（コンピュータ基礎２）

2008 年 12 月 13 日

３．コンピュータの動作原理

3.1 記憶の特性

破壊読取り／非破壊読取り，揮発性／不揮発性，消去可能／永久

半導体記憶素子

集積回路 (IC: Integrated Circuit)，LSI (Large Scale Integration),
VLSI (Very Large Scale Integration)

- **RAM**(書き込み可)
  - SRAM(Static RAM)
    - バイポーラ型 (bipolar transistor)
      (小容量，高速，高価格，消費電力大)
    - MOS 型 (Metal Oxide Semiconductor,
      電界効果トランジスタ (FET) ともいう)
      (小〜中容量，低速，中価格，消費電力中)
  - DRAM(Dynamic RAM) $\cdots$ MOS 型 (大容量，低速，低価格，消費電力小)

SRAM：フリップフロップ回路を集積したもの．リフレッシュ不要.
DRAM：コンデンサに電荷が蓄えられることを利用した記憶素子．リフレッシュ必要.

- **ROM**(書き込み不可)
  - mask ROM $\cdots$ 製造時に内容書き込む (消去不可)
  - PROM(Programmable ROM) $\cdots$ ユーザが内容書き込む (消去不可)
  - EPROM(Erasable PROM) $\cdots$ 紫外線により消去可

3.2 記憶装置

- **主記憶装置の性能**
  アクセス時間 (access time)：読み書きの指令が出てから読み書きの動作が終了するまでの時間
  サイクル時間 (cycle time)：アクセス時間を含めて，読み書きの指令が出てから，次の読み書きの指令が出るまでの時間

- **処理速度の評価**
  MIPS(Mega Instructions Per Second)：１秒間に 100 万回の命令を実行する性能
  MFLOPS(Mega Floating Operations Per Seconds)：１秒間に 100 万回の浮動小数点計算命令を実行する性能

- **記憶の階層構造**
  記憶をアクセス速度の高低，記憶容量の大小などから階層的に組み合わせて，用途に応じた記憶装置を構成することを記憶階層 (storage hierarchy) という.

← 高速・小容量・高価格

レジスタ群	バッファ記憶装置	主記憶装置	磁気ディスク装置	大容量記憶装置

低速・大容量・低価格 →

3.3 論理演算

コンピュータは，四則演算のほかに論理和，論理積，否定などの論理演算を行っている．真 (1) または偽 (0) のいずれかの値をとる変数を論理変数 (logical variable) といい，論理演算に対する演算表を真理 (値) 表 (truth table) という．

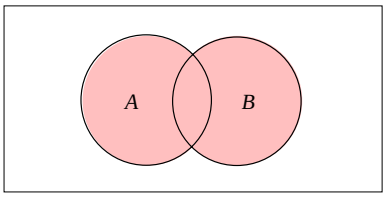
	論理演算子	論理記号	集合演算記号
論理和	$A+B$	$a \vee b$	$A \cup B$
論理積	$A \cdot B$ または $A \times B$	$a \wedge b$	$A \cap B$
否定	$\overline{A}$	$\sim a \ \ (\neg a)$	A^c
排他的論理和	$A \oplus B$	$(a \vee b) \wedge \sim (a \wedge b)$	$(A \cup B) - (A \cap B)$

- **論理和** (OR, disjunction, logical sum)

論理和の真理 (値) 表とベン図

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

```
    00001111
OR  01010101
------------
    01011111
```

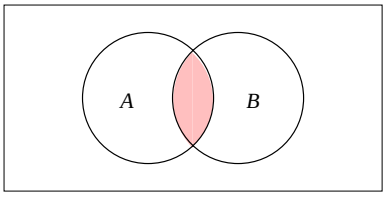


- **論理積** (AND, conjunction, logical product)

論理積の真理 (値) 表とベン図

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

```
    00001111
AND 01010101
------------
    00000101
```

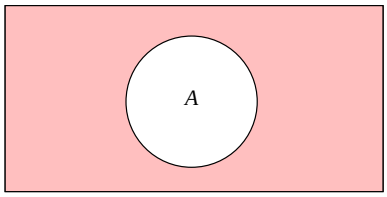


- **否定** (NOT, negation)

否定の真理 (値) 表とベン図

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

NOT	01010101
	10101010

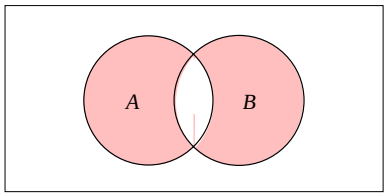


- **排他的論理和** (EOR, XOR, Exclusive OR)

排他的論理和の真理 (値) 表とベン図

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

	00001111
EOR	01010101
	01011010



- **否定的論理和** (NOR)

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

- **否定的論理積** (NAND)

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

- **基本公式**

交換律 (則)	$A+B=B+A$	$A\cdot B=B\cdot A$
結合律 (則)	$A+(B+C)=(A+B)+C$	$A\cdot(B\cdot C)=(A\cdot B)\cdot C$
分配律 (則)	$A+B\cdot C=(A+B)\cdot(A+C)$	$A\cdot(B+C)=A\cdot B+A\cdot C$
べき等律 (則)	$A+A=A$	$A\cdot A=A$
吸収律 (則)	$A+A\cdot B=A$	$A\cdot(A+B)=A$
ド・モルガンの法則	$\overline{A+B}=\overline{A}\cdot\overline{B}$	$\overline{A\cdot B}=\overline{A}+\overline{B}$

- **シフト 演算**

左シフト (8ビット)

0	1	0	0	1	0	1	0	$\cdots\ (74)_{10}$

0	1	0	0	1	0	1	0	0	$\cdots\ (148)_{10}$

右シフト (8ビット)

0	1	0	0	1	0	1	0	$\cdots\ (74)_{10}$

0	0	1	0	0	1	0	1	$0\cdots\ (37)_{10}$

ただし，算術シフトとよばれるシフト演算では，右シフトで符号ビットと同じものが空いたビット列に入る．

- **固定小数点の精度**

16 ビットの場合：-2^{15}〜$2^{15}-1=-32,768$〜$32,767$
32 ビットの場合：-2^{31}〜$2^{31}-1=-2,147,483,648$〜$2,147,483,647$
(負数は 2 の補数で表す)

- **浮動小数点の精度**

単精度

1つの数値を 32ビットで表現する浮動小数点数である．1ビットの符号部，7ビットの指数部，24ビットの仮数部の計 32ビットで表現する．表現できる値の範囲は -3.402823×10^{38}〜3.402823×10^{38} で，精度は6桁である．プログラム言語では float 型とよぶことが多い．

倍精度

1つの数値を 64ビットで表現する浮動小数点数である．1ビットの符号部，11ビットの指数部，52ビットの仮数部の計64ビットで表現する．表現できる値の範囲は $-1.79769313486232\times10^{308}$〜$1.79769313486232\times10^{308}$ で，精度は 15 桁である．プログラム言語では double 型とよぶことが多い．

- **オーバフローとアンダフロー**　固定小数点では，16ビットや 32ビットと大きさが決められるので，これ以上の数値を表現しようとするとオーバフロー (overflow) する．

浮動小数点では非常に大きな値または非常に小さな値同士を乗算すると，結果が指数部で表現できる範囲を超える場合がある．最大の数を超える場合をオーバフロー，逆に最小の数より小さい場合をアンダフロー (underflow) という．

4．論理回路

4.1 基本論理回路

コンピュータにおいては，2進法をもとにしたデータ 0 または 1 が入力されるとき，基本的な論理演算を組み合わせた論理回路によって 0 または 1 を出力する．基本的な論理回路には，論理和を行う OR (論理和) 回路，論理積を行う AND (論理積) 回路，否定を行う NOT (否定) 回路そして排他的論理和を行う EOR (XOR) 回路がある．それらは，図 1 のような回路記号で表される．ここで，データ 0 を低い電圧 L (たとえば，0.4 V)，データ 1 を高い電圧 H (たとえば，2.4 V) に対応させる方式を**正論理** (positive logic)，逆に 0 を高い電圧 H，1 を低い電圧 L に対応させる方式を**負論理** (negative logic) などという．

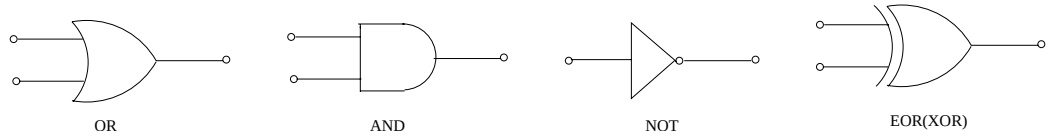


図 1: 基本論理回路

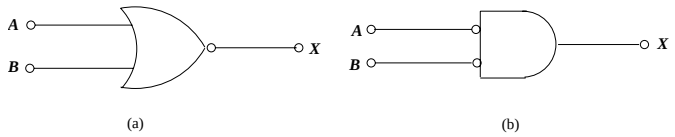


図 2: NOR 回路

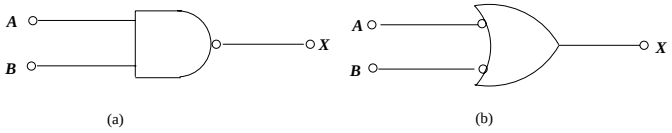
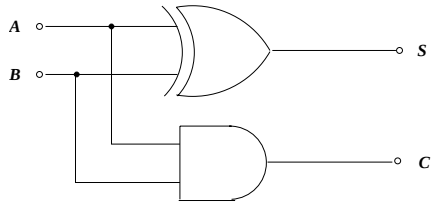


図 3: NAND 回路

- **半加算器**
  半加算器 (half adder) は，2進数の1桁の加算を行う回路で図 4 のように表され，真理値表はそこに示す表のようになる．

- **全加算器**
  n 桁の 2 進数の加算を行う場合，最下位の桁には半加算器を使用することができるが，上位の各桁の加算では下位からの桁上げを考慮した加算回路が必要となる．このような回路を全加算器 (full adder) という．基本的には 2 つの半加算器を組み合わせたものであるが，実際には NAND 回路等を用いた，図 5 のような回路が用いられる．真理値表は表 1 のように示される．

- **フリップフロップ回路**
  フリップフロップ回路 (flip-flop circuit) (flip-flop = flip-flap，バタバタの意味)，双安定マルチバイブレータで，双安定回路 (bistable circuit) ともよばれる．ビットの状態を保持する回路である．



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

図 4: 半加算器 ($S = A \oplus B$，$C = A \cdot B$)

- **レジスタ**

  レジスタ (register, 登録機の意味) フリップフロップ回路を並べて数ビットから数十ビットのデータを保持する装置で，演算を行う際などに用いられる記憶回路である．なお，とくに四則演算や論理演算などの結果を保持するレジスタを**アキュムレータ** (ACC, accumulator, 累算器) という．

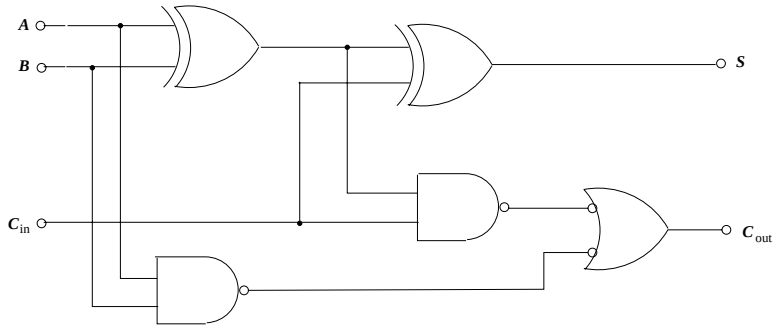


図 5: 全加算器 ($S = A \oplus B \oplus C_{\mathrm{in}}$，$C_{\mathrm{out}} = (A \cdot B) + C_{\mathrm{in}} \cdot (A \oplus B)$)

表 1: 全加算器の真理値表

入力			出力	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- **組合わせ回路と順序回路**

  組合わせ回路 (combination circuit) とは半加算器や全加算器のように，目的に合わせて基本的な回路を組み合わせた論理回路をいう．

  順序回路 (sequential circuit) とは，フリップフロップ回路やレジスタのように，出力値が入力値とその過去の値によって定まる論理回路である．

5．処理装置

5.1 処理装置の性能

コンピュータの中で各装置の制御やデータの計算・加工を行う中枢部分を処理装置 (processor) または中央処理装置 **CPU** (Central Processing Unit) という．PC ではこの CPU の機能を１つのチップに集積したマイクロプロセッサを用いている．１回の命令で同時に処理できるデータ量によって，８ビット，16 ビット，32 ビットなどの種類があり，ビット数の大きいものほど一般に性能がよいとされる．

コンピュータの処理速度を表すのに**クロック周波数** (clock frequency) が用いられることがある．これは，各装置の同期をとるための一定周期の信号 (水晶発振器に基づくクロック (時計) 信号) の周波数である．同じアーキテクチャ (architecture (建築), 構成，基本設計) のプロセッサであれば，クロック周波数の高いものほど，単位時間当たりの命令実行数は多い，すなわち，高速処理能力があるといえる．

5.2 命令と制御装置

コンピュータは，記憶されているプログラムにしたがって処理を行う．最も基本的なプログラムは，機械語 (マシン語) (アセンブラ言語 (アセンブリ言語, assembly)) の一連の**命令** (instruction) からなり，主記憶装置に記憶される．制御装置は，主記憶装置に収められている一連の命令を１つずつ取り出して解読し，各装置に対して必要な制御信号を与える．このように，あらかじめプログラムを主記憶装置に格納しておく方式をプログラム格納 (内蔵) 方式とよぶ．

5.3 プログラムの実行順序

主記憶装置に格納された命令は，１つずつ読出し・解読，そして実行され，次の命令に移る．その実行順序は，

命令サイクル (1)：主記憶装置から命令を読み出す

命令サイクル (2)：命令の解読，処理対象アドレスの計算

実行サイクル (3)：主記憶装置に対するデータの読み書き

実行サイクル (4)：命令が指定する演算の実行

このような命令の実行方式を逐次制御方式という．

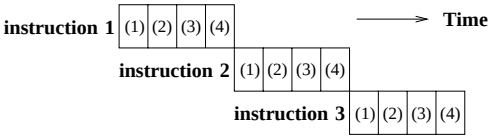


図 6: 逐次制御方式

- **パイプライン処理方式**
  通常は，１つの命令の読出し・解読・実行が完全に終わらないと，次の命令処理には移らない．しかし，処理速度を上げるために命令サイクルなどの空き時間を利用して，次の命令を前倒しに処理して行く図 7 のような処理方式をパイプライン処理方式という．

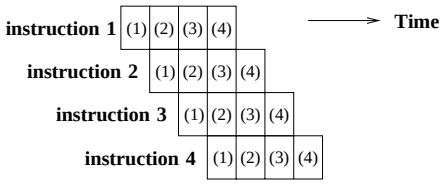


図 7: パイプライン処理方式

- **並列処理方式**
  科学技術計算の分野などで，大量の繰り返し計算を高速化するため，複数の処理装置を用いて並列的に計算を行うのが並列処理である．

5.4 命令の形式

機械語 (アセンブラ言語) は，**命令語** (instruction word) といい，図 8 に示されるように，命令部 (operation part) とアドレス部 (address part) から構成されている．いずれも 2 進数で表され，

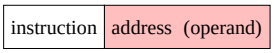


図 8: 命令部とアドレス部

アドレス部のように命令部の対象となる部分を**オペランド** (operand) ともいう．

5.5 アドレス指定方式

主記憶装置には固有のアドレスが付けられ，これを指定してデータを処理する．このアドレスを**絶対アドレス** (absolute address) という．命令部のアドレス部では，主記憶装置のアドレス指定の方法に次のようなものがある．

- 直接アドレス指定 (direct addressing)
  命令語のアドレス部で，データが記憶されている場所のアドレスを直接指定する方式である．

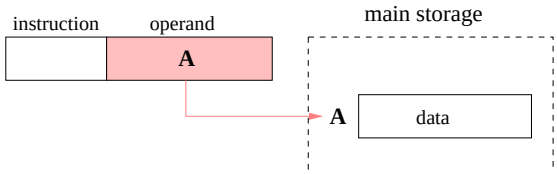


図 9: 直接アドレス指定

- 間接アドレス指定 (indirect addressing)
  命令語のアドレス部で，データが記憶されている場所のアドレスを直接指定せず，そのアドレスを記憶している場所のアドレスを指定する方式である．

- 即値アドレス指定 (immediate addressing)
  命令語のアドレス部に（アドレスではなく）処理の対象となるデータそのものをもつ方式である．

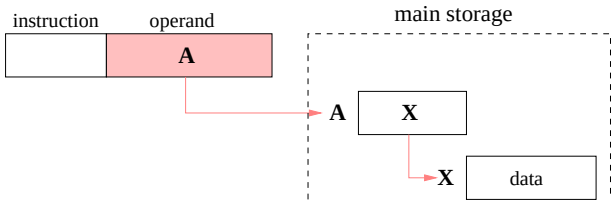


図 10: 間接アドレス指定

- 指標アドレス修飾 (index addressing)
  命令語のアドレス部では，レジスタ番号と数値をもち，その指定するレジスタ (**指標レジスタ**という) の内容とアドレス部とを加えて，対象とするデータが記憶されている場所のアドレスを得る方式である．

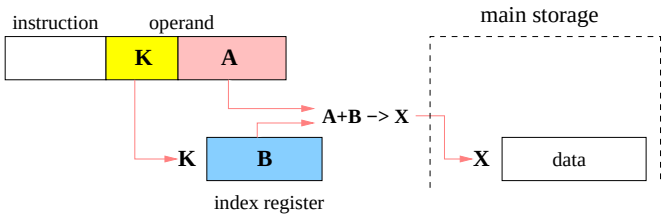


図 11: 指標アドレス修飾

- 基底アドレス修飾 (base addressing)
  図 11 において，指標レジスタ B を**基底アドレスレジスタ**とした方式である．その値を変えるだけで，主記憶装置の任意の場所にデータを記憶させる，あるいは任意の場所からデータを読み取ることができる．

なお，処理装置 (プロセッサ) の動作は，図 12 のような流れ図で記される．

5.6 プロセッサの構成要素

- **プログラムカウンタ** (program counter)
  命令アドレスレジスタ (instruction address register) ともよばれ，命令が実行されたあと，自動的に記憶内容が「+1」(increment) されて，次に実行すべき命令のアドレスが記憶内容となるレジスタである．

- **命令レジスタ** (instruction register)
  記憶装置から読み出された命令を一時記憶しておくレジスタである．

- **デコーダ** (decoder)
  復号器あるいは解読器ともよばれ，命令レジスタに読み取られた命令語の命令部を解読して，各装置への制御信号を発生する．

5.7 演算装置の構成要素

- **算術論理演算装置** (**ALU**: arithmetic and logic unit)
  加算器や補数器などを用いて，四則演算や論理演算を行う装置である．

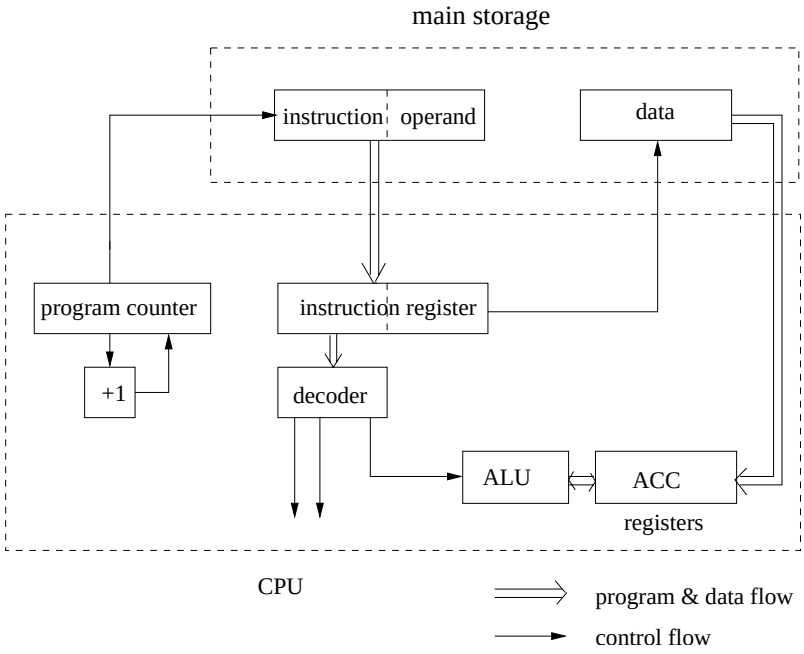


図 12: 処理装置の基本動作

- **アキュムレータ** (**ACC**: accumulator)
  累算器ともよばれる．演算結果を保持するための一種のレジスタである．
- **シフトレジスタ** (shift register)
  指定された数だけ 2 進数を左または右にシフトすることができるレジスタで，演算に使用される．
- **コンディションフラグ** (condition flags)
  演算結果が，0 か正か負か，桁あふれがあるかないかなどを表示するレジスタである．

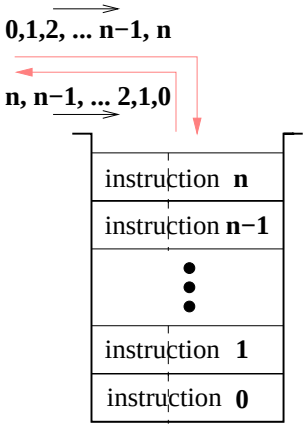


図 13: スタック

5.8 スタック (stack)

後入れ先出し 記憶装置をスタック (積み重ねの意味) という．複数の命令を順番に記憶するが，取り出しは最も後に記憶した命令から逆の順番で行う．サブルーチンの戻り番地の管理や多重の割り込み処理などに使用することができる．