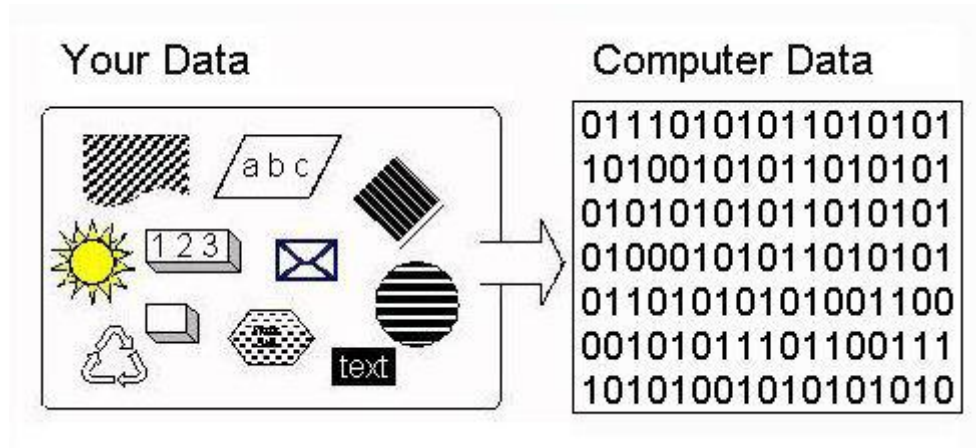


01-01. 機械語命令と2進数の関係

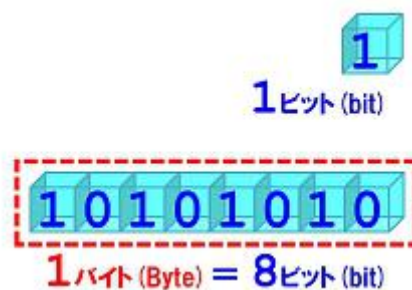
◆ 機械語命令とは

あらゆる情報を『0』と『1』の2進数を機械語として、CPUに対して、命令が実行される。



◆ 様々な進数とbitの関係

しかし、人間が扱う上では8進数あるいは16進数に変換して表現することが適している。2進数1ケタが『1 bit』と定義されている。8進数の1ケタは2進数の3ケタ (= 3 bit) に相当し、16進数の1ケタは2進数の4ケタ (4 bit) に相当する。



10進数	16進数	8進数	2進数	bit 数				
0	0	0	0	1 bit	2 bit	3 bit	4 bit	5 bit
1	1	1	1					
2	2	2	10					
3	3	3	11					
4	4	4	100					
5	5	5	101					
6	6	6	110					
7	7	7	111					
8	8	10	1000					
9	9	11	1001					
10	A	12	1010					
11	B	13	1011					
12	C	14	1100					
13	D	15	1101					
14	E	16	1110					
15	F	17	1111					
16	10	20	10000					
17	11	21	10001					
18	12	22	10010					
19	13	23	10011					
20	14	24	10100					
21	15	25	10101					
22	16	26	10110					
23	17	27	10111					
24	18	30	11000					
25	19	31	11001					
26	1A	32	11010					
27	1B	33	11011					
28	1C	34	11100					
29	1D	35	11101					
30	1E	36	11110					
31	1F	37	11111					

◆ Byte単位

1000 Byte = 1k Byte



記憶容量など大きい数値をあらわす補助単位

補助単位	意味	説明
キロ (k)	10^3	基本単位×1,000倍の意味
メガ (M)	10^6	基本単位×1,000,000倍の意味
ギガ (G)	10^9	基本単位×1,000,000,000倍の意味
テラ (T)	10^{12}	基本単位×1,000,000,000,000倍の意味



処理速度など小さい数値をあらわす補助単位

補助単位	意味	説明
ミリ (m)	10^{-3}	基本単位×1/1,000倍の意味
マイクロ (μ)	10^{-6}	基本単位×1/1,000,000倍の意味
ナノ (n)	10^{-9}	基本単位×1/1,000,000,000倍の意味
ピコ (p)	10^{-12}	基本単位×1/1,000,000,000,000倍の意味

◆ 一般的なCPUが扱える情報の種数

CPUでは、各データは2進法によって区別されている。CPUは4、8、16、32-bitバージョンと進歩し、2008年の後半からは64-bitバージョンのCPUが普及し始めた。1-bitは2種類の情報を表すことができるため、32-bitのCPUでは 2^{32} 、64-bitでは 2^{64} の種類の情報を扱う事ができる。

01-02. 機械語命令の種類

◆ 設定命令

- 実行アドレスをレジスタに設定する場合

例 実行アドレス (例 1000h) をレジスタ1 に設定する。

レジスタ番号	内容
1	

- 実行アドレスが指す語の内容をレジスタに設定する場合

例 実効アドレス（例 1000h）が指す語の内容をレジスタ 1 に設定する。



- レジスタの内容を実行アドレスに格納する場合

例 レジスタ 1 の内容を，実効アドレス（例 1000h）に格納する。



◆ シフト命令（※別節を参照せよ）

◆ 計算命令

レジスタから取り出した値を別の値と足し、その結果を元のレジスタに設定すること。

◆ 論理演算命令（※3章を参照せよ）

01-03. シフト命令

◆ 論理左シフト

2 進数の場合...

左に 1 bitシフトすると『2倍』

左に 1 bitシフトし、元の値を足すと『3倍』

左に 2 bitシフトすると『4倍』

左に 2 bitシフトし、元の値を足すと『5倍』

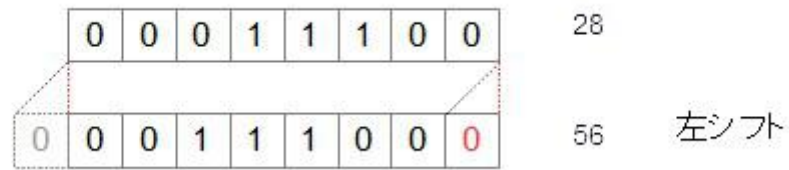
左に 2 bitシフトし、元の値を足して『5倍』。さらに 2 bitシフトすると『10倍』

左に 3 bitシフトすると『8倍』

- 正の数の場合

【具体例】

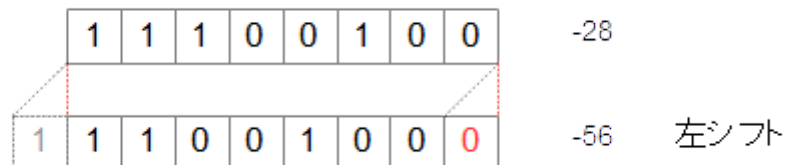
00011100



• 負の数の場合

【具体例】

11100100



◆ 論理右シフト

2進数の場合...

右に1 bitシフトすると『1/2』

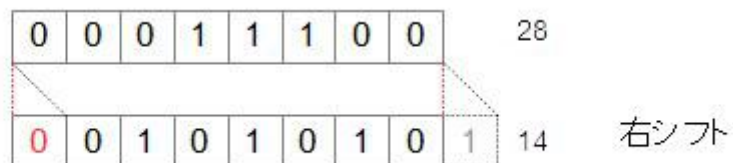
右に2 bitシフトすると『1/4』

右に3 bitシフトすると『1/8』

• 正の数の場合

【具体例】

00011100

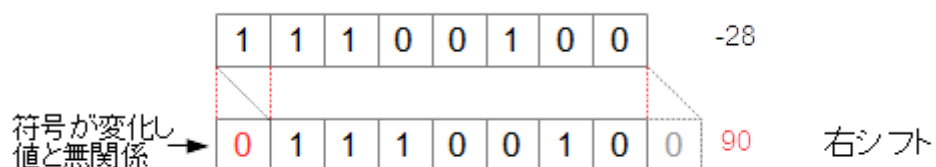


• 負の数の場合（計算はできない）

【具体例】

11100100

負の数で論理右シフトを行う場合、間違った計算が行われてしまう。こういう場合、算術シフトが用いられる。



◆ 算術左シフト

2進数の場合...

最上位には、正負を表す『符号bit』を置く。

左に1 bitシフトすると『2倍』

左に1 bitシフトし、元の値を足すと『3倍』

左に2 bitシフトすると『4倍』

左に2 bitシフトし、元の値を足すと『5倍』

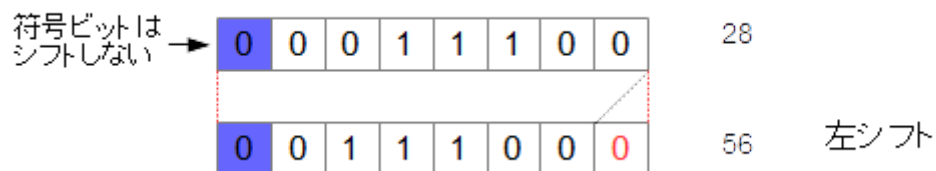
左に2 bitシフトし、元の値を足して『5倍』。さらに2 bitシフトすると『10倍』

左に3 bitシフトすると『8倍』

- 正の数の場合

【具体例】

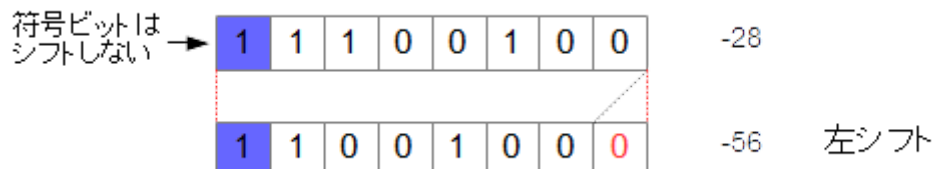
00011100



- 負の数の場合

【具体例】

00011100



◆ 算術右シフト

2進数の場合...

最上位には、正負を表す『符号bit』を置く。

右に1 bitシフトすると『1/2』

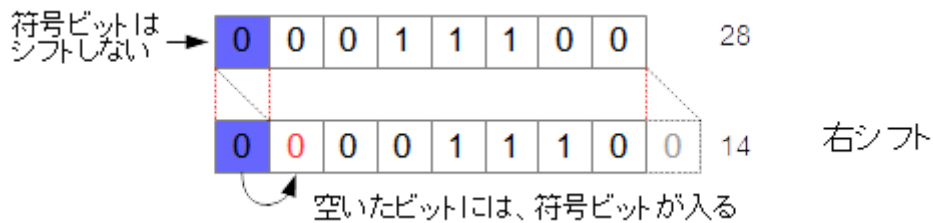
右に2 bitシフトすると『1/4』

右に3 bitシフトすると『1/8』

- 正の数の場合

【具体例】

00011100



• 負の数の場合



01-04. 機械語命令の実行手順

◆ 実行手順



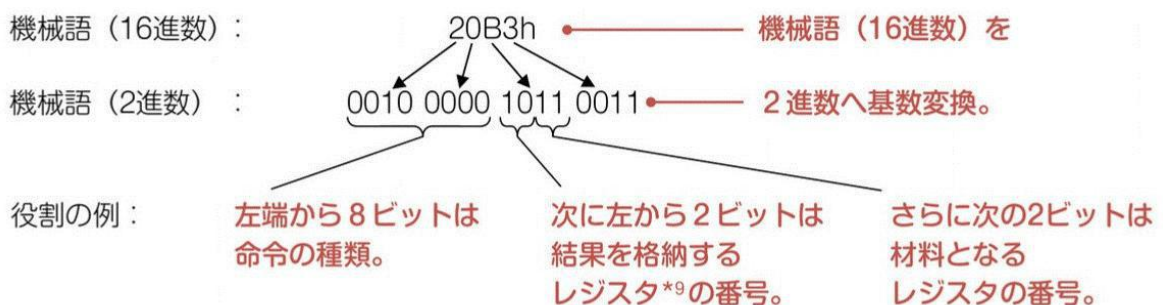
1. 16進数が2進数に変換され、記号へ値が割り当てられる。(ビット分割)
2. 記号の値を基に、実行アドレスの計算方法が選択され、実行される。(実行アドレスの計算)
3. 実行アドレスを基に、機械語命令が実行され、値がレジスタやメモリに書き留められる。(機械語命令のトレース)

◆ (1) ビット分割

【具体例】

命令 : 20B3h

• 16進数の2進数への変換



• 記号への値の割り当て



◆（２）実効アドレスの計算

• 実行アドレスの計算方法の選択

『X=2』、『I=1』より、表の網掛けの計算式を選択。

X	I	実効アドレス
0	0	adr
1～3	0	adr + [X]
0	1	[adr]
1～3	1	[adr + [X]]

表 実効アドレスの算出方法

• 実効アドレスの計算の実行

ここに、レジスタ番号と内容の表を張る。

(実効アドレス) = [adr + [X]]

= [1000h + [レジスタ2]] (※配列のように、レジスタ2の値を参照)

= [1000h + 0002h]

= [1002h]

= 1003h

◆（３）機械語命令のトレース

01-05. 構文解析における数式の認識方法

◆ 逆ポーランド表記法（後置表記法）

演算子（ $+$, $-$, \times , \div など）を被演算子（数値や変数, また計算の結果）の後ろに書くことで数式を表現する方法。ちなみに、人間が使っている表記方法は、『中置記法』という。

【具体例】

$$Y = (A + B) \times (C - (D \div E))$$

1. 括弧は先に計算するので塊と見なす。

$$(A + B) \Rightarrow AB +$$

2. 括弧は先に計算するので塊と見なす。

$$(D \div E) \Rightarrow DE \div$$

3. 括弧は先に計算するので塊と見なす。

$$(AB +) \times (C - DE \div) \Rightarrow (AB +)(CDE \div -) \times$$

4. 括弧を外しても、塊はそのまま。

$$(AB +)(CDE \div -) \times \Rightarrow AB + CDE \div - \times$$

5. 左辺と右辺をそれぞれ塊と見なす。

$$Y = AB + CDE \div - \times \Rightarrow YAB + CDE \div - \times =$$

01-06. CPUにおける小数の処理方法

◆ 固定小数点数

『この位置に小数点がある』な前提で数字を扱うことによって、小数点を含む数値を表現する方法。

「12345」

覚えろ



はいよ

123.45



ここに点を打て



はいよ

『誤差』：実際の数値とCPUが表現できる数値の間に生じるズレのこと。

◆ 無限小数

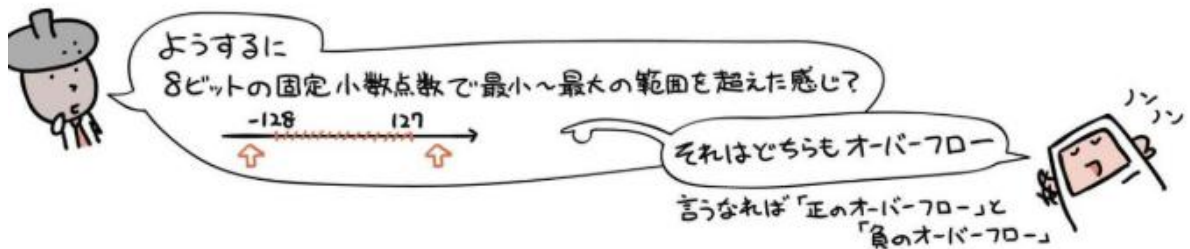
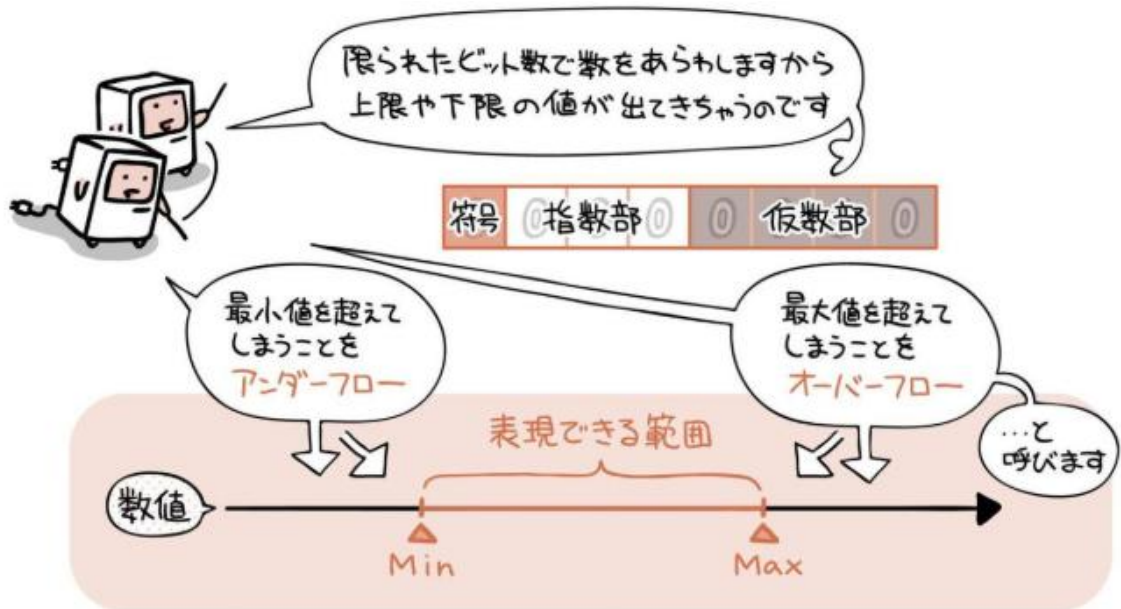


◆ 桁溢れ誤差

【具体例】

初代ドラクエ

初代のドラゴンクエストの経験値の上限は「65535」だった。これは、経験値が16bit（2byte）で表されており、桁溢れが起きることを防ぐために65535以上は計算しないようになっていた。



◆ 情報落ち

それはこーいうことなのです

① 仮数部を4けたであらわす浮動小数点数があったとします

符号 指数部 仮数部

② それで、次の足し算をします

説明のため
10進数を例にします

$$0.1234 \times 10^0 + 0.4321 \times 10^{-4}$$

③ 計算するには指数を揃えなきゃいけません

$$\begin{array}{r} 0.1234 \times 10^0 \\ + 0.000000004321 \times 10^0 \\ \hline 0.123400004321 \times 10^0 \end{array}$$

するとこんな
答えになる

④ 浮動小数点数なので、正規化をします

0.1234 × 10⁰

仮数部は
4けた

小さい数の方が
有効けた数から
はみ出しちゃうので

なかったことに
されてしまっ
ているのでは

ホントだ!
反映され
ない!

◆ 打ち切り誤差

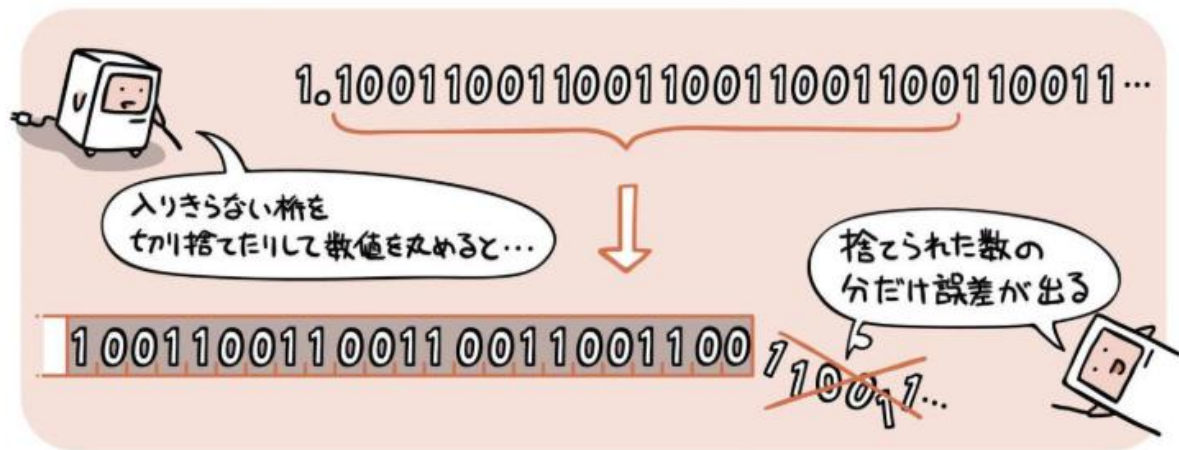
円周率は、途中で計算を打ち切る。



◆ 桁落ち



◆ 丸め誤差



02-01. N 進数 → 10進数（重み掛けを行う）

◆ 16進数 → 10進数

- 整数

【具体例】『CA125』

1. 『 $(16^0 \times 5) + (16^1 \times 2) + (16^2 \times 1) + (16^3 \times A) + (16^4 \times C)$ 』というように、下の位から、順に 16^N をかけていく。（AとCは、10進数に変換して『10』と『12』）
2. $(1 \times 5) + (16 \times 2) + (256 \times 1) + (4096 \times 10) + (65536 \times 12) = 827685$

- 少数

◆ 2進数 → 10進数

- 整数

『1101101』

1. 『 $(2^0 \times 1) + (2^1 \times 0) + (2^2 \times 1) + (2^3 \times 1) + (2^4 \times 0) + (2^5 \times 1) + (2^6 \times 1)$ 』というように、下の位から、順に 2^N をかけていく。

- 少数

02-02. 10進数 → N 進数（Nで割り続ける）

◆ 10進数 → 16進数

- 整数

【具体例】『27』

1. 27を16で割り続ける。
2. 10～15は、A～Fで表記されるため、11をBで表記。
3. 余りを並べ、答えは『1B』

16	27	余り	
16	1	...	11(=B)
	0	...	1

(1B)₁₆

下から順に並べる

- 少数

【具体例】『0.1015625』

- 『0.1015625』に16をかけ、整数部分を取り出す。(0.1015625 × 16 = 1.625。『1』を取り出し、16進数に変換して『1』)
- 計算結果の少数部分に16をさらにかける。少数部分が0になるまで、これを繰り返す。(0.625 × 16 = 10.0より、『10』を取り出し、16進数に変換して『A』)
- 少数部分が0になったので、取り出した数を順に並べ、答えは『0.1A』

◆ 10進数 → 2進数

- 整数

【具体例】『109』

10進数	余り
2) 109	... 1
2) 54	... 0
2) 27	... 1
2) 13	... 1
2) 6	... 0
2) 3	... 1
1	

下から順に並べると 1101101 となる。

- 少数

02-03. X 進数 → 10進数 → Y 進数

一度、10進数に変換してから、任意の進数に変換する。

◆ 16進数 → 2進数

- 整数

【具体例】『20B3』

1. 2、0、B、3を10進数に変換して、『 $(16^0 \times 3) + (16^1 \times 11) + (16^2 \times 0) + (16^3 \times 2) = 8371$ 』
2. 10と15を2進数に変換して、『0010』、『0000』、『1011』、『0011』
3. よって、AFは10進法に変換して『0010000010110011』

02-04. X 進数 → 10 進数 → Y 進数 → 10 進数

◆ 16進数 → 2進数

- 少数

【具体例】

2A.4C

1. 整数部分の2Aを10進数に変換して、
2. 42を2進数に変換して、『101010』。また、余り計算の時、余り1を 2^N に直しておく。
3. 整数の場合、下位の桁から、『 $(2^0 \times 0) + (2^1 \times 1) + (2^2 \times 0) + (2^3 \times 1) + (2^4 \times 0) + (2^5 \times 1) + (2^6 \times 0) + (2^7 \times 0) + (2^8 \times 0)$ 』
= 『 $2^5 + 2^3 + 2^1$ 』
(※16進数からの変換の場合、101010は、00101010として扱うことに注意)
4. 76を2進数に変換して、『1001100』。また、余り計算の時、余り1を 2^N に直しておく。
5. 少数部分の場合、上位の桁から、『 $(2^{-1} \times 0) + (2^{-2} \times 1) + (2^{-3} \times 0) + (2^{-4} \times 0) + (2^{-5} \times 1) + (2^{-6} \times 1) + (2^{-7} \times 0) + (2^{-8} \times 0)$ 』
= 『 $2^{-2} + 2^{-5} + 2^{-6}$ 』
(※16進数からの変換の場合、1001100は、01001100として扱うことに注意)
6. したがって、『 $2^5 + 2^3 + 2^1 + 2^{-2} + 2^{-5} + 2^{-6}$ 』

03-01. 論理回路

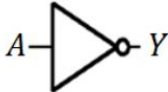
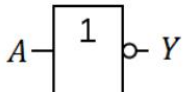

◆ 論理式

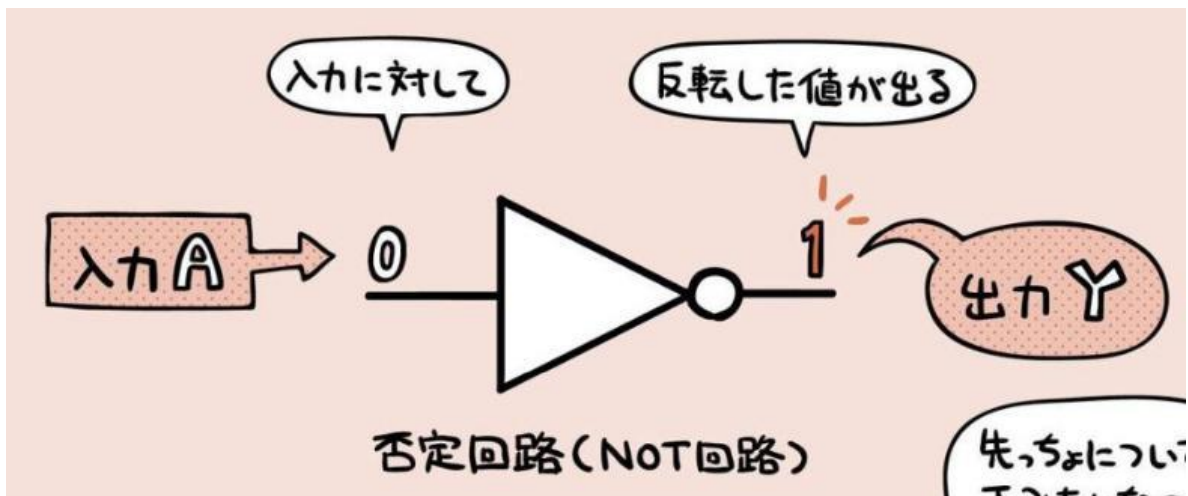
演算	意味	式
否定	1つの命題が「～ではない」と逆になる (NOT)	\bar{A}
論理積	2つの命題が「～かつ～」の関係 (AND)	$A \cdot B$
論理和	2つの命題が「～または～」の関係 (OR)	$A + B$

以下のベン図では、集合Aと集合Bは入力が『1』の場合、外側は入力が『0』の場合を表している。
演算方法を思い出すときには、ベン図を思い出せ。

◆ 否定回路 (NOT回路)、NOT演算、ベン図


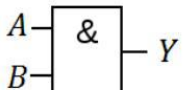
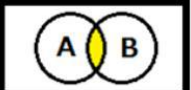
丸い記号が否定を表す。

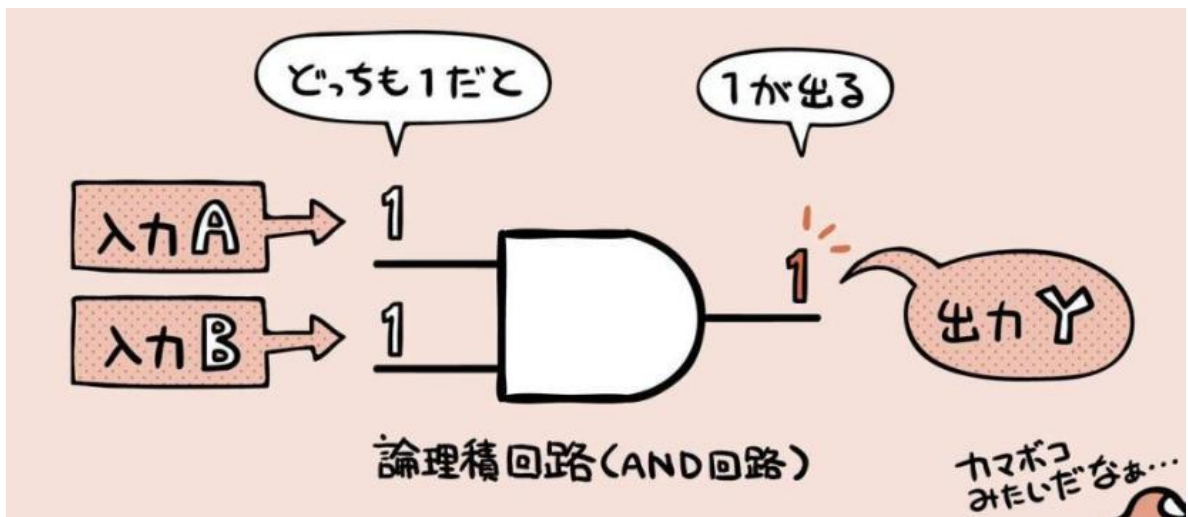
NOT	$Y = \bar{A}$			<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	
A	Y										
0	1										
1	0										



◆ 論理積回路 (AND回路)、AND演算、ベン図




2つのbitを比較して、どちらも『1』なら『1』を出力。

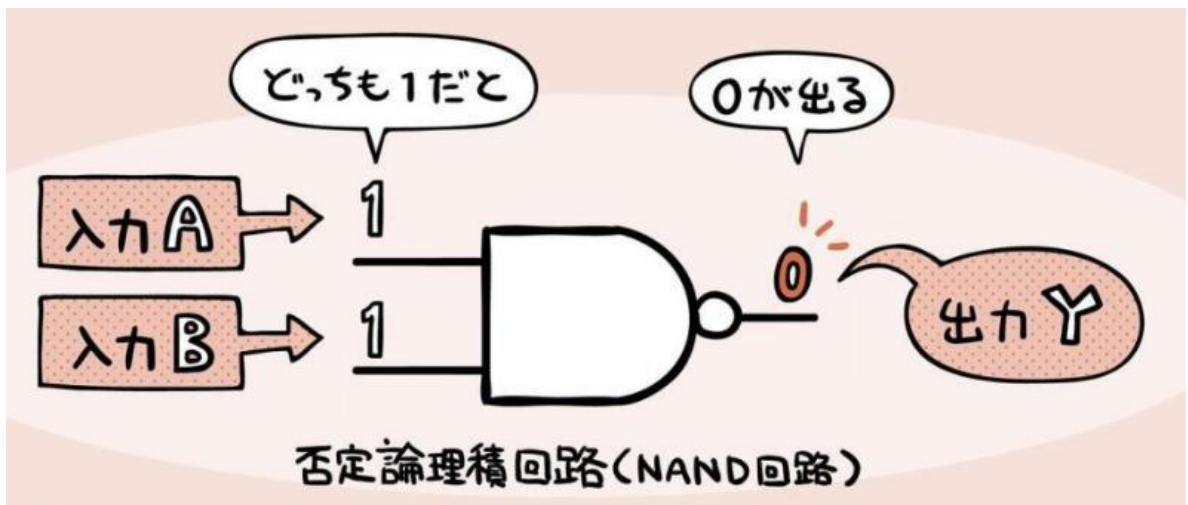
AND	$Y = A \cdot B$			<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	
A	B	Y																		
0	0	0																		
0	1	0																		
1	0	0																		
1	1	1																		



◆ 否定論理積回路 (NAND回路)、NAND演算、ベン図




2つのbitを比較して、どちらも『1』なら『0』を出力。ベン図では両方が『1』以外の場合を指しているが、回路の出力をうまく説明できない...

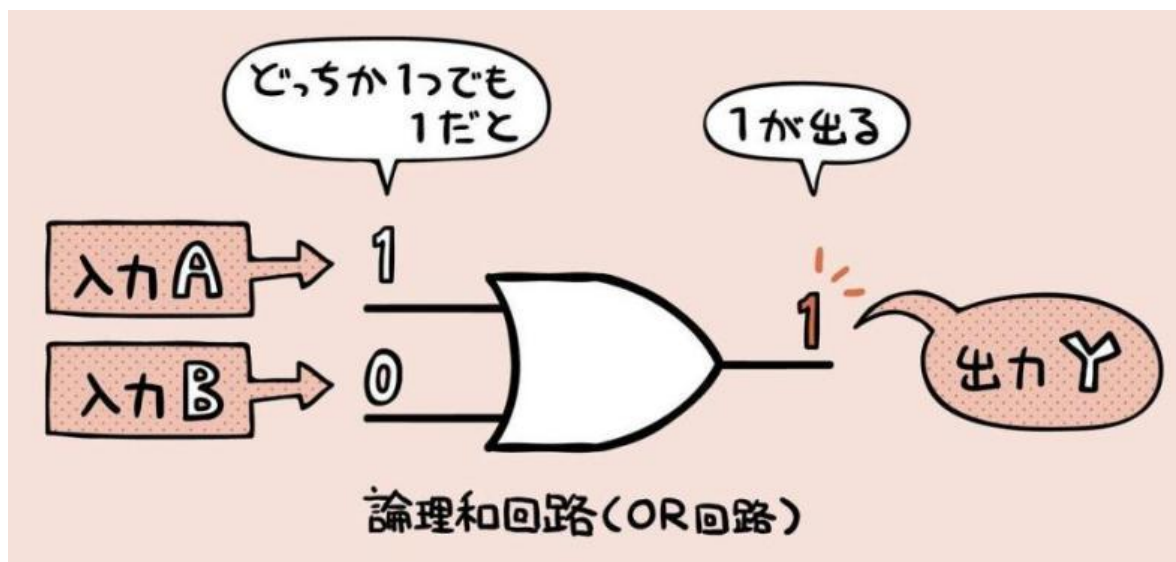
NAND	$Y = \overline{A \cdot B}$			<table border="1"><thead><tr><th>A</th><th>B</th><th>Y</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	
A	B	Y																		
0	0	1																		
0	1	1																		
1	0	1																		
1	1	0																		



◆ 論理和回路 (OR回路)、OR演算、ベン図




2つのbitを比較して、どちらかが『1』なら『1』を出力。

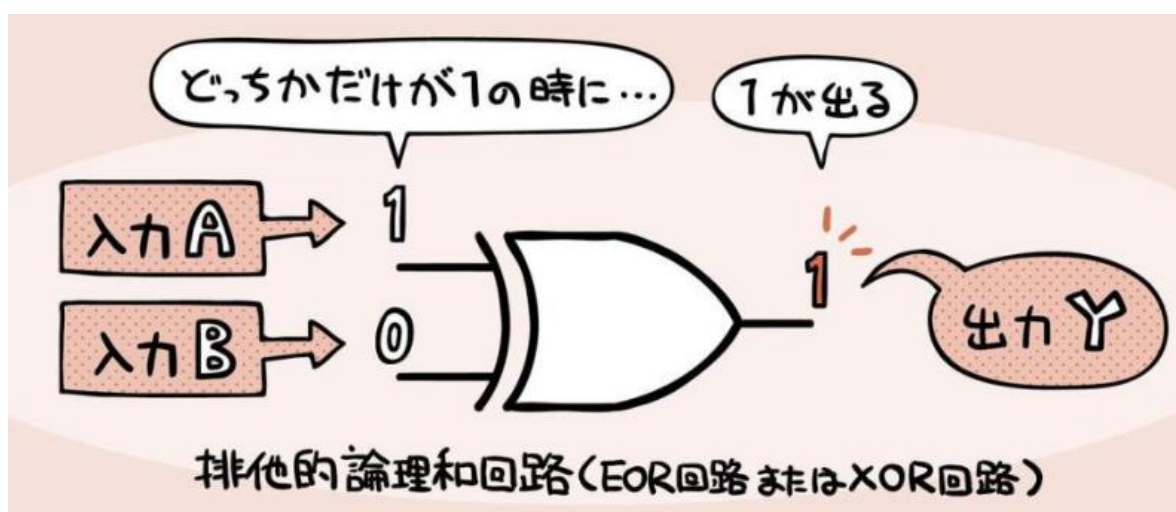
OR	$Y = A + B$			<table border="1" data-bbox="876 1731 981 1740"><tr><td>A</td><td>B</td><td>Y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	
A	B	Y																		
0	0	0																		
0	1	1																		
1	0	1																		
1	1	1																		



◆ 排他的論理和回路 (EOR回路/XOR回路)、EOR演算、ベン図




2つのbitを比較して、どちらかだけが『1』なら『1』を出力。

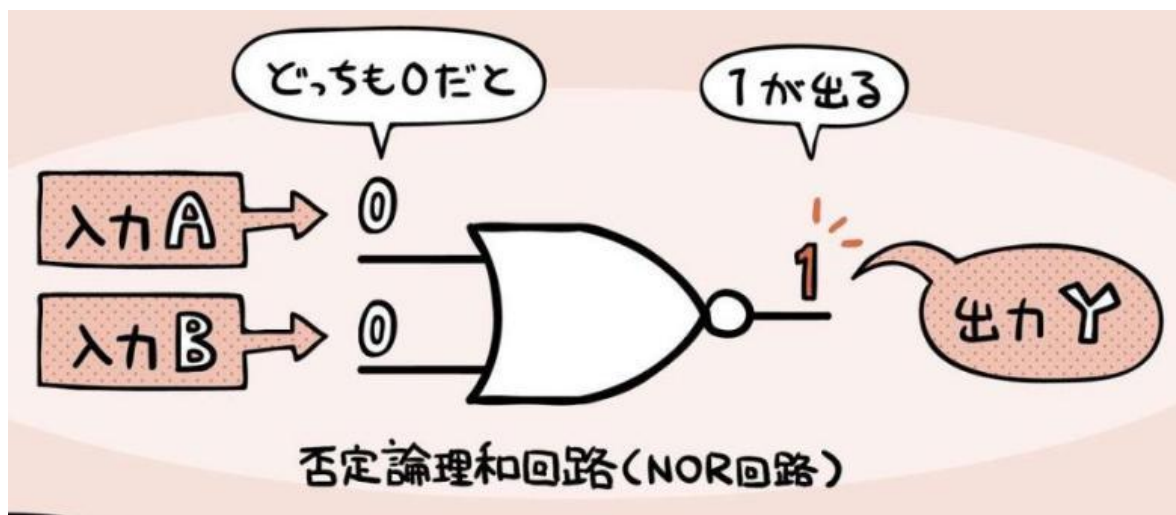
EX-OR (X-OR)	$Y = A \oplus B$ $= \bar{A} \cdot B + A \cdot \bar{B}$			<table border="1" data-bbox="916 864 1032 878"><tr><td>A</td><td>B</td><td>Y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	
A	B	Y																		
0	0	0																		
0	1	1																		
1	0	1																		
1	1	0																		



◆ 否定論理和回路 (NOR回路)、NOR演算、ベン図

2つのbitを比較して、どちらも『0』なら『1』を出力。

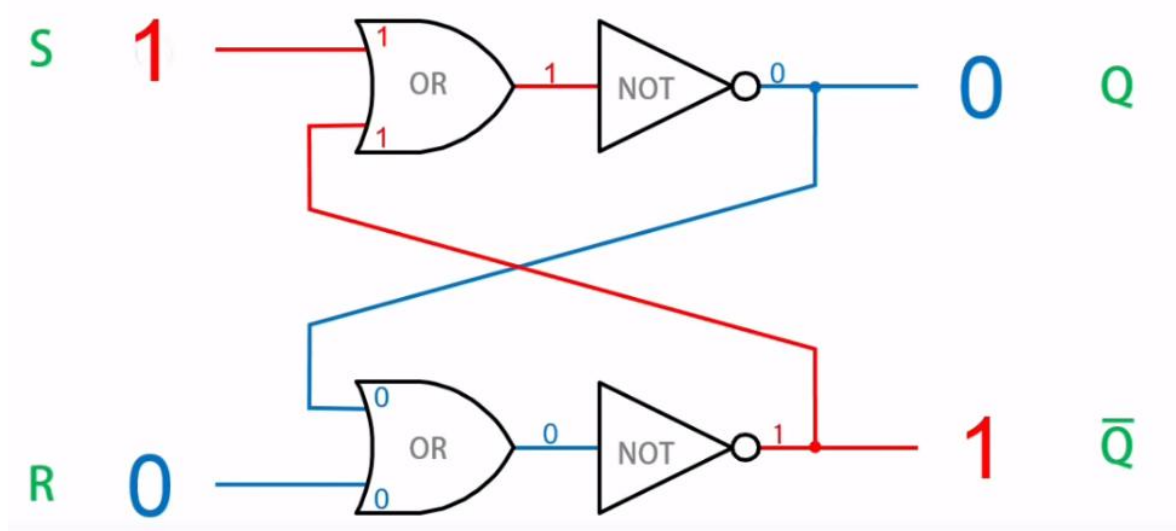
NOR	$Y = \overline{A + B}$			<table border="1"><thead><tr><th>A</th><th>B</th><th>Y</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	
A	B	Y																		
0	0	1																		
0	1	0																		
1	0	0																		
1	1	0																		



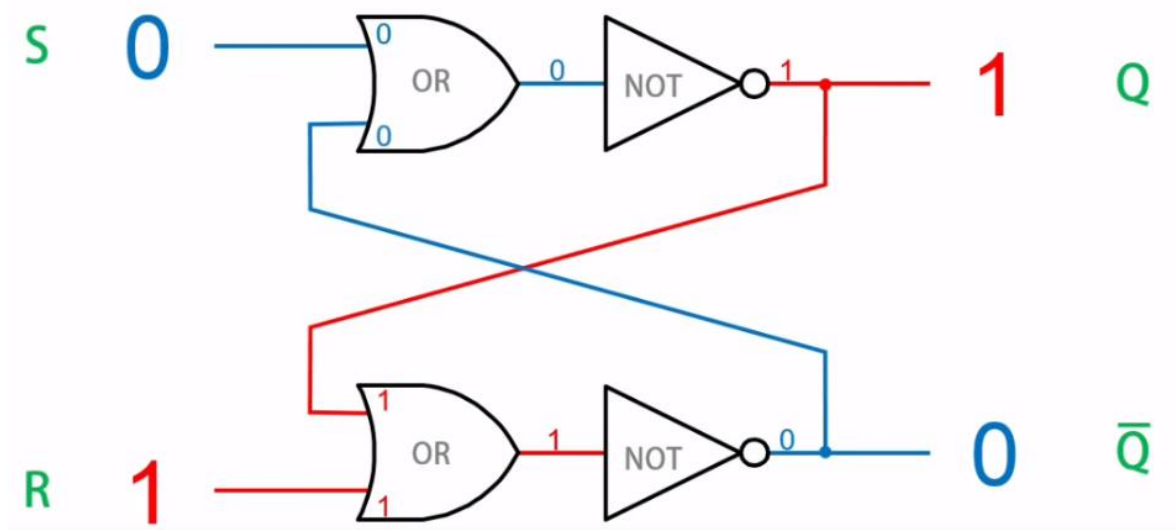
◆ フリップフロップ回路

わかりやすい動画解説：<https://www.youtube.com/watch?v=4vAGaWyGanU>

SRAMの電子回路に用いられている（6章を参照）。Set側に初期値『1』が入力される。入力を『0』に変えても、両方の出力結果は変わらず、安定している。



Reset側に『1』を入力すると、両方の出力結果は変化する。



03-02. 論理演算命令

◆ 論理積

【例題1】

16進数の『F』は、2進数で『0000 0000 0000 1111』で表される。よって、000Fを用いてAND演算した場合、下位4桁を変化させずに取り出すことができる。

```
1100 1101 1111 1000
0000 0000 0000 1111
-----
0000 0000 0000 1000
```

引用 : <https://ameblo.jp/kou05/entry-10883110086.html>

【例題2】

16進数の『7F』は、2進数で『0000 0000 0111 1111』で表される。よって、7Fを用いてAND演算した場合、下位7桁を変化させずに取り出すことができる。

```
1100 1101 1111 1000
0000 0000 0111 1111
-----
0000 0000 0111 1000
```

【例題3】

- 実効アドレスの値 : $(0000\ 0000\ 0000\ 0101)_2 = 0005h$
- レジスタr : $(0000\ 0000\ 0000\ 0011)_2 = 0003h$
- 実行後のレジスタr : $(0000\ 0000\ 0000\ 0001)_2 = 0001h$

◆ 否定論理積

◆ 論理和

- 実効アドレスの値 : $(0000\ 0000\ 0000\ 0101)_2 = 0005h$
- レジスタr : $(0000\ 0000\ 0000\ 0011)_2 = 0003h$
- 実行後のレジスタr : $(0000\ 0000\ 0000\ 0111)_2 = 0007h$

◆ 排他的論理和

- 実効アドレスの値 : $(0000\ 0000\ 0000\ 0101)_2 = 0005h$
- レジスタr : $(0000\ 0000\ 0000\ 0011)_2 = 0003h$
- 実行後のレジスタr : $(0000\ 0000\ 0000\ 0110)_2 = 0006h$

◆ 否定論理和

【例題】

XとYの否定論理積 $X \text{ NAND } Y$ は、 $\text{NOT}(X \text{ AND } Y)$ として定義される。 $X \text{ OR } Y$ を NAND だけを使って表した論理式はどれか。

$\Rightarrow X=0, Y=0$ のときに $X \text{ OR } Y$ が『0』になることから、『0』になる選択肢を探す。

- **$((X \text{ NAND } Y) \text{ NAND } X) \text{ NAND } Y$**
 $((0 \text{ NAND } 0) \text{ NAND } 0) \text{ NAND } 0$
 $= (1 \text{ NAND } 0) \text{ NAND } 0$
 $= 1 \text{ NAND } 0$
 $= 1$
- **$(X \text{ NAND } X) \text{ NAND } (Y \text{ NAND } Y)$**
 $(0 \text{ NAND } 0) \text{ NAND } (0 \text{ NAND } 0)$
 $= 1 \text{ NAND } 1$
 $= 0$
- **$(X \text{ NAND } Y) \text{ NAND } (X \text{ NAND } Y)$**
 $(0 \text{ NAND } 0) \text{ NAND } (0 \text{ NAND } 0)$
 $= 1 \text{ NAND } 1$
 $= 0$
- **$X \text{ NAND } (Y \text{ NAND } (X \text{ NAND } Y))$**
 $0 \text{ NAND } (0 \text{ NAND } (0 \text{ NAND } 0))$
 $= 0 \text{ NAND } (0 \text{ NAND } 1)$
 $= 0 \text{ NAND } 1$
 $= 1$