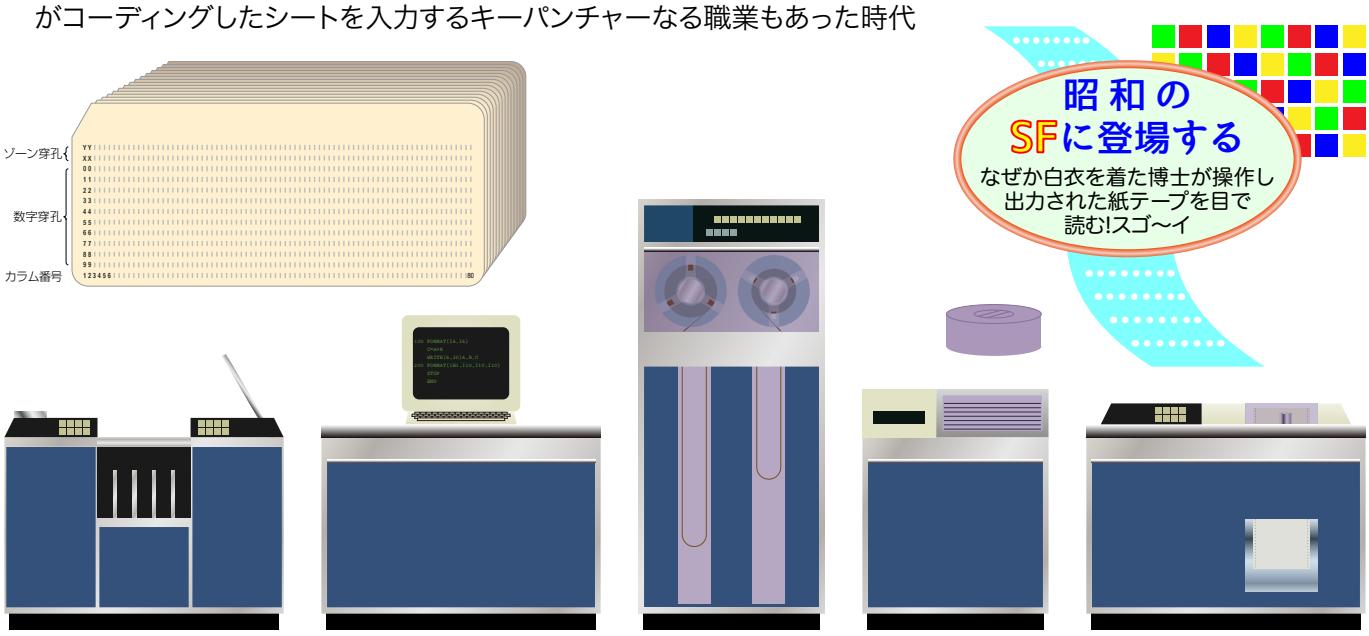
① 大型 コンピュータ

コンピュータは「問題解決のための道具」である
繰り返しの計算を得意とし、手順の決まったJOBを高速かつ正確に処理できる機械

1944年にリレー式コンピュータIBMのMARK-I(シーケンス制御のようなリレー回路)が作られた
リレーを真空管式にしたENIAC(真空管18800本が使用され交換メンテナンスが大変だったとも)で実用となるが配線盤で演算回路を組み替えていたので相当な労力を要したそうだ
フォン・ノイマン氏がプログラムもデータ同様、主記憶装置に内部記憶させるストアードプログラム方式を提唱
EDSAC機以降ノイマン型は継承されている

EDPS(Electronic Data Processing System)電子計算システムと呼ばれた時代、入力・記憶・演算・出力・制御で構成され
主記憶装置(RAMに相当)はマトリックス状に磁気コアを並べた磁気コア記憶装置
周辺装置としてコンソール(制御卓)、入力装置(カード読取装置、紙テープ入力装置等)、外部記憶装置(磁気テープ装置、磁気ドラム装置、磁気ディスク装置等)、出力装置(カード穿孔装置、磁気テープ装置、紙テープ出力装置、ラインプリンタ)データ通信機器

リーディングカンパニーはIBM(International Business Machines Corporation)で他社は互換機等で追随しプログラマーがコーディングしたシートを入力するキーパンチャーなる職業もあった時代



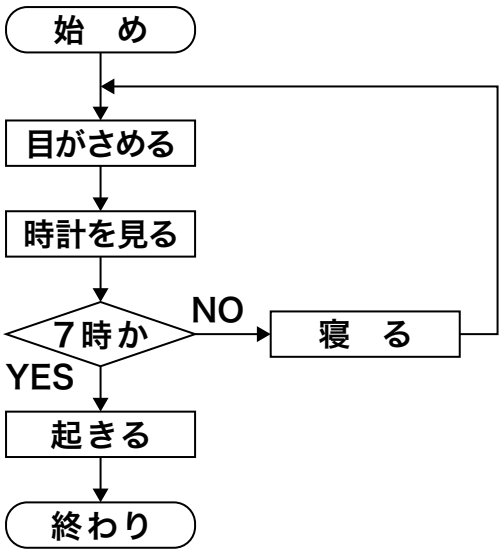
カード読取　主装置とコンソール　磁気テープ　磁気ディスク　ラインプリンタ

コンピュータに仕事をさせるには、その手順の命令と処理するデータを与える必要がある
命令は加算や判断といった単純なものであるため目的に応じた必要な命令群を作り実行させる
この命令群をプログラムという

プログラムを作る手順は人間が鉛筆で計算するのと同じで
プログラム言語が進化しても変わっていない

1. コンピュータに適合した手順を考える
2. 大まかな手順(フローチャート)を作る
3. フローチャートに従いプログラム言語で書く(コーディング)
4. コーディングされたプログラムを紙テープ又はカードにパンチ
5. 計算データを紙テープ又はカードにパンチ
6. 入力装置でプログラムとデータを読み込む
7. 記憶装置がプログラムとデータを記憶する
8. 演算装置がプログラムに従って計算をする
9. 制御装置が全体の動きを制御し制御装置もプログラムにより制御されている
10. 出力装置が8で処理されたデータを取り出す

〔フローチャート〕



② マイクロ コンピュータ

電子工学においてロジック回路はTTL(Tansister-Tansister-Logic)期にALU(Arithmetic Logic Unit)が出てきた
IC(Integrated Circuit)が LSI（Large Scale Integration:大規模集積回路）化していく時代にあって 専用の機能を持った
LSI から大量生産向きの汎用 LSI へと舵を切る･･･μコンピュータはCPUの歴史でもある

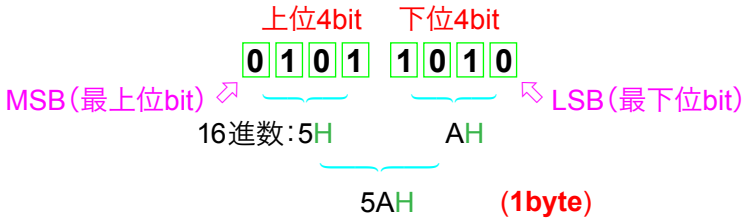
1971年	4004 (intel_4Bit)〔日本のビジコン社の電卓用IC受託〕					
1972年	8008 (intel_8Bit)					
1974年	8080 (intel_8Bit)	6800 (Motorola_8Bit〔68系〕)				
1975年			6502 (MOS_8Bit〔68系〕)			
1976年	Z80 (Zilog_8Bit〔80系〕)			TK-80 NEC(8080)1Bord kit		
	8085 (intel_8Bit_8080の改良)	CP/M		Apple Ⅰ (6502)		
1977年		UCSD Pascal		PET 2001 コモドール(6502)		
1978年	8086 (intel_16Bit〔x86系〕)			スペースインベーダー タイトー(8080)Game機		
1979年	8088 (intel_16Bit)	6809 (Motorola_究極の8Bit)		PC- 8001 NEC(Z80)		
1980年		68000 (Motorola_16Bit)				
1981年		MS-DOS	IBM-PC (8088)	FM-8 富士通(6809)		
1982年	80186 (intel_16Bit)			PC- 9801 NEC(8086)		
1984年		68020 (Motorola_32Bit)		IBM-PC/AT (80286)		
1985年	80286 (intel_16Bit)					
1986年		FLEX				
1987年	80386 (intel_32Bit_CISC)	68030 (Motorola_32Bit_CISC:Complex Instruction Set Computer=複雑命令セット)				
1989年	80486 (intel_32Bit)					
1991年	Linux	PowerPC (Apple-IBM-Motorola_64Bit_RISC:Reduced Instruction Set Computer=縮小命令セット)				
1992年		Windows 3.1		DOS-V (PC/AT互換機)		
1993年	Pentium (intel_32Bit〔x86系〕)					
1994年			Windows NT3.1	Power Macintosh (Power PC)		
1995年	Pentium Pro (intel_32Bit)	Windows 95	Windows NT4.0			
1997年	Pentium Ⅱ (intel_32Bit)					
1999年	Pentium Ⅲ (intel_32Bit)					
2000年	Pentium 4 (intel_32Bit) (Wintel時代)					
2001年	Xenon	Windows XP	Mac OS X			
2002年	Pentium4 Hyper-Threading (クロック競争限界からMulti Core化へ)					
2003年			Opteron (AMD_64Bit)	Power Mac G5 (Power PC970)		
2004年	Xenon (intel_64Bit_Nocona以降)		Athlon 64 (AMD_64Bit)			
2005年			Turion 64 (AMD_64Bit)			
2006年	Core 2 (intel_64Bit〔x86系〕)			Intel Mac		
2008年	Core i (intel_64Bit)					
2015年		Windows 10				
2017年			Ryzen (AMD_64Bit)			
2020年		M1(Arm) (Apple-64Bit_RISC) (汎用から専用へ)		Arm Mac		

③ 2進数

2進数は電気スイッチの On/Off に例えられるが電子回路では電圧の高低で区別する
Hの状態はTTLレベルでは5v CMOSレベルでは3.3v DDR2世代では1.8v DDR4世代では1.2vと段々低電圧化（省エネ化）していると同時にL→H逆にH→Lの変化も電位差が低いと高速になる
マイコンの波形はオシロスコープで見ると結構ノイズで汚ないので1と0の判別ポイント（スレッショルドレベル）がある（TTLレベルでは0.8V以下で論理L / 2.0V以上で論理Hとなる）

2進数は1桁で0と1の2値しか表現できない
この1桁をbit（ビット）という
4bitは 0000 ～ 1111 までの16通りの値をとるので16進数として 0H ～ FH で取扱う
（10進数と区別するためhexadecimalの略で末尾にHや頭に&Hを付ける）

16進コード（HEXコード）
4bitの各桁の重み付けで2進から16進の変換が簡単になる（右図）
この16進数で表される4bitを2桁にした8bitを1byte（バイト）という



16進 \ Bit	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

BCD コード（2進化10進法）

byteという単位はCPUが8bitから64bitになってもず～っと続いている
メモリー（RAM,HDD,SSD,DVD等）の容量の単位
2^8＝256MB(Mega Byte),2^9＝512MB,2^10=1024MB≃1GB(Giga Byte)
モニターの表示色は2の累乗で 2^8(1byte)＝256色　2^16(2byte)＝65536色

④文字コード

一般に0～9の数字、A～Zのアルファベット、＋－＊@等の特殊文字で64文字で表しアルファベットの小文字等を含めた1byteのASCIIコードによる2進化を使用する

KeyBordの配列はTypewriterのQWERTY配列が一般的
キーボードに内蔵されたCPUがKeyscanしTypeされるとPC本体への割込処理とBIOSへコードを伝達する
チャタリング（スイッチの誤動作）もプログラムで防止（マイコン時代のキーボードはPIOに接続されていた）

漢字は2byteで富士通FM-8やNEC_PC88～98ではマザーボード上に漢字ROMが搭載されていた

MS-DOSでは日本語入力フロントエンドプロセッサ（FEP：デバイスドライバ）で《かな漢字変換》をしていた
PC/AT互換（DOS/V）機の登場で日本語環境がソフトウェアで利用可能となり日本語が標準化された JIS X 0208 はShift-JISコードで普及し、かな漢字変換もインプットメソッド（IME)で行うようになった
その後、収録文字種が不十分だったのとグローバル化で全ての文字を統合するUnicodeへと規格が混在中

OpenTypeのフォントはUnicodeに対応されている

ホームページ（HTML5）での文字はUTF-8（Unicodeの8bit）を指定するのが一般的

⑤パリティチェック

パリティチェック(parity Check)はコンピューターでも間違うことがあるので冗長度（無駄）をもたせて誤りを防ぐ工夫
1byte（8bit）に含まれる「1」の数が偶数個か奇数個かで統一させ最後にparity bitとして追加

奇数パリティ（奇パリ）：1のbitが偶数個なら最後のparity bitを1にして1000 0010→1000 0010 1 奇数個にする

偶数パリティ（偶パリ）：1のbitが偶数個なら最後のparity bitを0にして1000 0010→1000 0010 0 偶数個のまま

8bitのデータが9bitになる（サーバーやワークステーションなどXeonプロセッサー搭載のPC等が使用）
通信ではCRCチェック（巡回冗長検査）が行われている
効率を追求するコンピュータにも無駄がいるんですよ!!

⑥ 2進数の演算

加算

```
   1101
 + 1001
  10110
```

減算

```
  10110
 − 1001
   1101
```

乗算

```
   1011   ←被乗数
 ×  101
   1011
  0000↵   } ←被乗数の桁をシフトして加算
 1011↵
 110111
```

除算

```
        1011
 101 ⟌ 110111
     −101
       01111
      −00000
        1111
       −101
         101
         −101
```

補数で減算を加算にする

01001 - - - ＋1001
↖正

10111 - - - −0111
↖負

真補数(2の補数)の求め方
(加算すると桁上がりする数)

```
 100000 ←基準数値
−  01001 ←＋1001
   10111 ←−0111
```

電子工学

```
 01001
   ↓      各桁を反転
 10110←1の補数(偽補数)
+    1   +1
 10111←2の補数(真補数)
```

基準数値を11111(1の補数)にすると0の補数が11111と重なるので＋1して2の補数(基準100000)を使用

```
  10110   ←引かれる数
 ＋10111  ←引く数(2の補数)
 101101   ←答01101
 ↖桁上がり:引かれる数>引く数 で無視
```

引かれる数 / 2の補数 / 答

```
  01101   ←引かれる数
 ＋01010  ←引く数(10110の2の補数)
  10111   ←答−01001
 ↖桁上がり無:引かれる数<引く数 なので 2の補数=01001
```

引かれる数 / 2の補数 / 答:2の補数 ← もう一度2の補数を求めて元に戻す

結論「コンピュータ(CPU)は加算しかできない!!」
コンピュータは繰り返しの計算が得意
ひたすら“足し算”している
補数を使った演算は符号付2進数として用いられる
4bitの例として最上位ビット(MSB : Most Significant Bit)が符号です
MSB＝1 の時、負となります
表現可能な数値は、最小値 −8 から最大値 ＋7 です
符号まで含めて演算ができる

〔4Bit符号付2進数〕

Bit 数値	MSB 符号	4	2	LSB 1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1

負 ↑ ― ↓ 正

⑦ 半導体

エレクトロニクスは半導体で成り立っている

材料のシリコン(**Si**)は絶縁体に近いが不純物を混ぜることにより半導体となる

不純物(**In**:インジウム)の価電子がシリコンより1個少ない場合ホール(正孔:⊕電荷)ができP(Posi)型半導体となる

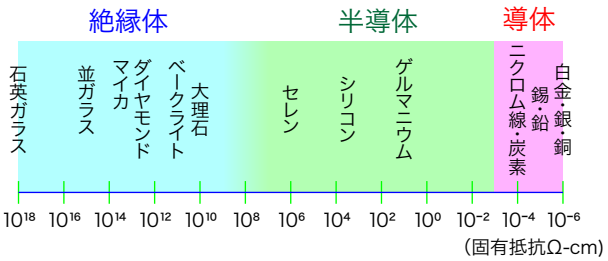
不純物(**As**:ヒ素)の価電子が1個多い場合は過剰電子(⊖電荷)ができN(Nega)型半導体となる

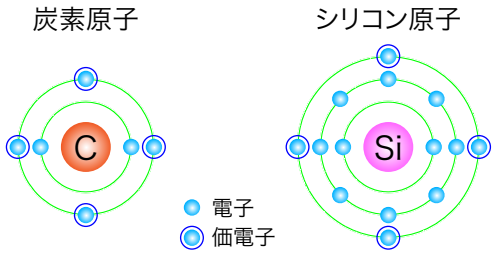
電荷を運ぶホールや過剰電子をキャリアという

電子は電流とは反対の方向に移動する
電子の移動が電流の流れである

温度が上昇すると金属の抵抗値は増加するが、半導体の電気抵抗値は下がる

光、電界、磁界によって電流が変化する





電子軌道は原子核を中心に内側から第1軌道、第2軌道とすると各軌道の電子の最大数は$2n^2$である
最も外軌道の電子(価電子)は、ある程度の熱、光、電界エネルギーを与えると移動し始める

PN接合ダイオード(Diode)は順方向にだけ電流を流す

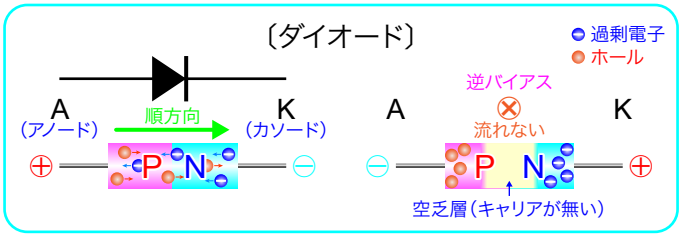
トランジスタ(Transistor)の基本は信号の増幅である
小さなベース電流の変化量に応じて電気伝導率が変わり大きなコレクタ電流が流れる

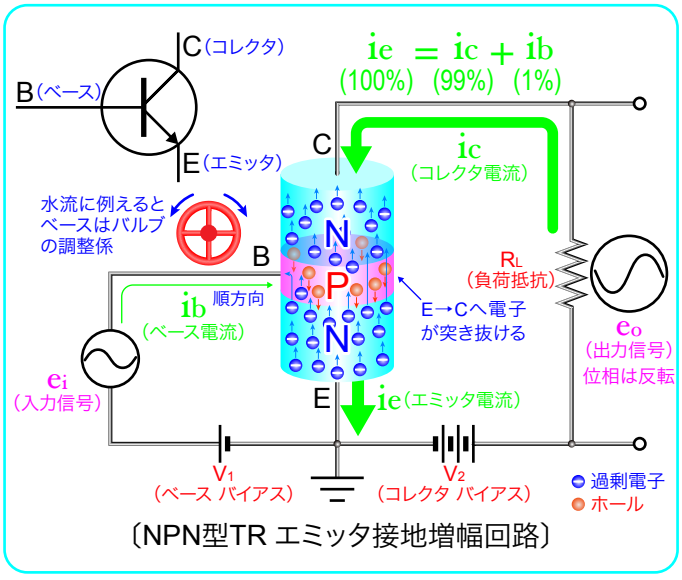
例えば小さな音楽信号を入力すれば大きな音楽信号として出力される

デジタル回路においては、増幅では無くスイッチとして使用する

入力信号としてベース電圧がH(1)になる=ベース電流が流れる➡コレクタ電流が流れる=コレクタ電圧がL(0)が出力信号となるインバーター(反転増幅器)である

デジタル回路はIC化されダイオードやトランジスタを素子単体として見ることは殆ど無い

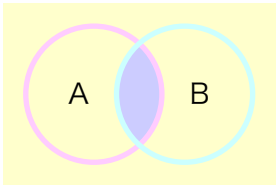




⑧ ブール代数と電子回路

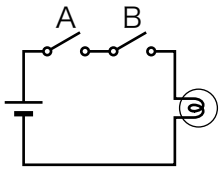
ブール代数とは、命題の真を“1”偽を“0”として論理演算する集合論

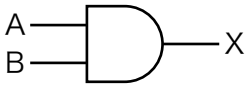
論理積(A∗B)



〔ベン図〕

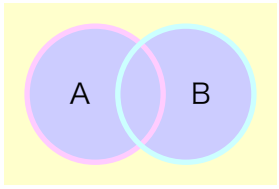
A∗B	0	1
0	0	0
1	0	1



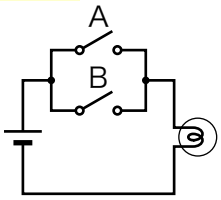


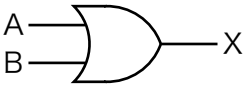
AND 回路

論理和(A+B)



A+B	0	1
0	0	1
1	1	1

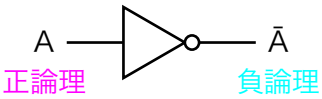


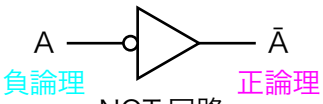


OR 回路

否定

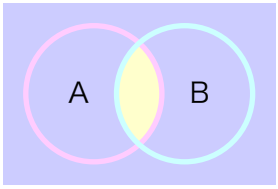
A	Ā
0	1
1	0



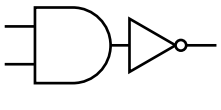


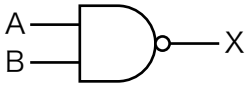
NOT 回路

否定論理積$\overline{(A*B)}$



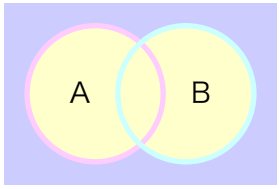
A∗B	0	1
0	1	1
1	1	0



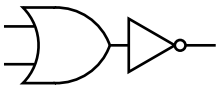


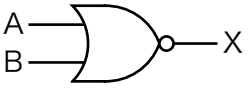
NAND 回路

否定論理和$\overline{(A+B)}$



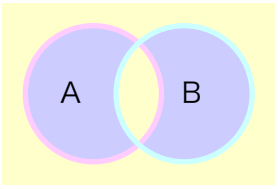
A+B	0	1
0	1	0
1	0	0



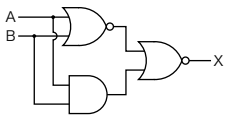


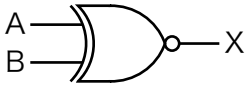
NOR 回路

排他的論理和(A∗$\bar{B}$)+($\bar{A}$∗B)



A+B	0	1
0	0	1
1	1	0



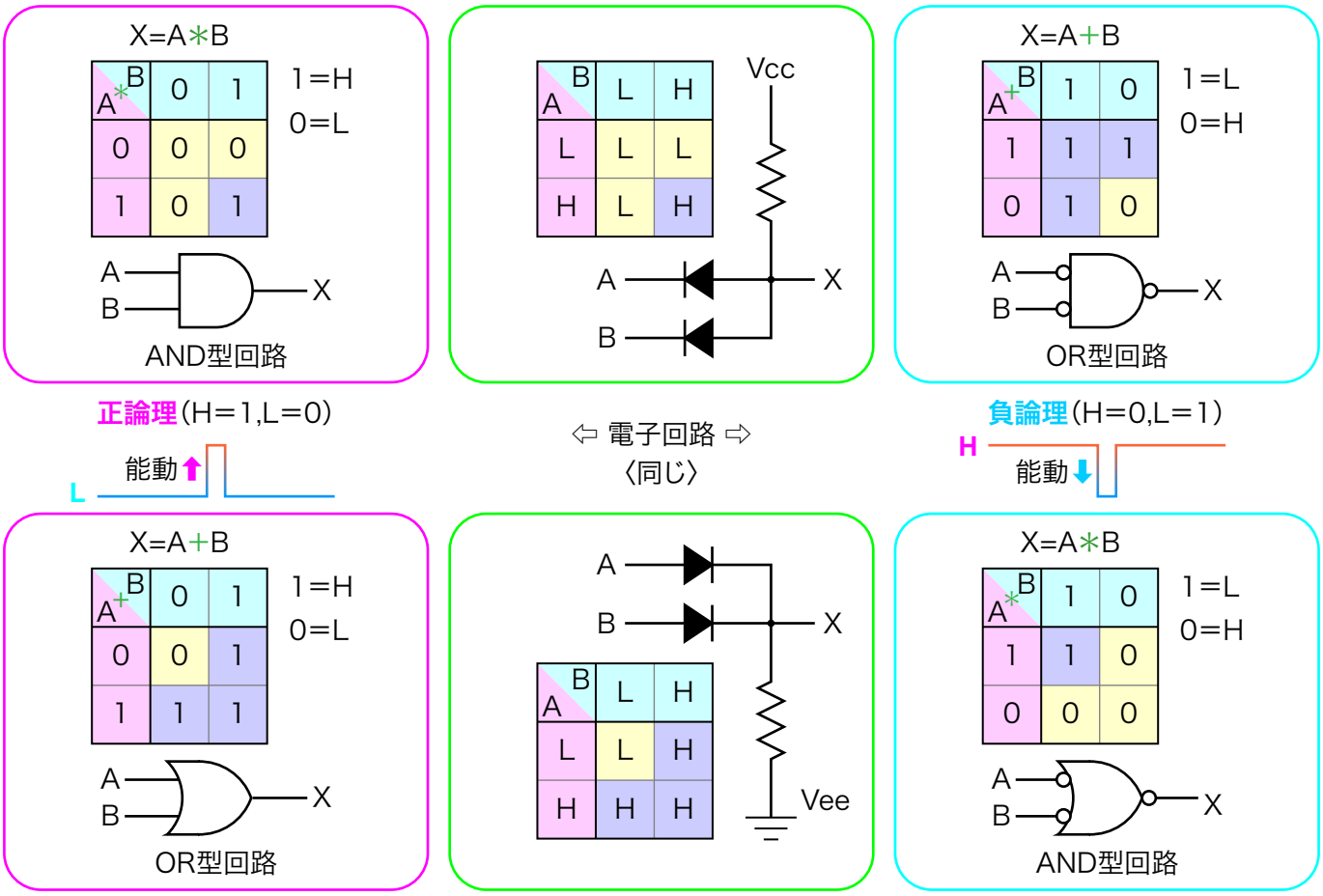


EXOR 回路

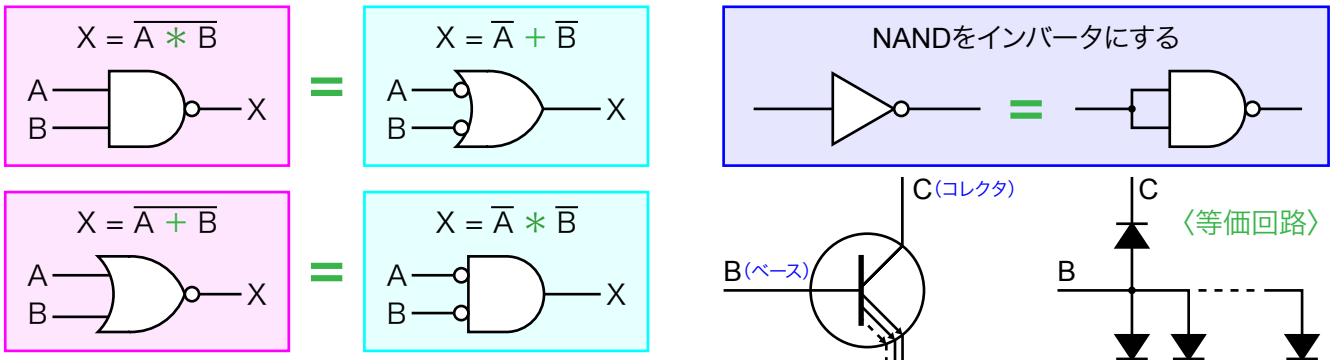
※電子回路図では○と○負論理同士接続する様に表記する(整合性)

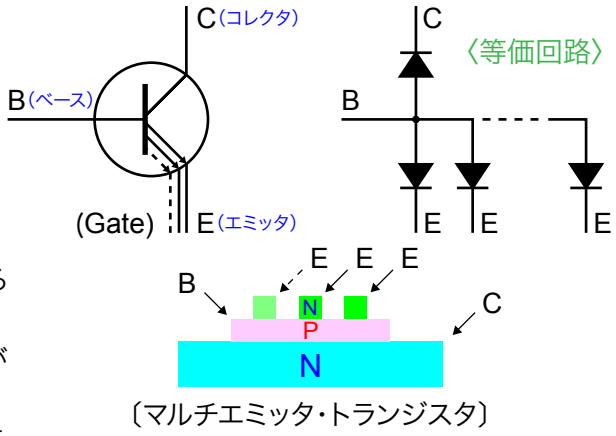
交 換 則	$A \cap B = B \cap A$ $A \cup B = B \cup A$	$A * B = B * A$ $A + B = B + A$
吸 収 則	$A \cap R = A$ $A \cap \varnothing = \varnothing$ $A \cup R = R$ $A \cup \varnothing = A$	$A * 1 = A$ $A * 0 = 0$ $A + 1 = 1$ $A + 0 = A$
結 合 則	$(A \cap B) \cap C = A \cap B \cap C$ $(A \cup B) \cup C = A \cup B \cup C$	$(A * B) * C = A * B * C$ $(A + B) + C = A + B + C$
分 配 則	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A(B + C) = A * B + A * C$ $A + (B * C) = (A + B)(A + C)$
ド･モルガン の 法 則	$\overline{A \cap B} = \bar{A} \cup \bar{B}$ $\overline{A \cup B} = \bar{A} \cap \bar{B}$	$\overline{A * B} = \bar{A} + \bar{B}$ $\overline{A + B} = \bar{A} * \bar{B}$

AND回路は 1＝H、0＝L の時 A＊B が成立したが、逆に 1＝L、0＝H に対応させると A＋B のOR回路になる
電子回路が同じでも正論理と負論理でAND回路とOR回路が入れ替わる



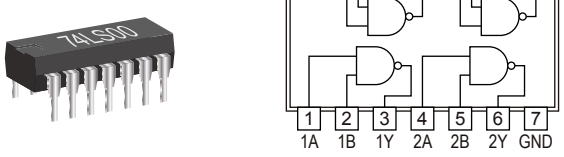
ANDなのかORなのかはMIL(Military Standard)規格で入出力に○を付ければ負論理としている
正確にはMIL規格は正・負論理ではなく L の時に能動な入出力の場合○を付ける
ド・モルガンの法則によりNAND回路とNOR回路を入れ換える
論理回路はNAND（またはNOR）のみで構成できる





〔マルチエミッタ・トランジスタ〕

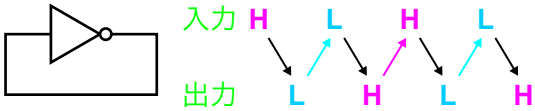
Digital回路の基本はインバータ（反転増幅器）である
ダイオードでのANDは増幅作用が無いので3～4段つなげると H L の判別ができなくなる
対策としてAND回路の後にインバータを入れるNAND回路が作られDTL(Diode-Transistor-Logic)となった
TTL(Transistor-Transistor-Logic)はDTLのゲート側のダイオードをマルチエミッタトランジスタに換えることで素子数を減らせた
TI(テキサス・インスツルメンツ)社の74シリーズが基準で**無印**：スタンダード、**H**：ハイスピード、**L**：ローパワー→**S**：ショットキー→**LS**：ローパワーショットキーと進化80年代にはHとLは淘汰されLSが主流であった
余談だがUSB端子はTTL-Level(+5V)を引き継いでいる



〔TTL-ICの例 74LS00 Quad 2Input NAND〕

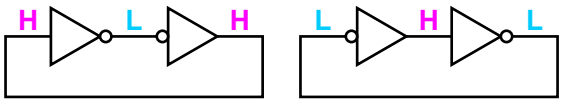
⑨ フリップ フロップ

ゲート回路は入力の状態に応じ回路の動作時間の遅延後、出力状態が決まる
出力を入力に戻さない限り X = f (A , B , C・・・・・・) で表される出力が得られるのを組み合わせ回路という
インバータの出力を入力に戻す(フィードバック)場合Hが入力されるとLが出力されLが入力に戻りHが出力される
出力はインバータの遅延時間の2倍の周期で発振する発振器(ロジカルオッシレータ)となる



〔ロジカルオッシレータ〕

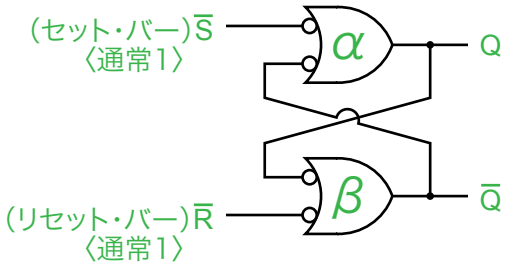
X = f (X , A , B , C・・・・・・)と自分の現在の出力Xによって決まる回路を順序回路という
インバータを偶数個接続すると矛盾が無く安定する
安定状態が2つある回路は記憶させることができる
このような回路をフリップフロップという
2連インバータでは指定した状態にセットできないのでゲート回路を接続してフィードバックループを切りコントロールする



〔2連インバータ〕

◎RSフリップ フロップで記憶

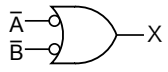
RS(リセット・セット)FF(フリップ フロップ)は、通常$\overline{S}$,$\overline{R}$は“1”である
❶Q=“0”,$\overline{Q}$ =“1”でスタート
$\overline{S}$=“0”にするとαの出力Q=“1”→βの入力“1,1”→$\overline{Q}$=“0”❷で安定
再び$\overline{S}$=“1”にするとαの入力=“0,0”→“1,0”→Q=“1”のまま❸で安定
$\overline{R}$を“0”にするとβの出力$\overline{Q}$=“1”→αの入力“1,1”→Q=“0”❹で安定
$\overline{R}$を“1”にすると❶の状態に戻る



〔RSフリップフロップ〕

ここで$\overline{S}$,$\overline{R}$を同時に“0,0”にするとQ,$\overline{Q}$=“1,1”になる(ゲートαβとも入力が“0”になり“off”になる)
続けて$\overline{S}$,$\overline{R}$を同時に“1,1”に戻すとQ,$\overline{Q}$は“0,1”になるか“1,0”になるか判からない欠点がある
従って入力を同時に“0”にした後、続けて同時に“1”にすることは禁止

思考の参考

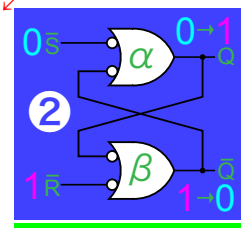


〔NOR〕

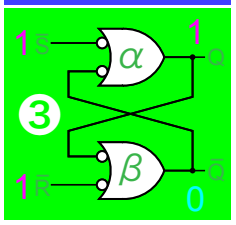
$\overline{A}$ ⁺ \ $\overline{B}$	0	1
0	1	1
1	1	0

$\overline{S}\overline{R}$ \ $Q\overline{Q}$	00	01	11	10
00	×	11	11	11
01	×	11	10	❷10
11	×	❶01	??	❸10
10	×	❹01	01	11

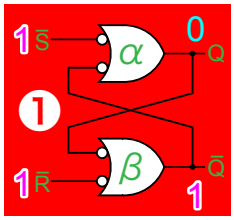
許されない入力



〈安定〉



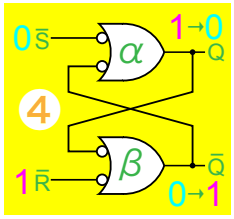
〈安定〉



〈安定〉

〔RSフリップ フロップのカルノー図〕

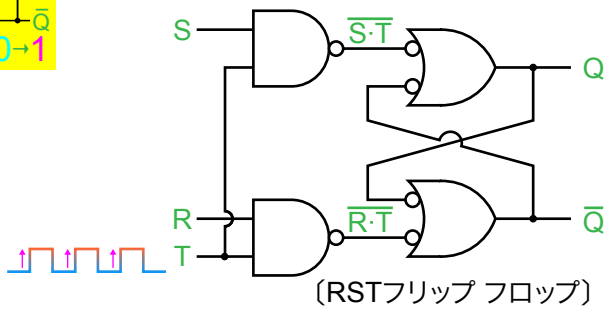
カルノー図とは真理値表で隣り合う変数が1つ変化(ハミング距離が1)する論理空間での配置図



〈安定〉

◎RST フリップ フロップ

RS FFを一般的なシステム・パルスの立ち上がりに同期させる
RST FFに改造
SとRが同時に“1”の時にTが加わると出力状態が不明になるのでJK FFに改良してIC化されている



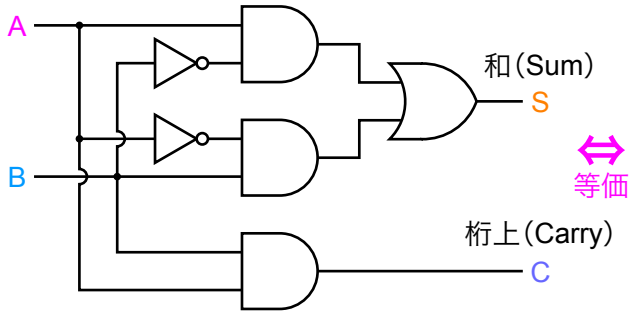
〔RSTフリップ フロップ〕

⑩ Adder（加算器）

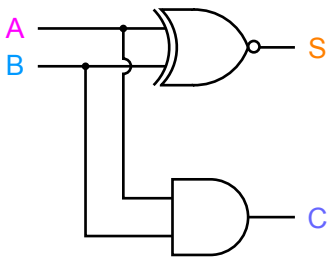
2進数の加算

AとBの和を求める回路

下からの桁上がり（キャリー）が加算できず不完全なので半加算器(Harf Adder)という



〔半加算器〕



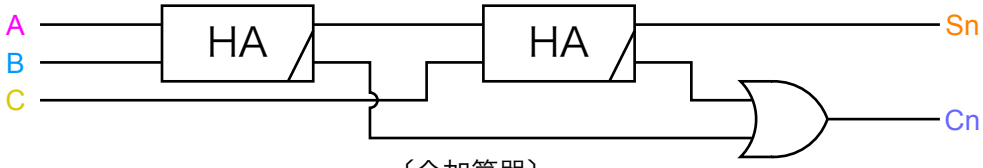
〔EXORによる半加算器〕

入力		和	桁上
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = A * \bar{B} + \bar{A} * B = A + B$

$C = A * B$

全加算器(Full Adder)はAとBの和と下からの桁上がりCが加算できる



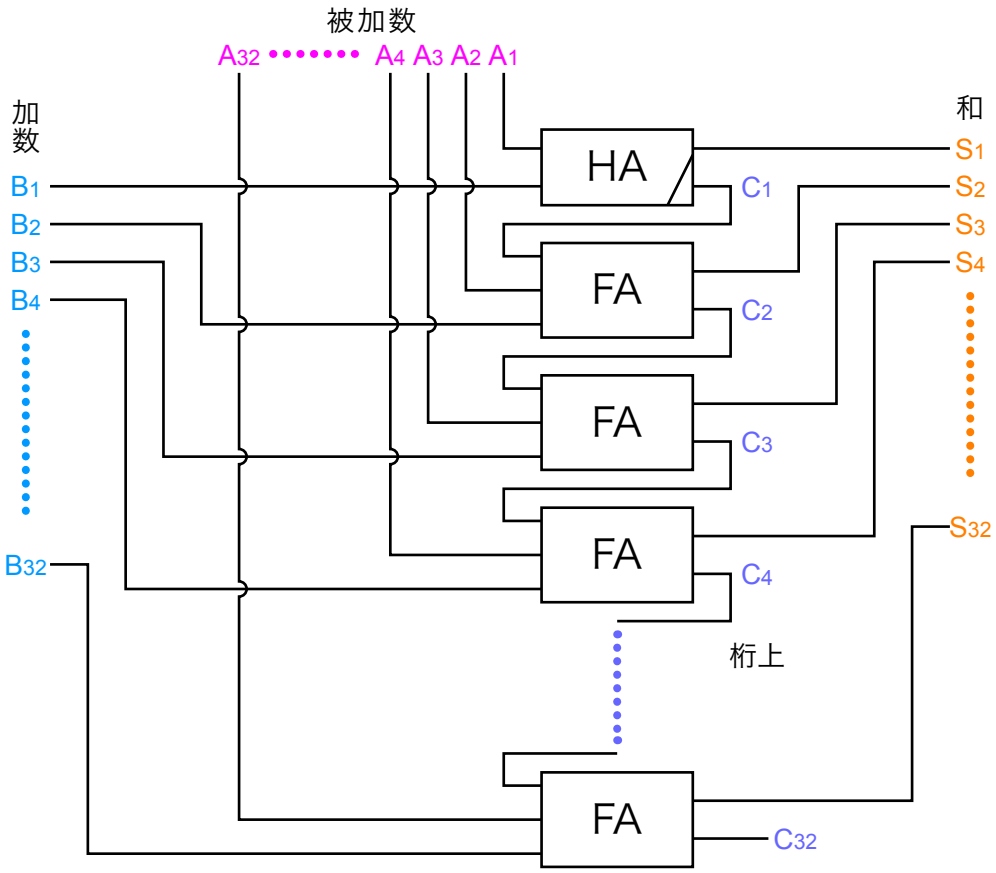
〔全加算器〕

入　力			和	桁上
A	B	C	Sn	Cn
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

$Sn = A * B * C + \bar{A} * \bar{B} * C + \bar{A} * B * \bar{C} + A * \bar{B} * \bar{C}$

$Cn = A * B + B * C + C * A$

i386のCPU内部には32個(32bit)の加算器を並べた並列加算器がある



符号付の演算では符号(MSB)＝1が負数を表す

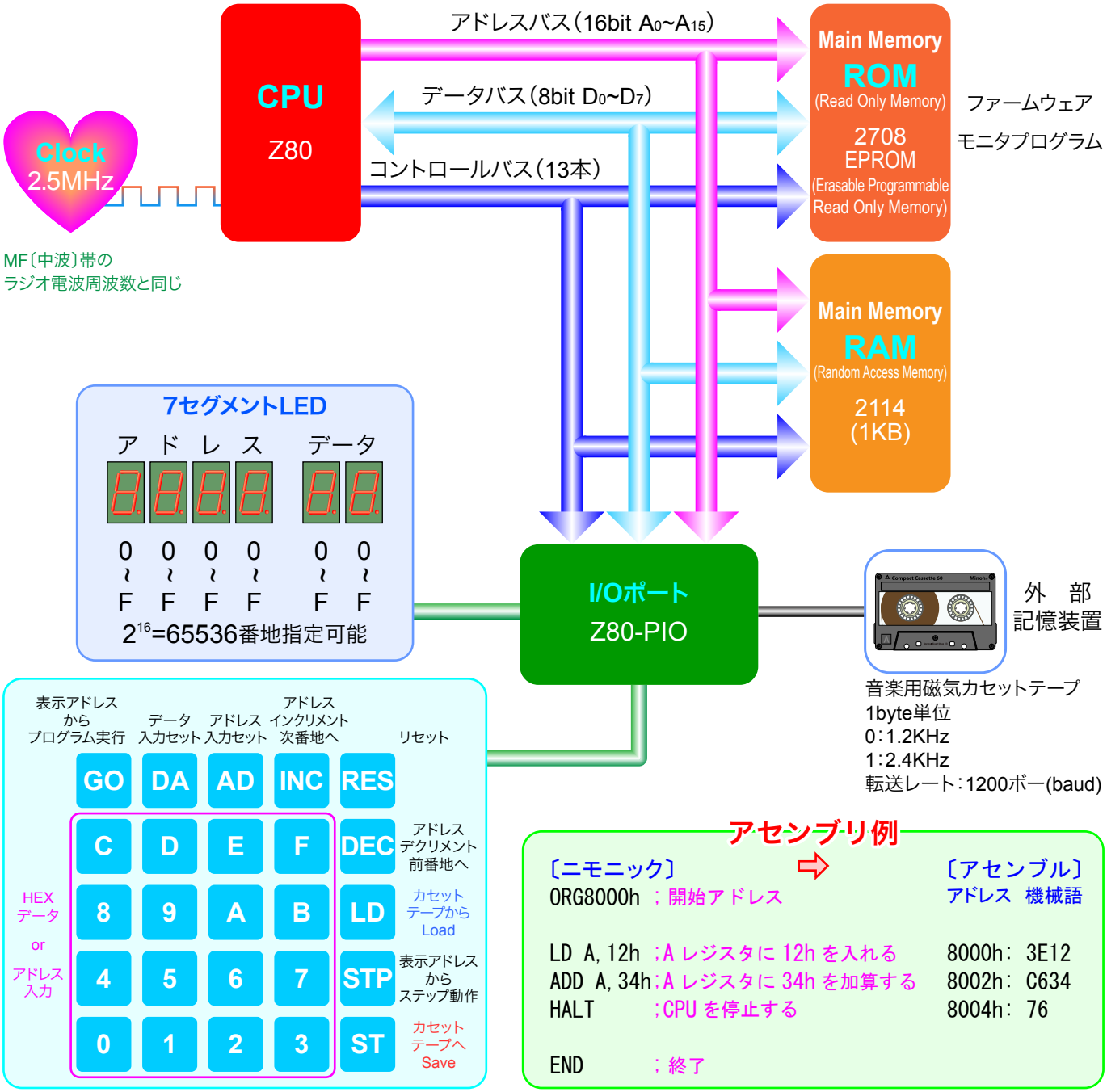
オーバーフローは
符号＝1 かつ キャリー＝0

アンダーフローは
符号＝0 かつ キャリー＝1
で検出する

CPUではフラグ(Flag)レジスタが監視している

ALUの中には加算器を中心に桁をずらすシフタ、入力を選択するマルチプレクサなどがある

⑪ マイコンの構造（Z80 CPU）

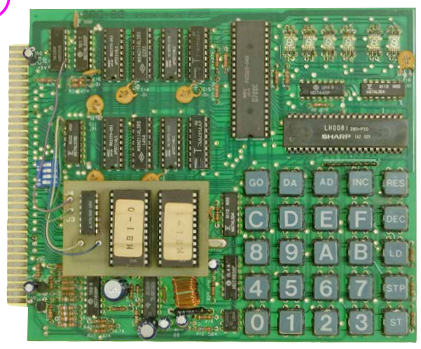


アセンブリ例

〔ニモニック〕		〔アセンブル〕
		アドレス　機械語
ORG8000h	；開始アドレス	
LD A, 12h	;A レジスタに 12h を入れる	8000h：3E12
ADD A, 34h	;A レジスタに 34h を加算する	8002h：C634
HALT	;CPU を停止する	8004h：76
END	；終了	

バス(Bus)はCPUとメインメモリ間などのデータ伝送路
8bit、16bit、32bit、64bitとバス幅（信号線の本数）がCPUの一度に扱えるbit数（2^）で決定される
Z80の場合

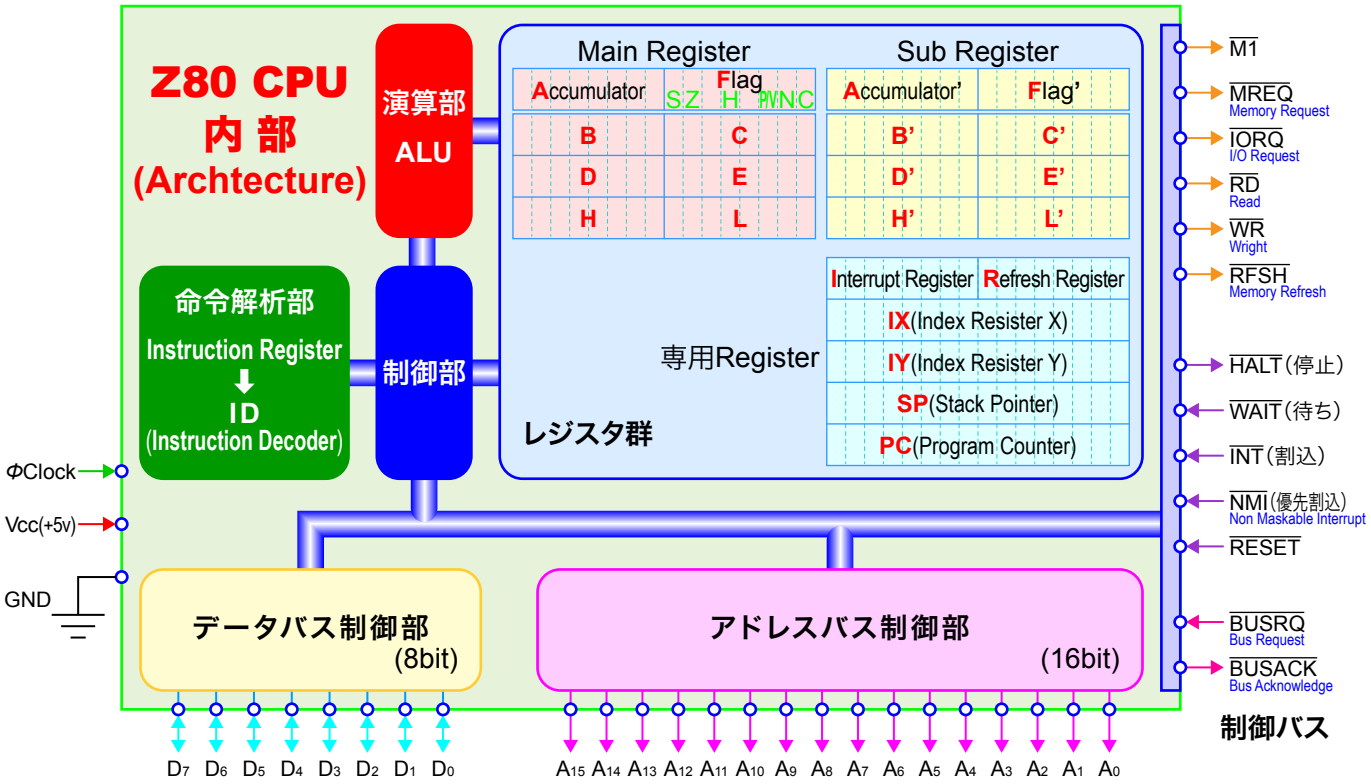
アドレス　バス：番地を指定〔16bit（65536番地扱える）〕
データー　バス：データ本体の伝送〔8bit（一度に8桁処理できる）〕
コントロール バス：全体を制御
動作周波数は2.5MHz（スピード）



◎基本動作

❶電源ON → ❷Reset → EP-ROMに書き込まれたモニタ プログラムが起動

- ◎ 7セグメントLEDの表示
- ◎ メモリーにデータを書き込み読み出し
- ◎ ユーザープログラムの実行
- ◎ ブレークポイントの設定
- ◎ ステップ動作
- ◎ キー入力
- ◎ カセットテープへデータを録音/テープからデータを転送（FAX/MODEMで生き残っている）



◉ Z80内部にはレジスタと呼ばれるメモリがある

A(アキュムレータ)は演算に使用

F(フラグ)は演算結果の状態(S[Sign]:負=1;Z[Zero]:零=1;H[HerfCarry]:3bit目の桁上(BCD時)=1;P/V[Parity/Overfllow]
符号付整数時Carry:桁上=1//AND,OR等命令結果が偶数=1;N[Subtract]:減算=1;C[Carry]:桁上=1)

I(インターラプトレジスタ)は割り込み処理

R(リフレッシュレジスタ)はリセット後フェッチサイクル毎に1増えてRAMのリフレッシュアドレスを指定

IX,**IY**(インデックスレジスタ)は先頭からの相対位置を記憶

SP(スタックポインタ)はサブルーチン(副プログラム)へ飛んだ後の戻り番地を記憶

PC(プログラムカウンタ)はプログラムの実行順番を数えている

◉ マシンサイクル(基本動作)

•フェッチ:メモリに書かれているオペコード(命令)をインストラクションレジスタに読込みデコーダで解読

•メモリ・リード:メモリーからCPU内のレジスタに読込む

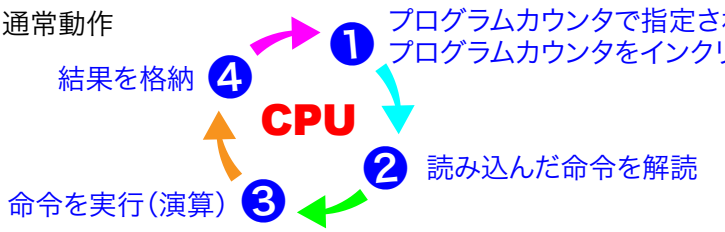
•メモリ・ライト:CPU内のレジスタからメモリに書出す

•I/Oリード:入力ポートからデータをCPU内のレジスタに読込む

•I/Oライト:CPU内のレジスタから出力ポートへデータを書出す

•リフレッシュ:DRAMをリフレッシュするためのアドレス信号を出す

◉ 繰り返される通常動作



機械に組み込んで制御する用途は別として、パソコンとしての使い道は皆無で少ないメモリ空間を上手に節約して何をさせようか?結局ゲーム?の代物だったマイコンです
7セグメントLEDにアドレスを表示し(Inc)(Dec)で16進のマシン語プログラムとデータを打ち込んでいた
当時のマイコンボードにはアセンブラすら用意されておらずハンドアセンブル(手動で機械語化)するかマイコン誌に掲載されていたサンプル機械語をひたすら打ち込むのが大方では無かったでしょうか?

⑫ パーソナルコンピュータ

当初マイコンと略されたマイクロコンピュータは個人で所有できる My Computer の意味もあった
徐々にキーボードやモニターのついたものはパーソナルコンピュータと呼ばれ、機器に組み込まれる場合（マイコン炊飯器、マイコン内蔵～）はマイクロコンピュータと呼び分けるようになった
関数電卓、電子手帳、ミニコン、オフコン(Office Computer)、ワープロ専用機は駆逐された

日本では32bitが主流となったCPU(Central Processing Unit)がMPU(Micro Processing Unit)と呼ばれるようになり補助記憶装置もFDD（フロッピー ディスク ドライブ）からHDD（ハード ディスク ドライブ）に移行しBASIC等で自前でプログラムする事も無くMS-DOS上にアプリケーション（一太郎やロータス123等）をインストールして利用するスタイルを経てWindows3.1の登場でパソコンという呼び名は定着したと思う

PCはパソコン通信からインターネットへと生活に密着していく一方CGや音楽でもプロフェッショナル仕様となり由も悪しきもあらゆる産業に影響していく

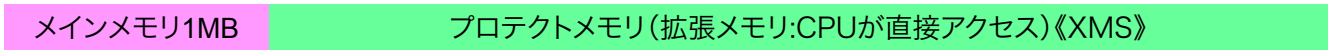
⑬ メモリ管理の歴史

8086のメモリ1MB（1024KB）

MicrosoftのWindowsは過去の互換性を大事にした
16Bit機以降のメモリ（RAM）管理は苦労されている
IBM-PCに採用されていた86系CPU8086（16bit）は2^20=1MB（1024KB）のメモリ空間にメイン（コンベンショナル）メモリ（640KB）とI/Oアドレス空間（384KB）＝アッパーメモリ：BIOS〈Basic Input Output System〉が格納されていてメモリ空間をフルに使用できなかったのでバンク切替やEMSで拡張して1MBの壁を克服した

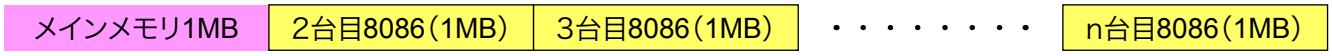
80286（16bit）になりプロテクトモードで16MBまで拡張された（8086時代の1MBはリアルモードと呼ばれた）

80286のメモリ〈プロテクトモード〉最大16MB



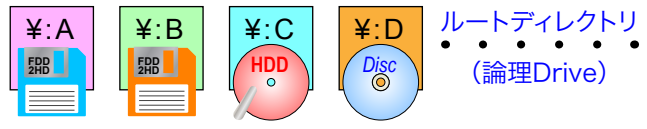
80386（32bit）以降8086用のプログラムを実行させるためプロテクトメモリを1MBづつ区切り複数の8086に見せかける仮想86モード

80386以降のメモリ〈仮想86モード〉

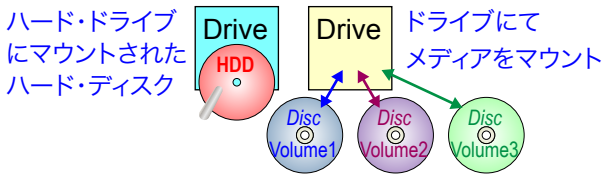


一方68系のMacintoshは相対アドレスで32bitメモリー空間を機種により最大容量は異なるが自由に使えた（メモリは非常に高価だった）画像のような大きなDataはMacでないと扱えなかった、、、が漢字Talkはよくフリーズした

世界が熱狂したWindows95になっても16bitを大事にしたMicrosoft社
外部記憶装置の論理ドライブ¥A：FDD、¥B：FDD であるのはデュアルフロッピー機の名残である
アップルはレガシーな環境はバッサリ（SCSI,ADB,RS-232C,IEEE-1394FireWire,DVI-D,etc）
Macはドライブではなくボリューム単位でマウントさせる



ABは物理Driveだよね



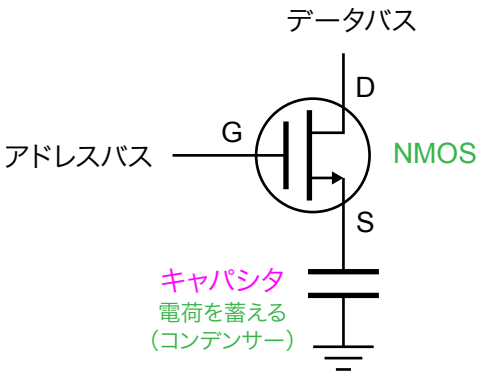
Windows XPは非常に安定したOSであった
32bit版 Windows OS は物理アドレス空間の先頭から4GBまでの領域にしかアクセスできないため I/Oアドレス が4GB未満の領域になるように配置されているもののメモリ空間は解放され事務用途から脱皮したWindowsとIntel連合でWintel時代の黄金期である

MPUが64bitになり8GBや16GBのRAM容量が当たり前でメモリマップの工夫も必要なくなった
外部記憶装置もDataの肥大化と共に大容量化しているが、トラブル時のダメージの大きさと、そのメディアの規格がいつまで使用できるかも重要だ
セキュリティ リスクの問題もあるがクラウド環境と物理メディアを使い分けるべきか？

⑭ メインメモリ(RAM)

マザーボードに実装されるメインメモリは、フリップフロップを集積したスタティックメモリ(SRAM)から始まった
高速で省エネではあったがbitあたりNAND回路が2個必要なため集積度とコストでCPU内部の使用に停まる

ダイナミックメモリ(DRAM)はキャパシタに電荷を貯めて記憶する
bitあたりNMOSとキャパシタのペアであるため集積度が上げやすいが
放電され(忘れ)ない内にリフレッシュ(書直し)しないといけないので
スタティックメモリに比べて消費電力は多い
CPUの高速化に伴い書込み読み出しを別サイクルで行っていたものを1サイクルにしたDDR(Double Data Rate)化

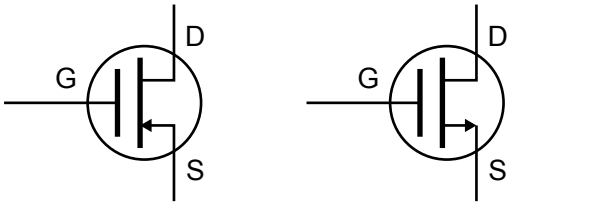


〔ダイナミックメモリ〕

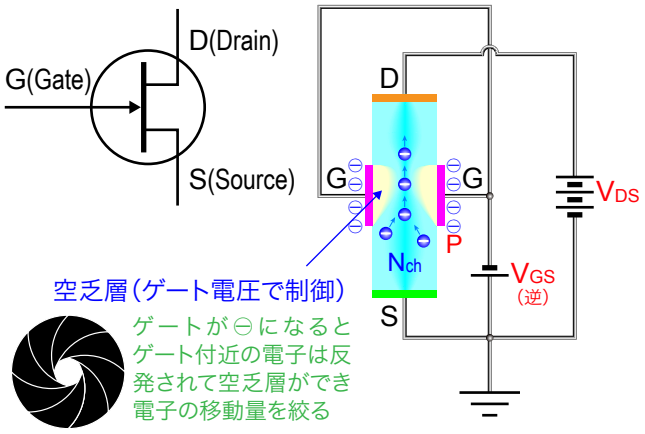
DDR2ではメモリ内部のクロックをバスの1/2に逓倍し処理を並列化することで高速化、さらに逓倍化でDDR3,DDR4と進化している
GraphicBordにはDDR5,DDR6を実装しているので画面描画していない時にGPGPU(General-Purpose computing on Graphics Processing Units)でCPUの演算を助けるGPU搭載マシンもある

◎電界効果トランジスタ(FET)

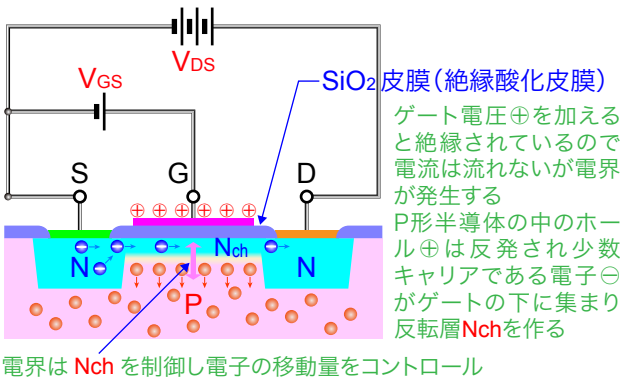
FET (Field-Effect Transistor)には接合型とMOS型
(Metal-Oxide-Semiconductor Field-Effect Transistor)の2種がある



〔Pチャンネル(PMOS)〕 〔Nチャンネル(NMOS)〕



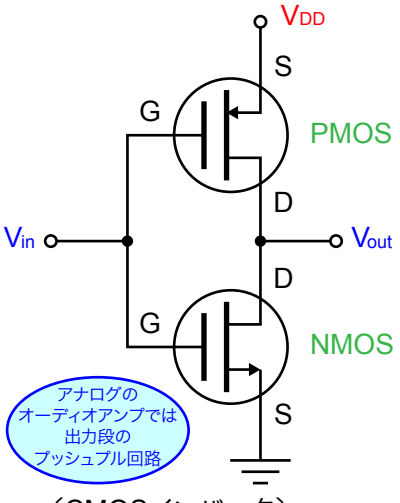
〔Nチャンネル接合型〕



〔NMOS型〕

電荷を運ぶキャリアはトランジスタ同様 P型がホールで N型は電子
キャリアの通路をチャンネル(ch)と言いゲート電圧で発生する電界で流量を制御する
バイポーラトランジスタが電流で電流を制御する電流制御型に対し FET は電圧で電流を制御する電圧制御型
入力インピーダンスが高い(接合型:10^{7}～10^{11}、MOS型:10^{10}～$10^{14}\Omega$)反面、静電気に弱い
高速スイッチ特性が良い、特に MOS型 は半導体表面近くの電荷を増大あるいは空乏できるのでゲートバイアスが0v時にドレイン電流を流せるデプレンション型と流さない(カットオフ)エンハンスメント型がある

CMOS(Complementary-Metal-Oxide-Semiconductor Field-Effect Transistor)は PMOSとNMOS を組み合わせた回路である
右図のインバータでは入力電圧に対しスイッチの動作は逆になるので PMOSがON の時は NMOS は OFF となり、PMOS が OFF の時は NMOS は ON となる入力が変化する時だけ電流が流れ保持状態では原理的に電力を消費しない
動作電圧幅が広く雑音余裕度が高い
反面、静電対策で導電スポンジに足をさしアルミ箔で包み持ち運び、組立作業にはリークしない半田コテを使用等取扱いには神経を使った
RCA社の4000シリーズとNSC社の74Cシリーズに始まりモトローラ社の4500シリーズと続いた
デジタル LSI は CMOS でできている2000年頃にCPUもCMOS化された



〔CMOSインバータ〕

⑮ 外部記憶装置

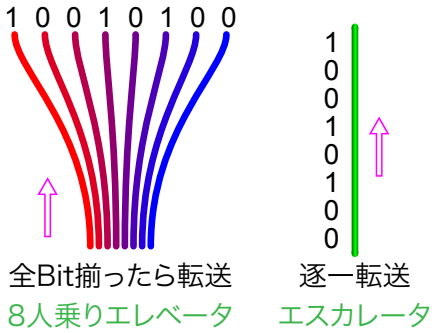
フロッピーディスクは8in→5in→3.5inとなりPCの主役であった(¥A: ¥B:に未だ鎮座している)
IBM 3340で採用されたウィンチェスターディスクがハードディスクドライブ(HDD)として普及する

接続方法は SASI (Shugart Associates System Interface) を元に❶SCSI (Small Computer System Interface)が ANSI (American National Standards Institute)で最初に規格が標準化されたパラレル接続による
デジーチェインでHDD、MO、スキャナー、プリンタ等を接続して使用した

❷IDE(Integrated Drive Electronics)を1994年に ANSI(米国国家規格協会)がATA-1で標準化したパラレル接続に移行

RS-232Cをルーツに持つシリアル転送はUSB以降スピードが上がりPC内も❸SATAでフラットケーブルも無くしでシリアル転送方式に変更した

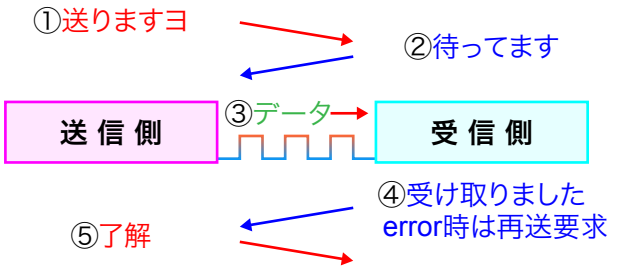
SSD (Solid State Drive)では❹NVMe(Non-Volatile Memory Express)接続に移行



⑯ 通信

コンピューター内部ではクロックに同期して動作しているが外部装置とデータのやりとりは送信側と受信側が合図をしてハンドシェイクで行われる

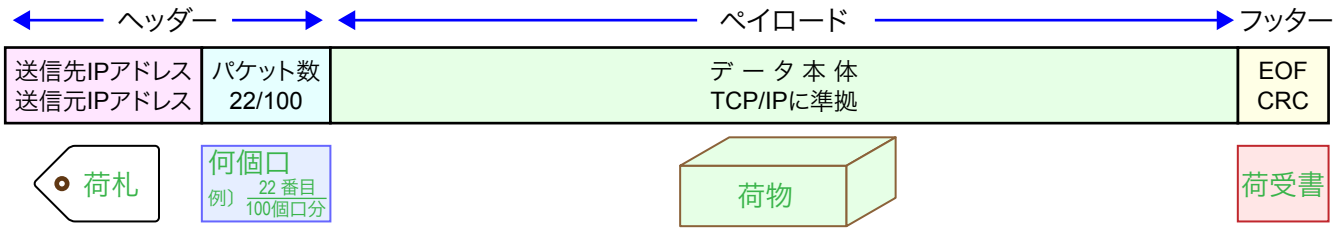
ネットワークの時代においてはデータをパケット(packet)という荷札付の単位に分割して伝送される
分割されたパケットは経路を問わず受信側に届けられ元通りの順序に並びデータが復元される



〔ハンドシェイク通信〕

ローカルエリアでの有線接続LANはイーサネット(Ethernet)で構築される10BaseTで普及し100BaseT、1000BaseTと高速化し10GBaseへと帯域幅を広げより高速化されている
また、パケットのデータ(ペイロード)を送信するデータ構造は「フレーム」といい最大1500byteであるが、より大量のデータを送信できるようにフレームの規格を拡張(ジャンボ・フレーム)した高速化もされている

インターネットに代表されるWAN(Wide Area Network)となるとTCP/IP(Transmission Control Protocol / Internet Protocol)での通信となりパケットもTCP/IPに準拠する



〔インターネット パケット〕

⑰ 起動

パソコンがパワーオンされるとマザーボード上の不揮発性メモリ(EP-ROM)に書込まれているBIOS(Basic Input/Output System)と呼ばれるファームウェアが起動し接続されている周辺機器や組み込まれているパーツのチェック後、使用できる状態に準備させ外部記憶装置にインストールされているOSを読込み起動させる

初期のマザーボードはCPUとペリフェラルと呼ばれた周辺LSIを実装し拡張カードで機能を持たせていた
Core 2 時代にはチップセットとしてノースブリッジ(MCH:メモリやGPU制御)とサウスブリッジ(ICH:HDDやUSB制御)が搭載されていた
Core i の登場でノースブリッジの一部機能はCPUに内蔵され第二世代でチップセットが1つになり、さらに第三世代からグラフィック機能もCPU内蔵化が進んでいる

一般的に機械は、構成部品が減ると故障率が下がり、さらに駆動部の無いSolid state化によって軽量化、省電力化、振動に強くなる反面、大容量ストレージのトラブルは被害が甚大である

PCに接続されるマウスやキーボードを含め周辺機器には必ずCPUが搭載されていて相互に通信するのだがメインとなるPC本体のCPUが一番注目される

⑱ プログラム言語

2進数の機械語は1対1で対応するアセンブリ言語（人間にわかりやすいニーモニックと呼ばれる命令セットでオペコード《命令》とオペランド《変数、定数、番地》の組み合わせ）になる
アセンブリ言語→アセンブラというソフトでアセンブリ（機械語に変換）し実行→デバッグ（修正）を繰り返す

1954年にIBM-704型のプログラム言語としてFORTRAN（Formula Translation：数式翻訳に由来）開発
行番号参照型で科学技術計算（浮動小数点計算）に適し “IF文” “GOTO文” “CONTINUE文” 等、後のBAISIC言語に影響している

1959年に事務処理用のCOBOL（Common Business Oriented Language：共通事務処理用言語に由来）開発

1964年にBASIC（Beginner's All-purpose Symbolic Instruction Code：初心者向け汎用記号命令コード）が米国ダートマス大学でコンパイラとして開発されマイクロソフトがインタプリタ（命令語を実行毎に機械語に翻訳しながらプログラムを進行）として提供し8bitマイコン時代にROMとして標準装備されたが、各社独自に開発させた結果MBASIC,N-BASIC,F-BASIC等、互換性の無いプログラムに進化した
実行速度は遅かったがRUNさせてSyntaxerrorでつまずきTRONでBug探しTry&Errorを繰り返すには簡便だったメモリー容量が64KBしか無いので節約プログラムの挑戦でもあった
行番号参照型なのでGOTO^文でスパゲティー状になるのをWHILE~WENDに書き直すもストラクチャー（構造化）プラグラムにはほど遠かった...が楽しめた

C言語は1972年にAT&Tベル研究所でUNIXのOS上のプログラミング言語として開発されたB言語（インタプリタ型）の改良版がC言語（処理の流れに沿ってプログラムを書いていく手続き型プログラムでコンパイル型）
MS-DOSもC言語で書かれC++、Java、C#、Linuxと（オブジェクト指向化）分化していく（アップルはObjevtive C）

WebサーバーではPHP(Hypertext Preprocessor)で書かれていてブラウザーで操作される
目的によっての使い分けも必要だ

⑲ **OS**(Operating System)

キーボード入力、マウス操作、画面表示、保存、読込み・・・etc
入出力の制御、記憶装置の読み書き、ユーザーインターフェース、アプリケーションソフトウェアの動作、コピー、削除、複数のタスクの割り当てやプロセスの管理等を受け持ちワープロや表計算はその上に乗っかる形で走る
TSS（Time Sharing System：時分割＝マルチタスク、マルチユース）時代にOSが登場した

1964年　IBM System/360シリーズに搭載されたOS/360およびDOS/360がOSの始まり
1969年　UNIX（AT&Tのベル研究所）
1976年　CP/M（i8080用 Control Program for Microcomputers）
1977年　UCSD Pascal（カリフォルニア大学）
1981年　IBM PC DOS（x86系）1982年共同開発のマイクロソフトがIBM以外に「MS-DOS」としてOEM提供
1986年　FLEX（68系）　TRON（米国のスーパー301条に潰された幻の日の丸OS）
1990年　OS/2（IBM、マイクロソフト）
1991年　Linux
1992年　Windows 3.1(Microsoft)
1994年　Windows NT3.1(Microsoft)
1995年　Windows 95(32bit化)
1996年　Windows 4.0(Microsoft)
1997年　Mac OS 7.6(漢字Talk（日本語版）から初めてOSと表示）
2001年　Windows XP
　　Mac OS X
2009年　Windows 7
2015年　Windows 10

⑳ 近似数学

平方根を求めるプログラムではニュートンラフソン法が使われる

関数 $f(X)$ とその導関数 $f'(X)$ の値を求め
解に近く解より大きい初期値を与える必要がある

$X=\sqrt{a}\quad(a>0)$

$X^2=a$

$f(X)=X^2-a$ ・・・①

公式

$$X_{i+1}=X_i-\frac{f(X_i)}{f'(X_i)}$$

$f'(X)=2X$ ◀・・・①を微分(導関数)

$$X_{i+1}=X_i-\frac{X_i^2-a}{2X_i}$$

$$=X_i-\frac{1}{2}X_i+\frac{1}{2}\frac{a}{X_i}$$

$$=\frac{1}{2}\left(X_i+\frac{a}{X_i}\right)$$ ◀・・・繰り返す

例) $X=\sqrt{2}\quad(a>0)$

$X^2=2$

$f(X)=X^2-2$

$$X_{i+1}=\frac{1}{2}\left(X_i+\frac{a}{X_i}\right)$$

$f'(X)=\lim_{h\to0}\frac{f(X+h)-f(X)}{h}$

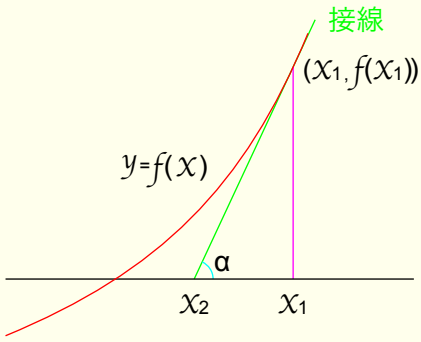
$=\lim_{h\to0}\frac{\{(X+h)^2-a\}-(X^2-a)}{h}$

$=\lim_{h\to0}\frac{(X^2+2hX+h^2-a)-(X^2-a)}{h}$

$=\lim_{h\to0}\frac{X^2+2hX+h^2-a-X^2+a}{h}$

$=2X$

《ニュートンラフソン法》



$\tan\alpha=f'(X_1)$

$$=\frac{f(X_1)}{X_1-X_2}$$

$$\therefore X_2=X_1-\frac{f(X_1)}{f'(X_1)}$$

一般的にX_1なる初期値より

$$X_{i+1}=X_i-\frac{f(X_i)}{f'(X_i)}$$

を繰り返せば近似値の精度向上

初期値2からスタート(条件として解に近く解より大きい初期値を与える)

$=\frac{1}{2}\left(2+\frac{2}{2}\right)$

$=1.5$ ◀・・・1回目

$=\frac{1}{2}\left(1.5+\frac{2}{1.5}\right)$

$=1.41\dot{6}$ ◀・・・2回目

$=\frac{1}{2}\left(1.41\dot{6}+\frac{2}{1.41\dot{6}}\right)$

$=1.414215686$ ◀・・3回目

$=\frac{1}{2}\left(1.414215686+\frac{2}{1.414215686}\right)$

$=1.414213562$ ◀・・4回目

電卓の √‾ も同じ方法で計算を繰り返している

テーラー展開とかオイラーの公式・・・・・

近似はあくまで近似である正解では無い

しかし回数(桁数)を増やすと誤差は小さくなる

基本的な関数計算はOSに用意されているので気にする事は無い
CPUの内部処理として加算しかしてないと言うこと

㉑ 行列式とマトリクス

$a_{11}\ a_{12}\ a_{21}\ a_{22}$ の4要素で$(a_{11}\ a_{22}-a_{12}\ a_{21})$の関係がある場合

$$a_{11}\ a_{22}-a_{12}\ a_{21}=\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

列↓ 行→

・・・(1)を行列式といい右辺の(右下りの要素の積)−(右上りの要素の積)

連立方程式を考える場合

$$\left.\begin{aligned} a_{11}x_1+a_{12}x_2+a_{13}x_3&=y_1\\ a_{21}x_1+a_{22}x_2+a_{23}x_3&=y_2\\ a_{31}x_1+a_{32}x_2+a_{33}x_3&=y_3 \end{aligned}\right\}\ \cdots(2)$$

(2)を配列し直して

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

・・・(3)をマトリクスという

$[A]\quad[X]=[Y]$ ・・・(4)要約表示できる

プログラムではDIMENTIONとして利用された(ガウス・ジョルダンの掃き出し法等)

㉒ 数字と文字

コンピュータでは数字の 1 と文字の“1”は違う
ワープロで「1個のリンゴ」と打ち込めば文字列としての“1”であり数字では無い
表計算ソフトでリンゴの計算セルに 1 と入力するのは数字の 1 である

数値定数としてはプログラム言語によりもよるが、整数形式、固定小数形式、浮動小数形式、16進形式等がある
変数としては数値を入れる数値変数と文字列を入れる文字変数がある

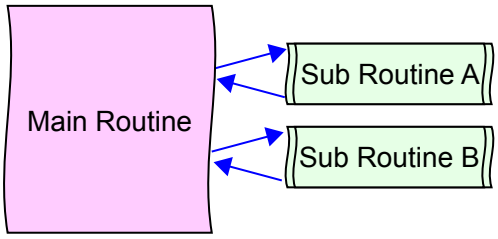
Basic言語の例として%（整数）、!（単精度）、#（倍精度）、$（文字変数）といちいち付けるのは面倒なのでプログラムの始めにDEF（define:定義する）コマンドによる型宣言をする

”REAL” ”INTEGER” ”COMPLEX”

㉓ 乱数

全くでたらめに並んだ数のことを乱数という。シュミレーションやゲームでは欠かせない
コンピュータで作り出すのは困難なのでRNDなどのコマンドは疑似乱数という方法で行っている
乱数は複数の系列を持つが同系列では同じ数になるのでBasicではRANDMIZEとTIMEを組合わせたりした
勝手なもので、同じ結果を期待する場合は誤差が生じランダムな結果を望む時は寸分違わない

㉔ 構造化プログラミング(Structured Programming)



膨大なプログラムを小さな処理(module)に分け階層的な構造にする1つの入口と1つの出口を持つプログラムは順次・選択・反復で構成される構造化定理を踏襲した結果GO TO文は排除された
行番号も廃止された
字下げによる階層化はHTMLのText Editerでも普通である

プログラムはトップダウン設計が基本ではあるが既存のモジュールを流用するとボトムアップ設計になる
臨機応変な組み合わせが一般的なようだ

㉕ オブジェクト指向(Object-Oriented)

「範囲を指定してコピーする」など関連するデータの集合体と、それを操作する手続きを「オブジェクト」(object)と呼ばれる1つの単位とし、オブジェクトの組み合わせによりプログラムを記述する

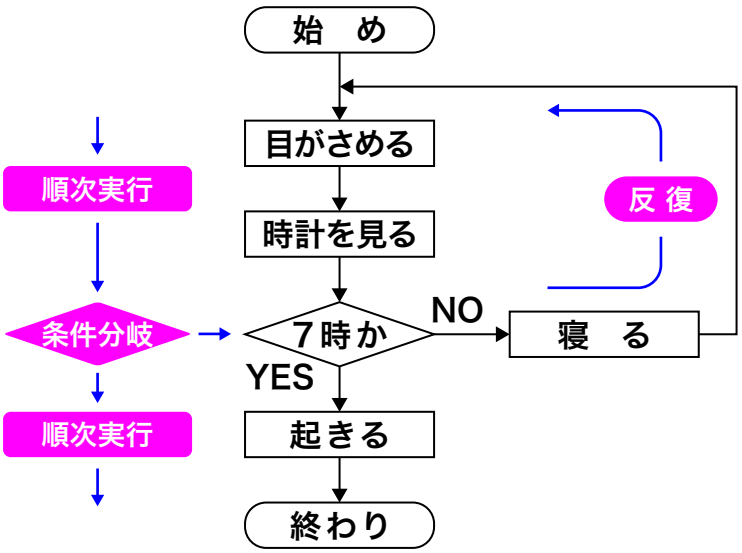
変数と関数の関係をクラスとして分けるクラスベースのオブジェクト指向が主流のようだ

C言語やC++はオブジェクト指向で無いプログラムも書ける

1995年 Sun Microsystems社のJavaから以降C#,Obujective C とオブジェクト指向の言語が登場している

㉖ プログラムの3要素

主流のノイマン型（逐次処理方式）のプログラムは
順次実行：順番に処理する
条件分岐：条件で次の処理を切り替える
反復：条件が満たされるまで作業を繰り返す
の3要素の組み合わせである



㉗ AI（Artifical Inelligence：人工知能）の考察

家電品的なAIはセンサーとマイコンで制御のようなイメージだろうか？
他にもファジィ(Fuzzy)コントロールも流行したが、販売用語は別として・・・

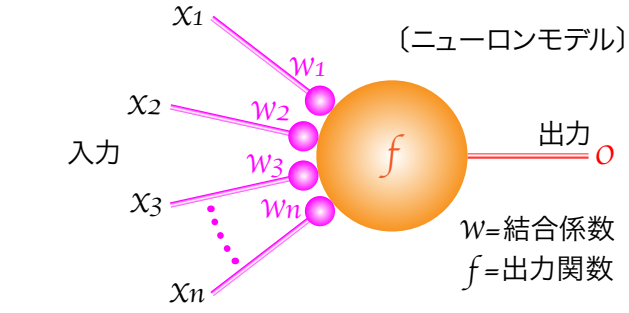
人間の脳神経細胞ニューロンを模したニューラルネットワークという考え方でディープラーニング（深層学習）というプログラミングの研究が進んでいる
適切なビッグデータを入力しラーニングさせた結果の「正誤」を教育する場合と「放任」して教育しない場合を試行錯誤されているらしい

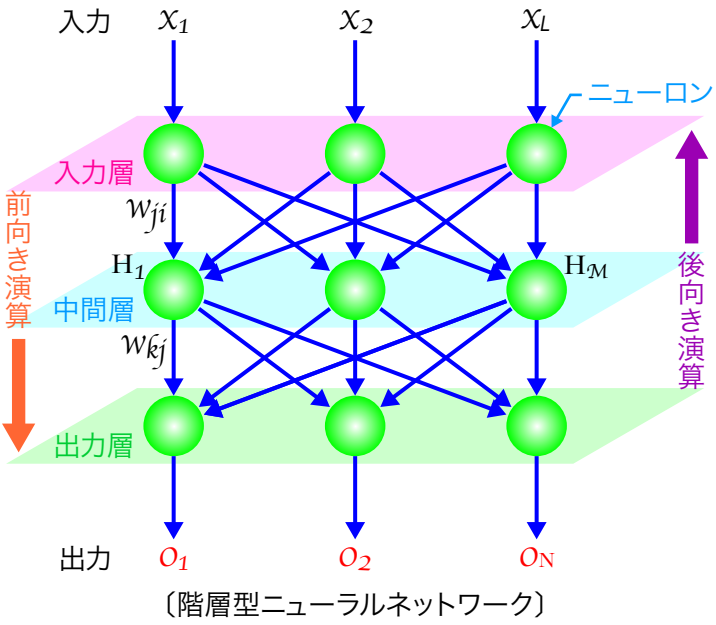
右図の中間層にあるwは重み（冗長度）で、その重みの計算をひたすら実行している

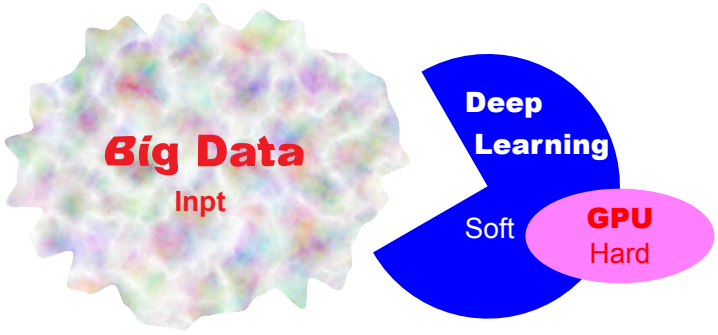
人工知能と言うけれど、思考すると言うよりはまだまだ計算している段階だそうだ

Gogle社の囲碁用コンピュータ“アルファ碁”はNvidia社のGPU 176機が一手毎にフル演算だったそうだ

ディープラーニングによりビッグデータを処理するには高速な演算能力が必要で主役はCPUでは無くGPUの時代なのだそうだ







Ⓩ 偉そ～に 能書を書きました（間違っていたらゴメンナサイ!! 当局は一切の責任を負いません）

自分で作成しても後日開くと理解に苦しむFile？ Program、DTP、CAD等 Digital Data は“3日経てば他人の物”
後の自分ためにも構造化やオブジェクト指向の考え方は理にかなっている

古い考えだが、画面に向かう前の準備や企画も重要だと思う“File名の付方”やDatabase等での管理“Backup”も大切だ“Bugの無いプログラムは存在しない”宇宙的に考えても法則や規則が適応できる範囲がある
Syntax errorに陥るとミスタイプなのか根本的に考え方が間違っているのか？　Try&Error の繰り返しになる
他人に聞くのは簡単だが、絶えず疑問を持って考え、調べ、参考になる手本を解析する

コンピュータの進化もさることながらメカトロニクス、メカニズムも重要である制御信号に追いつけないErrorはオートパイロットにとっては致命的である

頭の固くなった今、プログラムすることも半田コテを握ることも無く、望みもしないシステムのUpdateと充てがわれたソフトにサブスクリプションとかで延々と金を蝕まれ翻弄される毎日である

IDとPassword、Networkに監理され便利ツールであるべきコンピュータに振り回されるのは勘弁いただきたいリモートワークの時代、AIとアバターで仕事させるか・・・で人間は何をするか？だが確実に年老いていく究極、アンドロイドではなく無機質な人間型ロボットで身の回りの世話（バトラーサービス）をお願いしたいか