対する出力の関係を表にまとめたものが**真理値表** (truth table) である．多入力の論理回路への拡張は容易である．なお，図中＋ E は通常＋ 5 [V] である．

(a) 論理積（AND）回路 入力のうち一つでも 0 があれば出力は 0 で，入力がすべて 1 ならば出力が 1 となる演算を**論理積**という．関数 z は次式で表される．

$$z = x \cdot y \tag{3.2.1}$$

図 3.2.3 に **AND 回路**の回路例・真理値表・論理回路記号を示す．



ブール変数（入力）		ブール関数（出力）
x	y	$z=x\cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

図 3.2.3: AND 回路

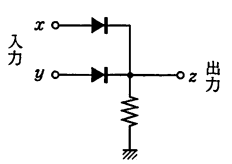
(b) 論理和（OR）回路 入力のうち一つでも 1 があれば出力は 1 で，入力がすべて 0 ならば出力が 0 となる演算を**論理和**という．ブール関数 z は次式で表される．

$$z = x + y \tag{3.2.2}$$

図 3.2.4 に OR 回路の回路例・真理値表・論理回路記号を示す．

(c) 否定（NOT）回路 入力が 0 ならば出力が 1，入力が 1 ならば出力が 0 となる演算を**論理否定**また単に**否定**という．ブール関数 z は次式で表される．

$$z = \overline{x} \tag{3.2.3}$$



(a) 回路例

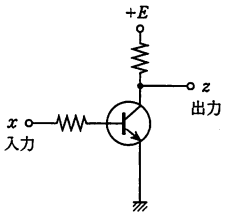
ブール変数（入力）		ブール関数（出力）
x	y	$z=x+y$
0	0	0
0	1	1
1	0	1
1	1	1

(b) 真理値表

(c) 論理回路記号

図 3.2.4: OR 回路

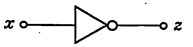
否定回路はトランジスタを用いて構成される．図 3.2.5 に NOT 回路の回路例・真理値表・論理回路記号を示す．



(a) 回路例

ブール変数（入力）	ブール関数（出力）
x	$z=\bar{x}$
0	1
1	0

(b) 真理値表



(c) 論理回路記号

図 3.2.5: NOT 回路

(d) 否定論理積（NAND）回路と否定論理和（NOR）回路 AND 回路の出力を否定したものを **NAND 回路**という．ブール関数 z は次式で与えられる．

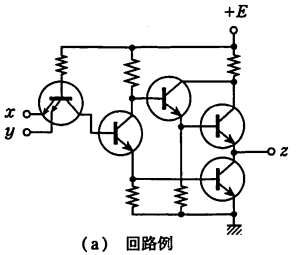
$$z = \overline{x \cdot y} \tag{3.2.4}$$

一方，OR 回路の出力を否定したものを **NOR 回路**という．ブール関数 z は次式で与えられる．

$$z = \overline{x + y} \tag{3.2.5}$$

図 3.2.6 に NAND 回路の回路例・真理値表・論理回路記号を示し，図 3.2.7 に NOR 回路の真理値表・論理回路記号を示す．NAND 回路，NOR 回路は広く利用されている．特に，NAND 回路は現在の論理回路の主流である．

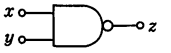
以上のように，高電位 H（＋5 [V]）を 1，低電位 L（0 [V]）を 0 に対応させる方法を**正論理**という．逆に，高電位を 0，低電位を 1 に対応させる方法を**負論理**という．このようにすると，例えば正論理の NOR 回路は NAND 回路を負論理にすれば得られることに注意されたい．表 3.2.1 に正論理と負論理の機



(a) 回路例

ブール変数（入力）		ブール関数（出力）
x	y	$z=\overline{x \cdot y}$
0	0	1
0	1	1
1	0	1
1	1	0

(b) 真理値表

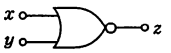


(c) 論理回路記号

図 3.2.6: NAND 回路

ブール変数（入力）		ブール関数（出力）
x	y	$z=\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

(a) 真理値表



(b) 論理回路記号

図 3.2.7: NOR 回路

能を示す．

なお，式 (3.1.2)，(3.1.3) より，任意のブール関数は論理演算の組 $\{+, \cdot, ^-\}$ で展開できることが明らかとなった．このような論理演算の組を**万能演算形**という．$\{+, \cdot, ^-\}$ は**万能演算形**であるが最小ではない．一方，$\{+, ^-\}\{\cdot, ^-\}$ は**最小万能演算形**である（演習問題 [3.5] 参照）．

3.3 メモリ素子

メモリに関する主なハードウェア技術は，主メモリの LSI 技術と外部メモリの磁性面記録技術である．ここでは，半導体メモリ（IC メモリ）素子について述べる．まず，以下に述べる SRAM と DRAM の特徴を表 3.3.1 に示す．

なお，メモリ素子の LSI 技術は論理回路・論理素子の構成方法・製造方法などと基本的に同一なので，次節 3.4 でとり上げる．

3.3.1 SRAM

SRAM（Static Random Access Memory）は当初バイポーラ型であったが，現在セルは MOS 型が大半である．クロックに同期して動作する同期型

表 3.2.1: 論理回路の記号と機能表

機能	正論理	負論理
x y | z H H | H L H | L H L | L L L | L	x, y → z AND	x, y → z OR
x y | z H H | L L H | H H L | H L L | H	x, y → z NAND	x, y → z NOR
x y | z H H | H L H | H H L | H L L | L	x, y → z OR	x, y → z AND
x y | z H H | L L H | L H L | L L L | H	x, y → z NOR	x, y → z NAND
x | z H | L L | H	x → z NOT	x → z NOT

H：高電位，L：低電位を示す．

表 3.3.1: SRAM と DRAM の比較

メモリ素子	動作速度	リフレッシュ	揮発性	集積度	価格	備考
SRAM	高速	不要	揮発性	低～中	高	NV RAM (不揮発性 RAM)
DRAM	中～低速	必要	揮発性	中～高	中～低	PS RAM (擬似 SRAM)

SRAM はキャッシュメモリなど CPU 周辺用が中心である．非同期型 SRAM は主メモリなど広く一般に用いられている．

図 3.3.1 に MOS 型メモリセルの例を示す．一つのセルはトランジスタを 4 つ用いたダイナミックフリップフロップ回路で構成されている．これは SRAM セルの主流であるが，高動作マージン・大容量・低電圧用として TFT（Thin Film Transistor）負荷型セルもある．ただし，TFT は論理用 LSI と製造プロセスが異なり混載用に向かない．

3.3.2 DRAM

DRAM（Dynamic Random Access Memory）はキャパシタに蓄えられる電荷の有無により記憶を保持するものである．メモリセルは 1 個のキャパシタと 1 個の MOS 型トランジスタで構成されている．図 3.3.2 にメモリセルの例を示す．

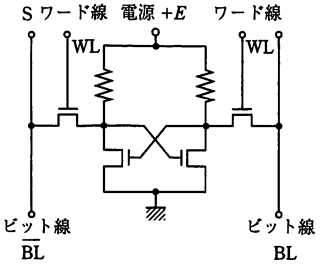


図 3.3.1: MOS 型 SRAM セルの例

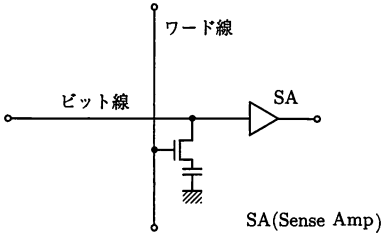


図 3.3.2: MOS 型 DRAM セルの例

書き込みはワード線を＋とし，ビット線にデータを与える．ワード線が＋なら電荷が蓄積 “1”，0 ならば電荷は放出 “0” される．読出しはワード線を＋としトランジスタを ON にすると，ビット線に出力が現れる．これをセンスアンプ（SA）で増幅する．MOS 型トランジスタが OFF のときのインピーダンスは大きい（数十 G [Ω]）ので，キャパシタンスが小さく（0.1p [F] 程度）ても電荷は数十 m [sec] は保持される．したがって数十 m [sec] ごとに再書き込み（リフレッシュ）が必要である．そのためダイナミック RAM とよばれる．リフレッシュのための時間は本来のアクセス動作ではないので無駄時間である．

DRAM はリフレッシュ回路やセンスアンプなど周辺回路を必要とするが，高集積・低価格が実現され，主メモリの主流である．

DRAM はメモリセルに蓄えられた電荷の破壊読出し・再書込み動作を行うため原理的にサイクル時間を短縮することは困難である．そのため，メモリバスの高速化に合わせさまざまな方式が工夫され実用化されてきた．SDRAM (Synchronous DRAM) [8], FPMDRAM (Fast Page Mode DRAM)

[8] メモリバスのクロックに同期した動作（立上りと立下りでデータ転送），パイプライン動作，チップ内メモリバンク選択動作などを用いる．

EDOMDRAM (Extended Data Output Mode DRAM), RDRAM (Rambus DRAM) [9]などがある.

特に, 画像用にはDRAMの転送速度の限界から専用DRAMが開発・実用化されている.

3.4 デバイス

ここまでアーキテクチャ, 回路, 素子とブレークダウンしてきた. 最後に残されたものは素子をどうやって製造するか, すなわち製造プロセスである.

以下に論理回路・論理素子とメモリ素子のバイポーラ型技術とMOS型技術について簡単に述べる. まず, 製造プロセス, 回路方式などを表3.4.1に示す.

表 3.4.1: 半導体素子 (論理素子・メモリ素子) の分類

種 類	製造プロセス	回路方式	特 徴			用 途
			動作速度	集積度	消費電力	
シリコン (S_i)	バイポーラ型	TTL[1]	中~高速	小~中	中	CPU内レジスタ
		ECL[2]	高速	小~中	中	キャッシュメモリ
	MOS型		低~中速	大	小	主メモリ
ガリウム砒素 (GaAs)	MESFET[3]	BFL SDFL DCFL LPFL	超高速	小	大	

1 Transistor Transistor Logic
2 Emitter Coupled Logic
3 MEtal Semiconductor Field Effect Transistor

集積回路 (IC) は同一の半導体基板上に多数の半導体デバイスを作り, アルミニウムの蒸着により配線し回路を構成したものである. この回路規模を大きくしたものがLSI, VLSIであり, いずれもモノリシックICとよばれる. これらの製造プロセスは写真製版に似ており, 量産に向く.

3.2節で述べた最小万能演算系 (例えばNAND系) は単なる数学的興味より, むしろ実用的に重要な意味がある. すなわち, 任意のブール関数の論理回路を構成するのに唯一の論理回路素子を用いて実現できる. このことは, ICの製造プロセスと整合し極めて有利である.

(a) バイポーラIC 図3.4.1に示すように, npnバイポーラトランジスタをシリコンウェハ (基板) 上にマスクをかけ不純物濃度を制御し, イオン注入して作成する. このようなトランジスタは1チップ上に数十万~数百万個を一気に作り上げ, これらをp型基板に逆バイアスをかけて個々のトランジスタを絶縁する. これらのトランジスタをアルミ蒸着による配線で相互接続し, 機能を作り出す. このとき, ダイオードはトランジスタのエミッタとベース・コレクタ間の性質を用い, また抵抗は拡散層を使用し, 同じ構造部分を用いて実現される.

[9]分布定数回路と進行波を用いて設計された時間位相を合わせる方式, 500M~1G [バイト/秒] のデータ転送を可能とする.

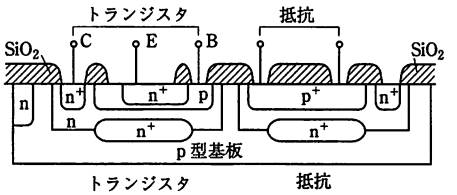


図 3.4.1: バイポーラIC

(b) MOS IC MOS型トランジスタにはnチャネル型とpチャネル型がある. 両方のトランジスタを用い消費電力を小さく, 動作マージンを大きくしたCMOS (Complementary MOS, 相補型MOS) が主流である. 数千万個以上のトランジスタを用いたVLSI技術はMOS型に限られる. したがって, MOS型の性能向上と共に普及型のCPU・メモリデバイスはバイポーラ型からMOS型に移行しつつある. 図3.4.2に示すように, nチャネル型のときはp型基板の上にn型のソース・ドレインを作り, 両者間の表面にアルミニウムまたは多結晶シリコンの薄い酸化膜をはさんでゲートを設けトランジスタを作る. 抵抗もMOSトラジスタを用いる.

なお, バイポーラ型の高速性とMOS (CMOS) 型の高集積度の利点を生かしたBi (C) MOS型がある. 例えば, メモリ素子をMOSで, アクセス用回路をバイポーラで実現したメモリもある.

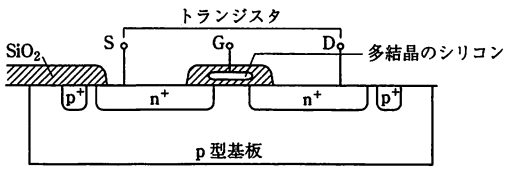


図 3.4.2: MOS IC

演習問題

［3.1］ 1変数ブール関数 $B_1 = \{f_1, f_2, f_3, f_4\}$ を示せ.

［3.2］ （半加算器）表3.1.5から，主乗法標準形を用いて式（3.1.8）を求めよ.

［3.3］ （全加算器）表2.3.2から
(1) 主加法標準形を用いて式(3.1.9)を求めよ.
(2) これをNAND回路[10]のみを用いて表せ.

［3.4］ **半減算器**（Half Subtracter: HS）と**全減算器**（Full Subtracter: FS）の真理値表・論理式・論理回路を示せ.

［3.5］ 最小万能演算系の例を2つ以上示せ.

［3.6］ ブール代数における**双対性**とはなにか．また，**正論理**と**負論理**の関係を示せ.

［3.7］ 3入力多数決回路についての(1)真理値表，(2)論理式，(3)論理回路を示せ.

［3.8］ 10進数における9の補数，10の補数を説明せよ．また2進化10進符号の補数回路を示せ.

［3.9］ 次の10進数の1桁の加算回路を示せ.
(1) 2進化10進符号
(2) エクセス3符号

［3.10］ (1) RS Flip Flop，(2) $J = K = 1$ のときのJK Flip Flopの遷移図を示せ.

［3.11］ トランスファースイッチを用いて，1階からも2階からも点滅できる階段ランプのスイッチ回路を構成せよ.

[10] $z = x|y \quad (= \overline{x \cdot y} = \overline{x} + \overline{y})$

4 ソフトウェア

ユーザ（利用者）から見たソフトウェアとは，要求分析・仕様記述・基本設計・詳細設計・コーディングの各工程を経て作成される成果物（要求仕様書・基本設計書・詳細設計書・プログラムを総称したもの）であるが，狭義には最終のプロダクトであるプログラムの集合体を指す．ここでは，ユーザの応用プログラムに近いものを述べるのが目的ではないので，ハードウェアに最も近い**オペレーティングシステム**（Operating System，以下**OS**と略す），特に**カーネル**（kernel）とよばれる制御（管理）プログラムとコンパイラについて簡単に述べる.

本書の改訂前ではメインフレームのOSをとり上げたが，ここではUNIXを視野に入れて具体的に説明する．UNIXは簡潔で身近（オープン化）なものであり，しかもマルチプログラミング環境をもち，マルチメディア対応，並列・分散OSとしての機能をもっている．ただし，むやみに偏ることを避けメインフレーム用からPC用OSの基本機能・共通機能を考慮している．この分野の術語はOSの系列・開発企業各社により必ずしも統一されていない．そのため，なるたけ古くから用いられている術語も記載している.

4.1 オペレーティングシステムの概要

OSはハードウェアに最も近い下位のソフトウェアである．主な機能はハードウェアを有効利用するための制御・管理とユーザに使いやすい環境を提供することである.

4.1.1 オペレーティングシステムの歴史

OSの誕生以前には，パネルスイッチなどで**ブートストラップローダ**を呼び出し，プログラムを主メモリに読み込み可能な初期状態にし，**テストプログラム**を流して準備していた．この結果，プログラムの実行には多くの人手の介入を必要とし，またプログラマがオペレータも兼ねた．このような運用方法を

5.3 情報セキュリティとネットワーク技術

ここではまず，**DES** や **RSA 暗号**など暗号系の特徴，暗号強度などの実状，実際に応用する際必要となる**認証技術**について述べ，さらに電子商取引や電子現金を実現するために必要なセキュリティシステムの例として**公開鍵基盤**（Public Key Infrastructure: **PKI**）の考え方について述べる．

なお，ここでは電子商取引に必要な技術的解決法について述べるが，ネットワークの不正利用は単に技術的対策だけでは解決が難しい．ユーザの情報倫理・道徳を啓蒙したり，法律を整備することも必要である．また，具体的に不正を未然に防止するしくみ（セキュリティポリシーの確立・運用，監査など）や危機管理プログラム策定など各方面から取り組まなければならない．

5.3.1 暗号系と暗号アルゴリズム

(1) 暗号系のモデル

暗号系のモデルを図 5.3.1 に示す．情報源 $\mathcal{M}$ から出力された通常の誰にも意味が理解できる**平文**を m，これを**暗号化鍵** K_E を用いて暗号器により暗号化し意味が理解できない**暗号文** c に変換し通信路に送り出す．通信路には誤りはないものとし，復号器の入力 c から**復号鍵** K_D により復号し元の平文 $\hat{m}$ を受信者 $\hat{\mathcal{M}}$ に出力する．これを次のように表す．

$$\begin{aligned} &\text{暗号化 } C : c = C_{K_E}(m) \\ &\text{復　号 } D : \hat{m} = D_{K_D}(c) \end{aligned} \tag{5.3.1}$$

ここで

$$m = D_{K_D}(C_{K_E}(m)) \tag{5.3.2}$$

が成立しなければならない．ここでは，m, c とも 2 元記号または 10 元記号とし，数十［ビット］以上，あるいは 10 進数 数百桁のブロックごとに暗号化・復号するブロック暗号を示すが，1 ビットあるいは 10 進数 1 桁ごとに逐次暗号化・復号するストリーム暗号もある．

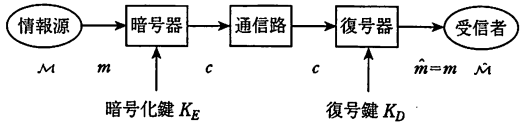


図 5.3.1: 暗号系のモデル

(2) 暗号系と不正

暗号の役割は大別して情報の**秘匿**と**認証**である．暗号系では図 5.3.2(a) に示すように暗号文 c が不正な第三者（盗聴者）により**盗聴**されること，(b) に示すように暗号文 c が不正な第三者により**改ざん**され c' となることの他，不正な第三者が正当な送信者になりすまし，あるいは送信者（受信者）が平文 m を送信（受信）しているにも拘らず送信（受信）していないと主張する**否認** などを考え対策を立てなければならない．改ざん，なりすまし，否認には認証機能が必要である．

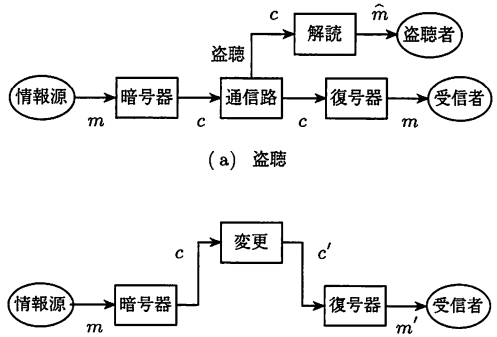


図 5.3.2: 暗号系における不正

(3) 秘密鍵暗号系

秘密鍵暗号（慣用暗号，対称鍵暗号，共通鍵暗号ともいう）は送信者（情報源）と受信者があらかじめ共通の秘密情報（秘密鍵 K）を共有し，この情報をもとに暗号化することにより秘匿や認証を行う方式である．したがって，図 5.3.1 および式 (5.3.1)，(5.3.2) において $K_E = K_D = K$ である．アルゴリズムを公開する現代暗号では，基本的には鍵に基づいて記号（文字）の**転字・換字**を繰返し適用する[47]．記号はビット，バイト，英数字などを単位としている．アルゴリズムが簡単で高速処理が可能である．秘密鍵暗号で有名なものは，1977 年米国 NBS により標準暗号として制定された DES である．

[47] 1949 年，C.E.Shannon は転字・換字を繰返し用いることにより，暗号強度が向上することを明らかにしている．

(4) DES

64［ビット］の平文 m をランダム化した 64［ビット］の暗号文 c に変換する．鍵は 64［ビット］（うち 8［ビット］はパリティビット）から簡単な方法で 16 個の 48［ビット］の部分鍵 $K_1, K_2, \cdots, K_{16}$ を作る．まず，2 元記号を単位に転置を実行し，次いで 64［ビット］の 2 元情報（平文）を 32［ビット］の 2 つのブロックに分割し，これに対し 16 段にわたりそれぞれ**転字**（転置）（記号の順序を入れかえること），**換字**（記号を他の記号におきかえること）をくり返す[48]．最後に最初の転置の逆転置を行う．ここで，換字処理は **S ボックス**とよばれる非線形変換である．そのため，平文とそれに対応する暗号文を入手したとしても鍵を割り出すのが困難とされている．図 5.3.3 に DES アルゴリズムを示す．

このように，暗号化アルゴリズムは完全に公開され，暗号の安全性は鍵に依存している．それゆえ，安全性を示す一つの指標として鍵の長さが問題となる．鍵の総当り[49]による解読が不可能な**計算量的安全**を確保するためには，現在少なくとも 80［ビット］以上が必要とされている．DES は，鍵の長さ $|K| = 56$［ビット］であり，トリプル DES（DES をさらに 3 段実行する方式[50]）や新しい型の秘密鍵暗号方式（Advanced Encryption Standard: **AES**）が検討されている．

秘密鍵暗号方式はアルゴリズムが簡単なため高速処理・小型化が可能で，現在データの暗号化を目的として実用化されているものの大半はこの方式である．しかし，暗号強度を鍵の管理により保持するため鍵の管理が重要である．また，N 人で相互に通信するために必要な鍵の数が $\binom{N}{2} = \frac{N(N-1)}{2}$ と多いのも欠点の一つである．

(5) 公開鍵暗号系

公開鍵暗号（非対称鍵暗号ともいう）は，暗号化鍵 K_E と復号鍵 K_D を一対ずつ作り，暗号化鍵を公開，復号鍵を秘密にする暗号方式である．送信者 A は公開リスト（電話帳のようなもの）上の受信者 B の暗号化鍵を用いて暗号文を作成し，受信者 B は自分だけが知っている復号鍵（これは秘密とする）を用いて暗号文を復号し平文を得ることができる．

公開鍵暗号系では暗号化鍵 K_E と復号鍵 K_D が異なり，公開された K_E から

[48] この操作をインボリューションという．
[49] 総当たり攻撃法以外に差分攻撃法・線形攻撃法などがある．
[50] AES が確立するまでの暫定的な標準．

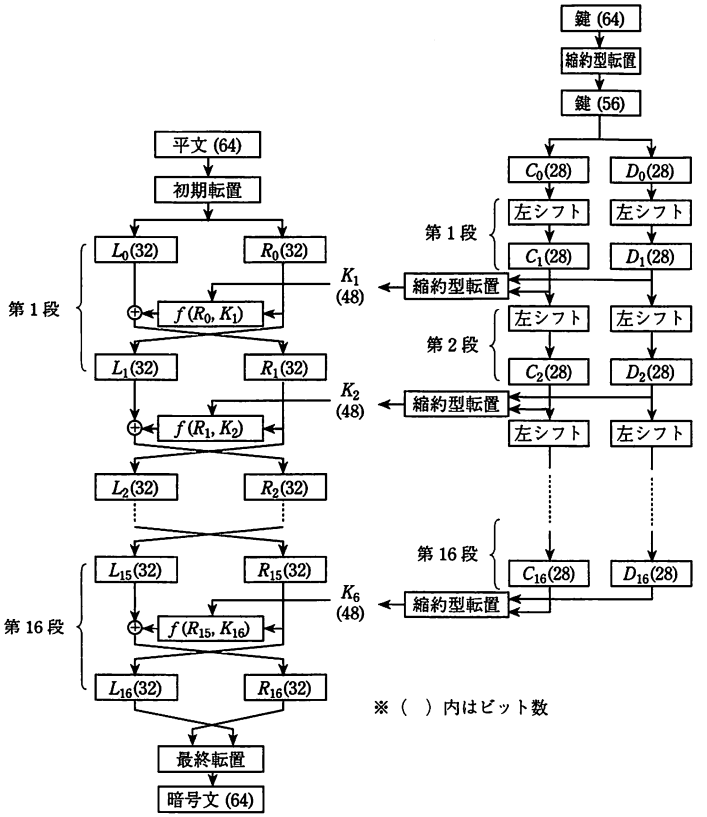


図 5.3.3: DES アルゴリズム

秘密にする K_D が容易に求まらない．このしくみを作るには**一方向性関数**が重要な役割をはたす[51]．

公開鍵暗号では，暗号解読の難しさは数論の難問を解く難しさに対応している．

(6) RSA 暗号

RSA 暗号はその安全性を大きな整数（例えば，10 進数 200 桁）の**素因数分解問題**を解く困難さに根拠をおいている．すなわち，2 つの素数 p, q を与えたときその積 $pq = n$ を計算することは極めて容易であるが，整数 n が与えられ

[51] 関数 $f(\cdot)$ が一方向性であるということは，順方向 $f(\cdot)$ の計算は容易（多項式時間で計算可能）であるが，逆方向 $f^{-1}(\cdot)$ の計算は（多項式時間では）困難であることをいう．

たとき $n=pq$ となる素数 p,q を求めるのは困難であるという一方向性を利用しているのである．

[RSA 暗号のアルゴリズム]

準備

(1) 2 つの大きな素数 p,q を選び，RSA 合成数 $n=pq$ を求める．p,q は秘密にしておく．

(2) オイラー関数 $\phi(n)=(p-1)(q-1)$ を計算し[51]

$$\mathrm{GCD}(d,\phi(n))=1 \tag{5.3.3}$$

$$ed=1 \pmod{\phi(n)} \tag{5.3.4}$$

を満たす対 (e,d) を求める．すなわち，$\phi(n)$ と互いに素な整数 d を求め[52]，$\phi(n)$ を法とする演算の下に d の逆数を e とする[53]．

(3) ここで

$$\begin{aligned} K_E(\text{暗号化鍵}) &: (e,n) \\ K_D(\text{復号鍵}) &: (d,n) \end{aligned} \tag{5.3.5}$$

とし，K_E を公開し，K_D を秘密に保つ[54]．

暗号化と復号

(1) 送信者 A はメッセージ（平文）m を $0\le m\le n-1$ の整数で表現し，受信者 B の公開鍵 K_E を用いて暗号文 c を

$$\text{暗号化}: c=m^e \pmod{n} \tag{5.3.6}$$

とする[55]．もちろん，暗号文 c は $0\le c\le n-1$ となる整数である．

(2) 受信者 B は自分だけが知っている秘密の鍵 K_D を用いて

$$\text{復　号}: m=c^d \pmod{n} \tag{5.3.7}$$

を計算し，平文 m を得る．

素因数分解問題が n の桁数の多項式オーダで解けるやさしい問題かどうかは明らかではないが，現在知られている最良のアルゴリズムを用いても素数 p

[51] 通常は $p-1$ と $q-1$ の最小公倍数を用いる．
[52] 実際には，$p-1$ と $q-1$ に対し互いに素であれば十分である．
[53] これは，ユークリッド互除法を用いて求めることができる．
[54] n を公開しているから，結局 d だけを秘密とする．もちろん，p,q は秘密である．
[55] $c,(e,n)$ から直接 m を求めることは離散対数問題でやはり困難である．

の準指数的オーダ，例えば，$O(e^{\sqrt{2\ln p\ln\ln p}})$ 程度[56]である．このため，コンピュータの処理速度の向上と共に RSA 暗号で用いる素数の積 n の桁数を増さなければならない．基本演算 1μs/step のコンピュータを用いて素因数分解を実行するときに要する処理時間を表 5.3.1 に示す．

表 5.3.1: 素因数分解の処理時間

n の桁数（10 進）	処理回数	処理時間
50	1.4×10^{10}	3.9 時間
100	2.3×10^{15}	74 年
200	1.2×10^{23}	3.8×10^{9} 年
300	1.5×10^{29}	4.9×10^{15} 年

公開鍵暗号では，秘密鍵暗号系のようにあらかじめ鍵を配送共有する必要はない．しかもこのとき，N 人で相互に通信するために秘密に管理しなければならない鍵の数は N でよい．しかし通常，整数の冪乗演算と剰余演算を実行しなければならず，計算速度が落ちるという欠点をもっている．そのため，長い平文の暗号化に用いられることは少なく，パスワードや（秘密鍵暗号系の）秘密鍵（共通鍵）の暗号化に用いられることが多い．

5.3.2 認証技術

(1) ディジタル署名

（送信）情報の正当性を保証する方法がディジタル署名である．RSA 署名，ElGamal 署名，Fiat-Shamir 署名などがある．

RSA 暗号を用いたディジタル署名（**RSA 署名**）はネットワーク上で確かに，ある情報を生成したことを保証する署名や押印に相当する機能である[57]．署名文を作ることができるのは**署名鍵**（RSA 暗号の復号鍵）$K_D=(d,n)$ を持っている者のみという公開鍵暗号系の特徴を生かしたものである．ここで，**検証鍵**（RSA 暗号の暗号化鍵）$K_E=(e,n)$ を公開している．図 5.3.4 に示したように，署名者（認証相手）B は認証文（認証子）h を RSA 暗号の秘密鍵 K_D で暗号化し，公開鍵 K_E で復号しても元の認証文に復元できることを用い

[56] ln は $\log_e$ を示す．
[57] 共通鍵暗号系でも信頼できる認証局が仮定できれば実現可能である．また，ゼロ知識証明を用いることもできる．

る．すなわち，認証文を h，その暗号文を s とすると，署名者 (B) は

$$\text{署名:} \quad s = h^d \pmod{n} \tag{5.3.8}$$

とし，平文 m と[58]署名の暗号文 s の対 (m, s) を送る．検証者 (A) は (m, s) より

$$\text{検証:} \quad h = s^e \pmod{n} \tag{5.3.9}$$

が成立するか否か検証し，成立すれば署名は正しいものとする．なお，ディジタル署名は認証文 h をメッセージ，ユーザ名，端末アドレス，時刻などとすることにより，それぞれメッセージ認証，エンティティ認証，端末認証，時刻認証などを実現することができる．

ディジタル署名を用いて**エンティティ認証**（送信者の同一性の確認）とメッセージ認証（情報の完全性の確認）を行うことができる（後述．図 5.3.8 参照）．

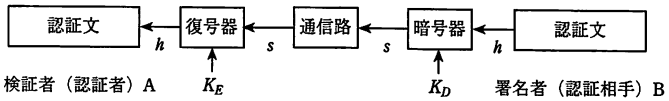


図 5.3.4: ディジタル署名

(2) 個人（ユーザ）認証技術

ネットワークを通して認証を可能とする技術の他に，機器（端末）が個人を認証する技術も必要である．最も簡単な方法として **ID 名** (identification name) とパスワードを用いる方法が普及している．表 5.3.2 に，現在開発中のものも含むこの分野の現状を示す [Koma 00]．いずれも本人であることを確認するため，本人しか持ち得ないものを検証することにより行われる．ここでは登録された本人のものと入力されたものを 1 対 1 に照合することを想定しているが，登録された N 人のものと入力したものを比較し個人を識別する方法もある．特に，個人の身体的特徴・特性を用いたものを**バイオメトリック個人認証**という．

[58] したがって，図 5.3.4 において B が A に署名文を送ることになる．なお，他人名義の公開鍵の不正取得を防止するために認証局（後述）の存在を仮定している．また，実際には平文 m のハッシュ値を h とする．ハッシュ関数は送受信者両方であらかじめ共有しておく（ハッシュ関数とは衝突を起こしにくい圧縮関数である）．もちろん，平文に直接署名してもよい．

表 5.3.2: 個人認証の方法

認証対象		具体的方法	留意事項
個人の知識		認証番号 パスワード	忘れる危険性
個人の所有		ID カード 鍵	盗難，紛失する危険性
バイオメトリックス	個人の身体的特徴	顔，掌形，指紋 網膜，紅彩	時間と共に変化する危険性
	個人の身体的特性	筆跡，声紋， キーストローク	

5.3.3 ネットワークセキュリティの技術

(1) 情報セキュリティ技術の階層

5.3.1，5.3.2 項で述べた暗号の固有技術を核として，**鍵管理・アクセス技術**などセキュリティ管理技術を加味し，システムとしてのネットワークセキュリティ技術を構築する．これを図 5.3.5 に示す．ここで，プロトコルはネットワークアーキテクチャの各レイヤにおける対策手順を示し，後に (3) でインターネットを想定したときの構造について述べる．

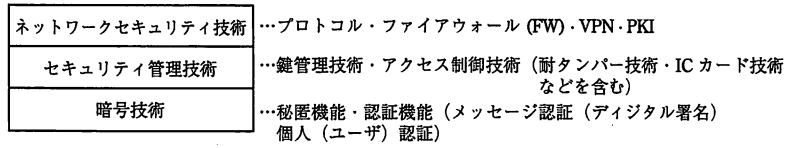
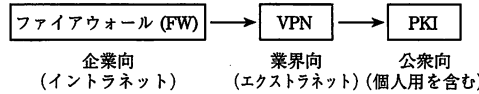


図 5.3.5: 情報セキュリティ技術

ネットワークセキュリティ技術から見ると，図 5.3.6 のようにシステムとして機能が向上していく．



VPN (Virtual Private Network)
PKI (Public Key Infrastructure)

図 5.3.6: ネットワークセキュリティ

(a) ファイアウォール（FW）…FW はフィルタ機能をもつゲートウェイ（インターネットではルータとよぶ）である．外部ネットワーク（インターネットなど）からの不正アクセスを防ぎ，外部ネットワークへの不正流出を防止するアクセスを可能とする（**アクセス制御**）．したがって，FW は外部と内部のサブネットワーク間のポリシー分界点に設置する (図 5.3.7 参照)．

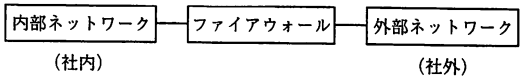


図 5.3.7: ファイアウォールによる内部ネットワークの保護

そのためパケットの送信アドレス・ポート番号・受信アドレスによってパケットをフィルタリングする．これが**パケットフィルタリングゲートウェイ**である[59]．FW にはこの IP フィルタリング機能以外にモニタリング・ログ機能がある[60]．

(b) VPN…VPN（Virtual Private Network）は公衆網を利用して，仮想的に専用線による私設網を構築したネットワークである．特定のルータ間の通信を暗号化することにより機密保持機能が得られる．その結果，不特定多数のユーザによる公衆網において，特定のユーザ（企業内など）だけが排他的に利用する閉域ネットワークを論理的に構築することができる．低トラフィックのユーザには専用線によるよりも低コストで機密保持が実現できる．通常は企業の内線番号による通信が可能である．

(c) PKI…PKI は公開鍵暗号[61]を利用して，証明書の作成・管理・格納・配布・破棄などに必要な資源・ポリシー・プロトコルによってサービスされる基盤（infrastructure）である．**認証局**（Certification Authority: **CA**）という信頼できる**第三者機関**（Trusted Third Party: **TTP**）を用いるのが特徴である[62]．5.3.4 項で詳しく述べる．

以上より，ウィルス対策を含むネットワークセキュリティ技術に対する不正対策とシステムの機能をまとめると表 5.3.3 のようになる．

[59]この他，TCP/IP のレイヤより上位のレイヤのアプリケーションからのアクセス要求を制御するサーキットレベルゲートウェイ・アプリケーションレベルゲートウェイがある．
[60]フィルタリングには，①第三者が詐称できる，②IP アドレスやサービスポート番号による場合，必要なものまで除去されることがあり利便性が低下する，という欠点がある．
[61]技術的には秘密鍵暗号を用いることも可能である．
[62]TTP を用いないで当事者間で通信を行う方法もある．

表 5.3.3: 不正対策とシステムの機能

	盗聴	改ざん	なりすまし	否認	暗号化	認証
ウィルス対策		○				
ファイアウォール	○					
VPN	○				○	
PKI	○	○	○	○	○	○

○印は不正対策が施されていることを示す．

(2) セキュリティアーキテクチャ

セキュリティアーキテクチャ（ISO 7948）では，①**相手認証**，②**アクセス管理**，③**秘匿**，④**情報の完全性**，⑤**否認防止**が規定され，それぞれに暗号技術が用いられる．すなわち，①④⑤には**認証機能**，③には**秘匿機能** が用いられる．

まず，①は通信相手が本当に正当な者であるかを確認することで，正当な通信者以外のなりすましを防止すること，②は①を行った後，あらかじめ許可された範囲内の資源 (例えばファイル) への**アクセス制御**であり，許可なく資源への不正アクセスを防止すること，③は**盗聴**などにより不正に情報が取られることを防止すること，④は通信回線上での意図的な**改ざん** や**誤り**が生じた場合，それらを検出可能なこと，⑤は通信者が通信に関して行った行為を後になって**否認**することを防止することである．これらは情報の秘匿の他に，その情報を誰が送信したか，それは正当な送信者であるかを証明する必要があるからである．そのために，**メッセージ確認**，**エンティティ認証**，**ユーザ認証**などの機能が必要である[63]．

表 5.3.4 に暗号ネットワークにおける不正行為とその対策を示す．セキュリ

表 5.3.4: 不正行為と対策

ネットワークへの攻撃・不正アクセス	セキュリティ対策
盗聴（解読）	秘匿（暗号化）
改ざん	メッセージ認証[1]（情報の完全性）
なりすまし（偽造）	エンティティ認証[1]・アクセス管理・ワイタイムパスワード
否　認	エンティティ認証[1]（送信者の同一性）
その他	

[1] RSA 署名などディジタル署名を用いる．

[63]これらは秘密鍵暗号系でも実現することができる．

ティ対策は暗号技術・セキュリティ管理技術が用いられる．表中のその他には破壊・裏切り・運用防害・結託・不注意・災害などさまざまなものが考えられる．

(3) ネットワークアーキテクチャとプロトコル

表 5.3.5 にプロトコルから見たネットワークセキュリティの技術を示す．

表 5.3.5: TCP/IP アーキテクチャとプロトコル

レイヤ	プロトコル
7: アプリケーション層	相手認証，アクセス権限制御
4: トランスポート層	回線選択（盗聴・障害回線の回避）
3: ネットワーク層	IP フィルタリング（FW）
2: データリンク層	回線暗号，コールバック，オフライン接続

データリンク層では通信回線に暗号装置を挿入し暗号化（回線暗号）したり，外部ネットワークからの着信呼は一旦回線を切断し内部ネットワークからかけ直し相手を確認するコールバックを用いる．場合によっては，ディスケットなどによるオフライン接続も用いられる．

ネットワーク層では IP パケットのアドレス・ポートからアクセス許可表を参照しながらフィルタリングを行う．これは FW の機能である．トランスポート層では不良な回線（パケット盗聴や通信障害が多いなど）を回避して安全な回線を選ぶ．アプリケーション層ではアプリケーションごとに通信相手を確認（相手認証）したり，アプリケーションへのアクセスの可否を確認する．

5.3.4 PKI

公開鍵基盤 PKI は公開鍵暗号を用いて電子商取引を可能とする公衆向ネットワーク基盤[64]である．暗号技術を用いた秘匿機能と認証機能により，① 電子認証システム，②電子公証システムを提供する[65]．認証・公証のためにディジタル署名技術が活躍し，証明書（鍵）を管理する認証局（CA）が重要な役割を演じる．

公開鍵暗号技術を用いて文書の暗号化・ディジタル署名により盗聴・改ざん・

[64]政府認証基盤（GPKI）は平成 11 年 12 月決定の「ミレニアムプロジェクト」の一つとしてあげられた「電子政府」を実現する核となるシステムである．ディジタル署名・認証に関する法制度の整備と各省庁が認証システムを構築し，これら（外国政府・民間の認証局も含む）を相互に接続するためのブリッジ認証局を設置し，相互認証のための審査基準を策定している．

[65]電子認証システムは取引相手を証明し，電子公証システムは取引日時と取引内容を証明するシステムである．

なりすまし・否認を防止（表 5.3.4 参照）し，電子商取引を可能とするものである．

(1) ディジタル署名の手順

5.3.2 項 (1) に RSA 署名について述べた．その根拠は，署名文を作ることができるのは RSA 暗号の復号鍵を持っている者に限られるというところにある[66]．これを，エンティティ認証とメッセージ認証に用いる手順の例を図 5.3.8 に示す．プロセス A を検証者（認証者），プロセス B を署名者（認証相手）とする．

(a) エンティティ認証（相手認証）は本当に通信相手が B であることを A が検証する方法である．

[エンティティ認証]

① A は B に乱数 r を送る．

② B は乱数 r を B の復号鍵（秘密鍵）$K_D^{(B)}$ で暗号化し，暗号文 $s = C_{K_D^{(B)}}(r)$ を送る．

③ A は s を B の暗号化鍵（公開鍵）$K_E^{(B)}$ で復号し，その結果 $r' = D_{K_E^{(B)}}(s)$ を得る．$r = r'$ であれば認証し，$r \neq r'$ のとき否認する．

これにより，B の他人によるなりすましが防止できる．

(b) メッセージ認証は B が署名し，これを A が確認する方法である．

[メッセージ認証]

① B は送りたいメッセージ（平文）m を自分の復号鍵（秘密鍵）$K_D^{(B)}$ で暗号化し，A に暗号文 $c = C_{K_D^{(B)}}(m)$ を送る．

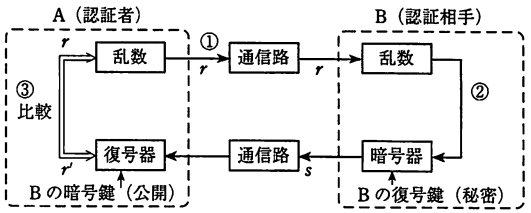
② さらに B は m から圧縮（要約）文 $h = h(m)$（$h(\cdot)$ はハッシュ関数）を作り，①と同様 $K_D^{(B)}$ を用いて A に署名文 $s = C_{K_D^{(B)}}(h)$ を送る．

③ A は暗号文 c から B の暗号鍵（公開鍵）$K_E^{(B)}$ を用いて m を復号し，これより B と同じハッシュ関数 $h(\cdot)$ を用いて圧縮文 h を作る．さらに，署名文 s から $K_E^{(B)}$ を用いて圧縮文 $h' = D_{K_E^{(B)}}(s)$ を作り $h = h'$ ならば B の署名を正当とし，$h \neq h'$ ならば不当とする．

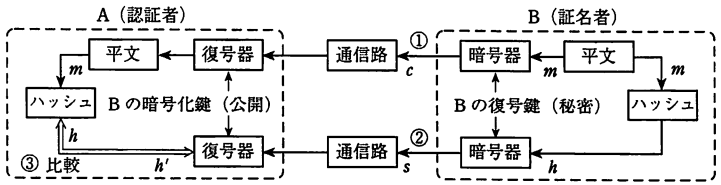
[66]RSA 暗号では暗号化鍵（公開）と復号化鍵（秘密）を入れかえても暗号機能をもつ．

これにより，メッセージ作成者がBであることを保証し，同時に改ざんや伝送路の誤りを発見することができる．

なお，図 5.3.8 (b) において，①を平文のまま送信することもできる（ただし，平文が露呈する）．またこのとき，②でハッシュ関数を用いない方法もあるが，通信量が2倍になる．さらに，①で平文 m を共通鍵暗号で暗号化し，その共通鍵を公開鍵暗号で暗号化する方法もある．



(a) エンティティ認証（相手認証）



(b) メッセージ認証

図 5.3.8: ディジタル署名

以上述べたように，メッセージ認証により盗聴・改ざんを防止することができる．また，メッセージ認証によりエンティティ認証とメッセージに含まれる行為情報の否認防止を行うことができる．

(2) 認証局の役割

ディジタル署名ですべての不正防止を解決したかに思えるが，実はそうではない．公開鍵暗号系の唯一の残された問題は公開鍵が本物かということである．例えば，他人名義の公開鍵が不正登録できればどうなるか．署名付き借用書は公開鍵で認証できるから身に憶えのない請求書をつきつけられることになる．そのために，申請者の本人確認と公開鍵が本当に申請者のものであることを確認し証明書を発行する信頼できる機関，**認証局（CA）**が必要となる[67]．認証局 CA の役割は次の通りである．

① **（申請者の本人確認）**ユーザ A は自分の公開鍵・秘密鍵のペア $(K_E^{(A)}, K_D^{(A)})$ を生成し，CA に加入を申請する．CA はユーザ A の身元調査（身分証明書，戸籍謄本，会社登記簿などによる）を行い身元証明を行う．

② **（公開鍵証明書の発行）** $\{\text{A}, K_E^{(A)}, \text{アルゴリズム}, \text{有効期限}, \cdots\} = M$ に対する CA のディジタル署名 $S = C_{K_D^{(CA)}}(M)$ を発行し A に渡す．

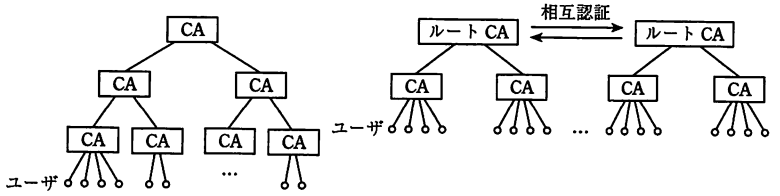
③ **（証明書の運用）** CA は申請ユーザからの証明書について有効性確認・廃棄・更新など運用管理を行う．

なお，②の（公開鍵）証明書のフォーマットは ITU-T で標準化が進んでおり，勧告 X.509（ISO/IEC 9594-8 も同じ）では，図 5.3.9 のように定められている．

バージョン番号
シリアル番号
証明者の署名アルゴリズム
有効期限
被認証者の情報公開鍵
発行者の情報
付加情報
発行者電子署名

図 5.3.9: X.509 公開鍵証明のフォーマット

さらに，図 5.3.10 に示すように CA を階層的に構成することも定められている．下位の CA は上位の CA によって信用が保証される．



(a) 階層的認証局　(b) 相互認証局

図 5.3.10: 認証局の構造

[67] この公開鍵証明は印鑑証明と同じ機能である．あらかじめ市役所などに登録された印鑑（実印）を発行された印鑑証明書により，契約書に押印された印鑑が確かに本人のものであることを保証するのである．この印鑑証明書に相当するものが**公開鍵証明書**である．

(3) 電子商取引のための手順

わが国においては，形式的証拠能力として署名や押印が本人のものであることを別途証明しなければならない（民事訴訟法第 228 条第 4 項）．そのための印鑑証明書の機能はディジタル署名によって実現可能であることを述べた．これが**電子認証システム**である．

一方，企業間取引を考えると，取引日時・取引内容を公証人（法曹資格を有する者）によって証明（公証業務）を行うものとされている．そのために，第三者機関（TTP）が次のようなサービスを提供する**電子公証システム**が必要である．

① 時刻証明…当該情報が当該時刻に存在したことを証明する（タイムスタンプ機能）．

② 内容証明…該当内容の情報が通信されたことを証明する．

③ 配達証明…受信者が当該情報を受取ったことを証明する．

④ 原本保存…情報（原本）を保存し，要求に応じ謄本を発行する．

①の例を図 5.3.11 に示す．ユーザ A の文書はハッシュにより圧縮される．TTP はこれに時刻情報を付加し，さらにハッシュにより圧縮する．これがタイムスタンプである．これを TTP からユーザ A を見た（これを TTA とする）復号鍵（秘密）$K_D^{(TTA)}$ で暗号化したものを時刻証明記録とする．証明記録は TTA の暗号化鍵（公開）$K_E^{(TTA)}$ で誰（ユーザ X）でも復号できるから，文書に添付することにより時刻証明ができる．

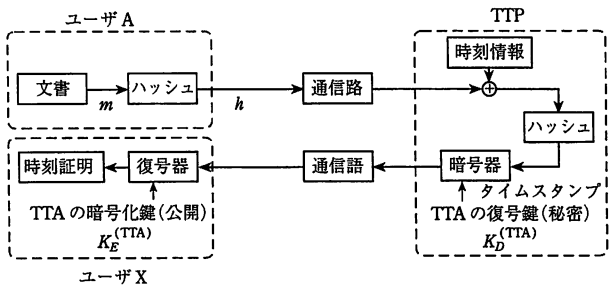


図 5.3.11: 時刻証明

5.4 データベース

データベース（Data Base: DB）は大量のデータの集合である．正確にいえば，それらデータを統合し共有利用するためのシステム（**データベース管理システム**（Data Base Management System: **DBMS**））によって，情報として新しい知識（価値）を生むものでなければならない[68]．

データベースは 1963 年，米国 GE (General Electric) で開発された IDS（Integrated Data Store）が最初とされている．この頃のデータベースはいずれも**構造形データベース**（現在，**階層形データベース**と**ネットワーク形データベース**がある）とよばれるものでレコード間をポインタで結ぶものである．1970 年，リレーショナルデータモデルが提案され，DBMS の開発・普及と共に表形データベースである**リレーショナルデータベース**が全盛を迎えた．1980 年代に入り，オブジェクト指向プログラミング言語によりデータと手続きが一体化したオブジェクトを単位とする**オブジェクト（指向）データベース**が実用化されている．さらに，文書・画像・音声などを対象とする**マルチメディアデータベース**，ネットワーク環境を利用した分散コンピューティングのデータベース，蓄積されたデータから演繹推論を実行する**演繹データベース**，データベースから埋もれた知識を発見的に発掘するデータ（ベース）マイニングも開発されている．

5.5 節で述べる情報検索システムもデータベースの一種である．しかし，情報検索システムで扱うデータは形が定まっており，ユーザのデータ更新を許さない．ここでは，データの作成者がデータの利用者であるような（狭義の）データベースシステムをとり上げる．したがって，実世界のデータからデータベースを構築するための規則を表現するデータモデルが重要な役割を果す（図 5.4.1 参照）．以下に，まずデータモデルとデータベースを管理するデータベース管理システム（DBMS）について述べ，最後に B 木やハッシュ技法などデータベース管理システムで用いられる重要なアルゴリズムを紹介する．

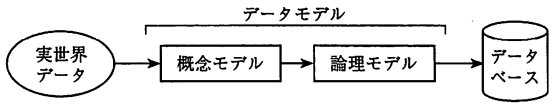


図 5.4.1: データモデルの役割

5.4.1 データモデル

実世界のデータからデータベースを構築し，DBMS を設計するためにはデー

[68] ここでは，データ と情報は若干異なる意味で用いている（第 1 章まえがき脚注参照）．データは実世界に存在する未加工のある事柄に対する知らせで記号列により表現される．人間の五感を通して確認できるものはすべてデータである．情報はユーザによりデータの意味が付与された知識で，一般に言葉で表現され必要なときとり出せる事実である．データを単に蓄積し，そのまま利用するシステムはファイルシステム（アプリケーションプログラムによって属性が定められた専用のデータ）でありデータベースとは言わない．