ビットについて

SecPrj Intro-phase

ビットとは

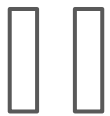
ビットとは、情報量の最小単位で、二つの選択肢から一つを特定する情報の量。語源は “binary digit” （二進法の数字）と言われ、コンピュータなどでは0と1のいずれかを取る**二進数**の一桁として表される。情報をすべてビット列に置き換えて扱うことを「デジタル」（digial）という。

（ビットとは - IT用語辞典 e-Words　より）

『0 / 1で表現される，情報量の最小単位』

10110010

$10110010_{(2)}$



$178_{(10)}$

普段は10進数を使っている

使える数字は0~9の10種類

9の次は繰り上がりで，10になる

ちなみに時間は24進数

分/秒は60進数

1 7 8

= 100 * **1** + 10 * **7** + 1 * **8**

2進数を考える

使える数字は**0,1の2種類**

1の次は繰り上がりで，$10_{(2)}$になる

1 0 1 1 0 0 1 0$_{(2)}$

= 128*1 + 64*0 + 32*1 + 16*1 + 8*0 + 4*0 + 2*1 + 1*0

= 128*1 + 32*1 + 16*1 + 2*1

それぞれの桁には重みがあり，2^(重み)で計算される

1 0 1 1 0 0 1 0 (2)

2^n（重み）	7	6	5	4	3	2	1	0
2進数	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**
値	128	64	32	16	8	4	2	1

n桁目の重みは(n-1)

1 0 1 1 0 0 1 0 (2)

2^n（重み）	7	6	5	4	3	2	1	0
2進数	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**
値	128	64	32	16	8	4	2	1

= 128 + 32 + 16 + 2

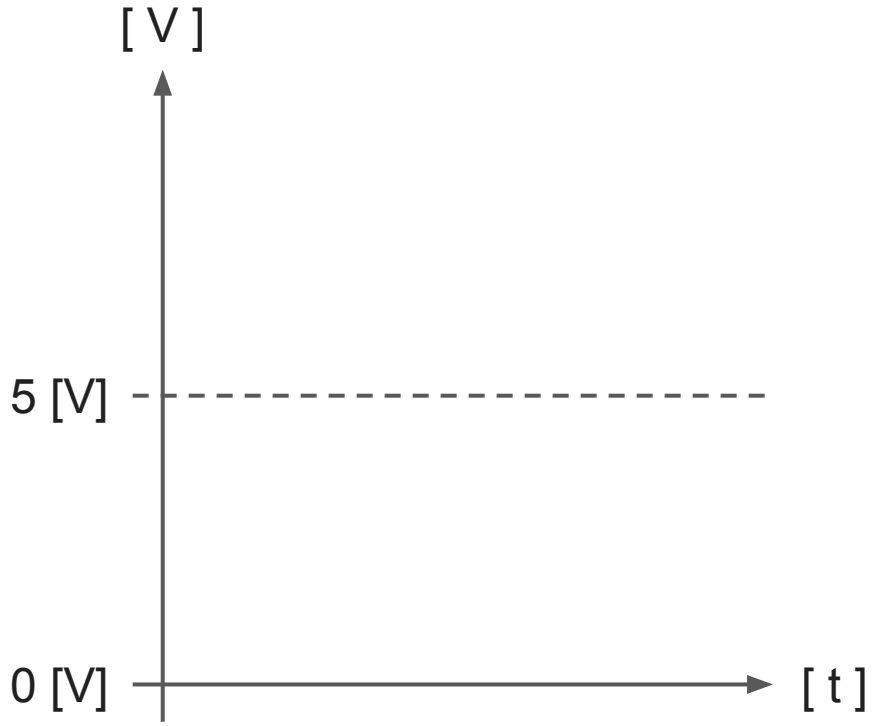
= 178

コンピュータ内でどう扱うか

コンピュータは電気によって動いている

0 / 1を電気の**ある / ない**に置き換える

具体的には，電圧が閾値以上かどうか

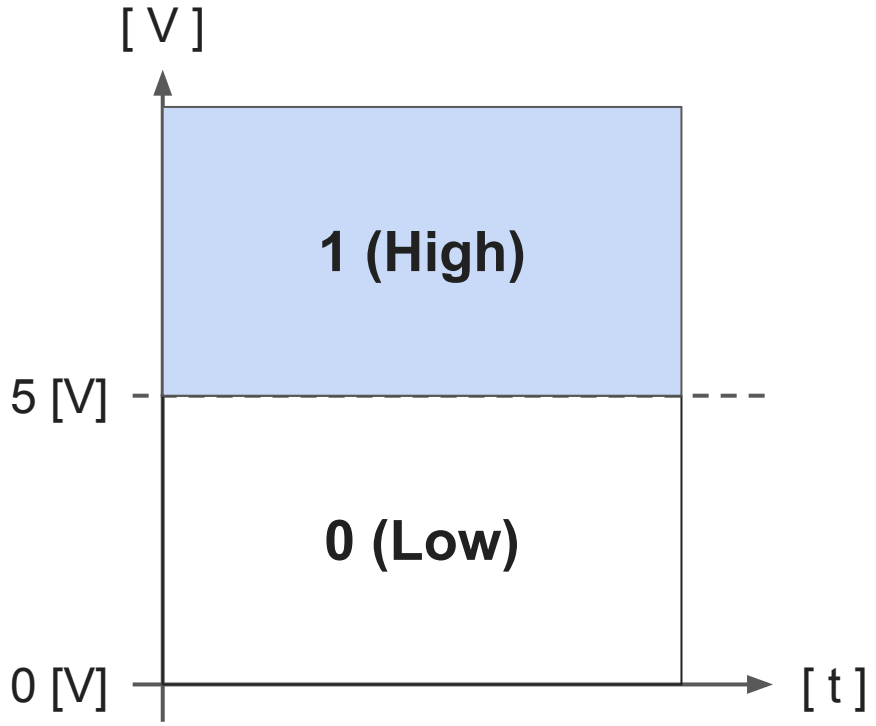


コンピュータ内でどう扱うか

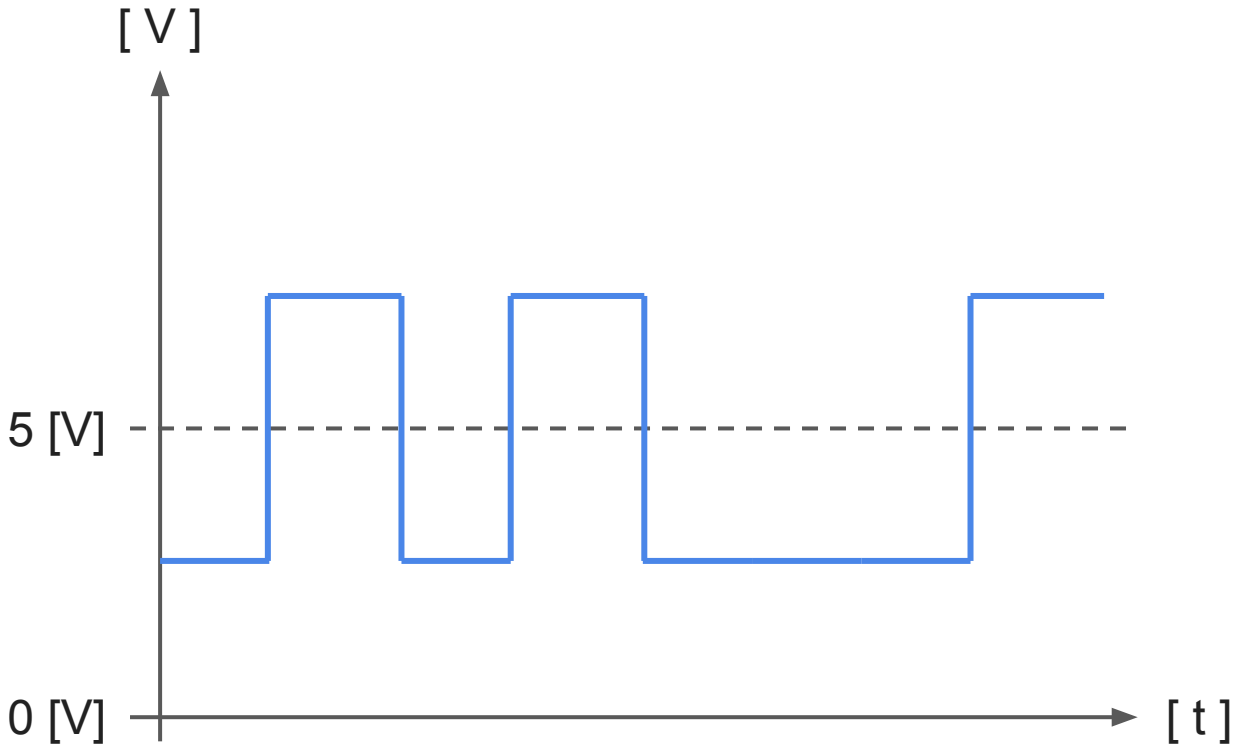
コンピュータは電気によって動いている

0 / 1を電気の**ある / ない**に置き換える

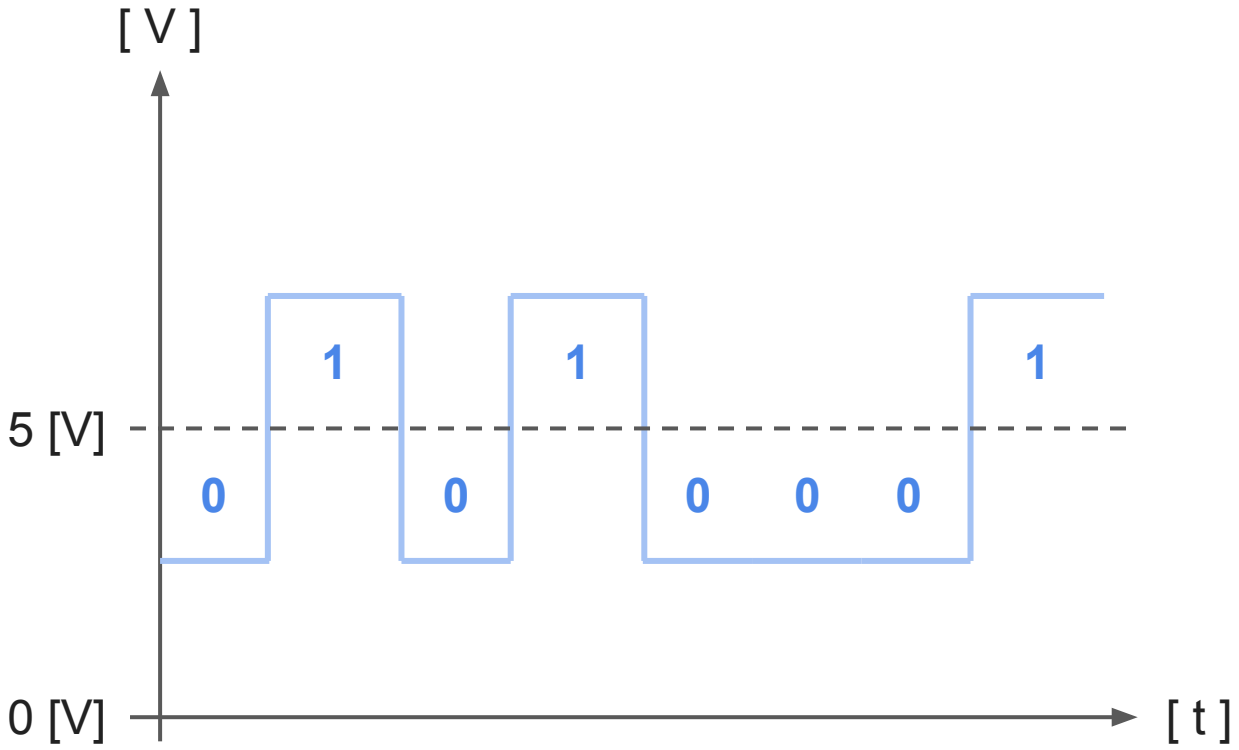
具体的には，電圧が閾値以上かどうか



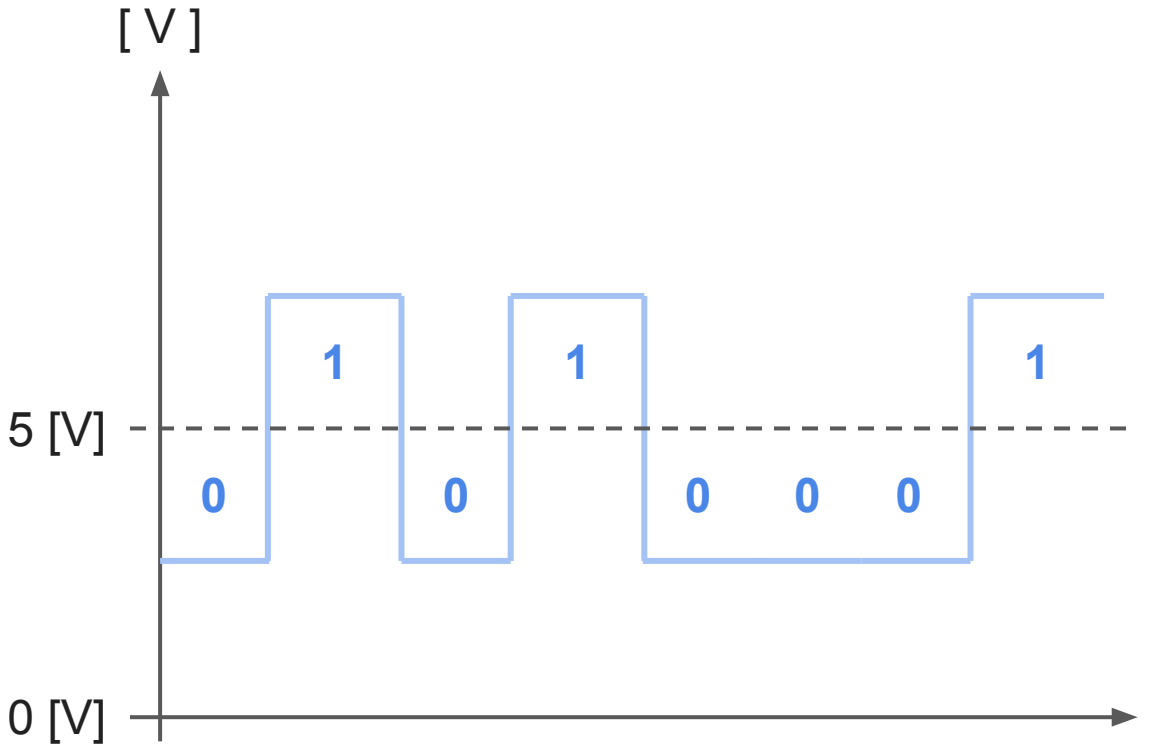
電気信号からビット列への変換（A/D変換）



電気信号からビット列への変換（A/D変換）



電気信号からビット列への変換（A/D変換）



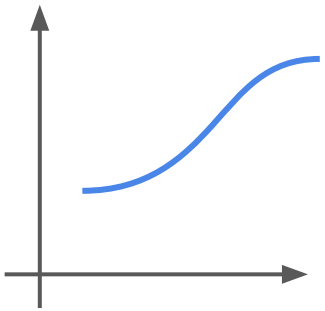
バイトオーダー
（どっちから解釈するか）
は予め決めておく
（Big Endian / Little Endian）

$01010001_{(2)}$

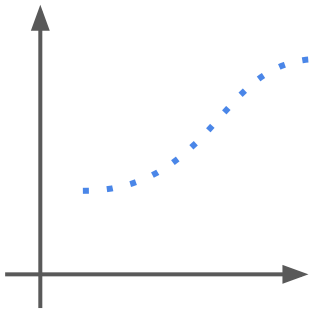
$81_{(2)}$

アナログとデジタル（余談）

アナログは連続的な表現，デジタルは離散的な表現



アナログ（途切れなく続いてる）



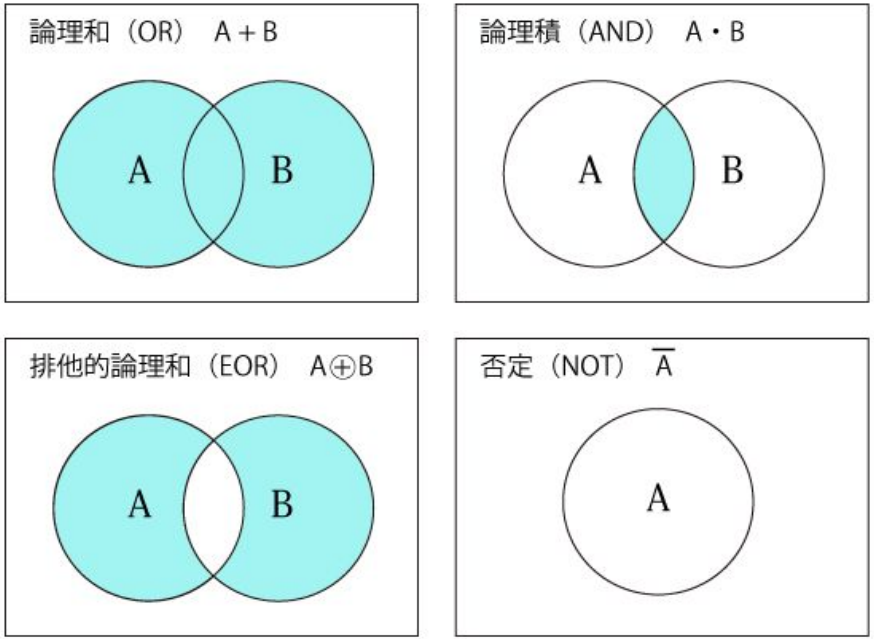
デジタル（ある一点を切り出してる）

論理演算

論理演算とは、真（true）と偽（false）の二通りの状態を取る**真偽値**（真理値/ブール値）の間で行われる演算。コンピュータでは真を1に、偽を0に対応付けた**ビット演算**として行われることが多い。

（論理演算 - IT用語辞典 e-Words　より）

こういうやつ



株式会社K-fix
http://www.k-fix.jp/skill/it/01/page08.html

ビットで考える

Y = A **OR** B

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Y = A **AND** B

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Y = A **XOR** B

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Y = **NOT** A

A	Y
0	**1**
1	**0**

排他的論理和(Exclusive OR)は,
XOR / EOR / ExOR などの表記がある

否定もある（出力にNOTかませる）

Y = A **NOR** B

A	B	Y
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

Y = A **NAND** B

A	B	Y
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

Y = A **XNOR** B

A	B	Y
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

論理素子（論理ゲート）

Y = A **OR** B

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Y = A **AND** B

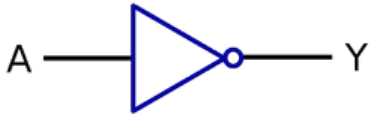
A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

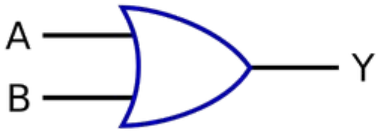
Y = A **XOR** B

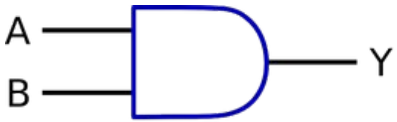
A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

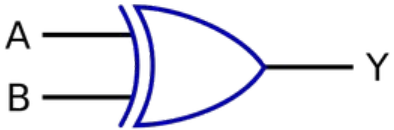
Y = **NOT** A

A	Y
0	**1**
1	**0**





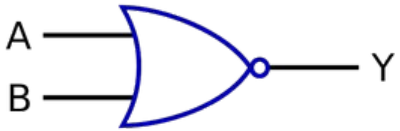




論理素子（論理ゲート）

Y = A **NOR** B

A	B	Y
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

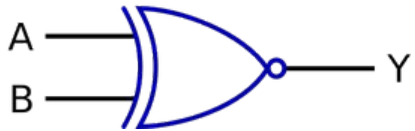


Y = A **NAND** B

A	B	Y
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

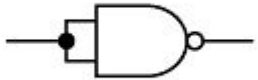
Y = A **XNOR** B

A	B	Y
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

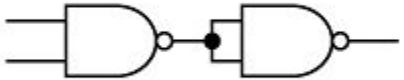


NAND素子は何にでもなれる（余談）

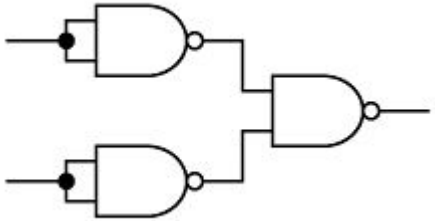
NOT



AND



OR



色んな素子作るより全部NAND素子で賄えた方が安上り

（USBメモリとかSDカードとか，NANDフラッシュメモリが使われてる）

図：NAND - MONOist
https://monoist.atmarkit.co.jp/mn/articles/0803/26/news136.html

2進数での論理演算（ビット演算）

$10011010_{(2)}$

OR

AND

XOR

:

$00101001_{(2)}$

ビット演算 - OR

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

$10011010_{(2)}$ **OR** $00101001_{(2)}$

A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

同じ重み同士で演算

算数の筆算みたいな感じ

桁数が違ったら足りない部分を 0で埋める

ビット演算 - OR

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

$10011010_{(2)}$ **OR** $00101001_{(2)}$

A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Y =

$10111011_{(2)}$

ビット演算 - AND

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

$$10011010_{(2)} \ \textbf{AND} \ 00101001_{(2)}$$

A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

$$Y = 00001000_{(2)}$$

ビット演算 - XOR

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

$10011010_{(2)}$ **XOR** $00101001_{(2)}$

A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**

Y =

$10110011_{(2)}$

A	Y
0	**1**
1	**0**

ビット演算 - NOT

NOT $10011010_{(2)}$

A	1	0	0	1	1	0	1	0
Y	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**

Y =

$01100101_{(2)}$

ビット演算 - NOT - XORで実現

A	B	Y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

NOT $10011010_{(2)}$ = $10011010_{(2)}$ **XOR** $11111111_{(2)}$

A	1	0	0	1	1	0	1	0
B	1	1	1	1	1	1	1	1
Y	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**

Y =

$01100101_{(2)}$

ビット演算 - シフト

右シフト $10011010_{(2)}$

A	1	0	0	1	1	0	1	0
Y	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**

右シフトは ÷ 2

$$Y = 01001101_{(2)}$$

論理シフトや算術シフトの違い

ローテートの概念は割愛

ビット演算 - 任意のbitを取り出す

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

$10011010_{(2)}$ **AND** $00001111_{(2)}$

A	1	0	0	1	1	0	1	0
B	0	0	0	0	1	1	1	1
Y	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**

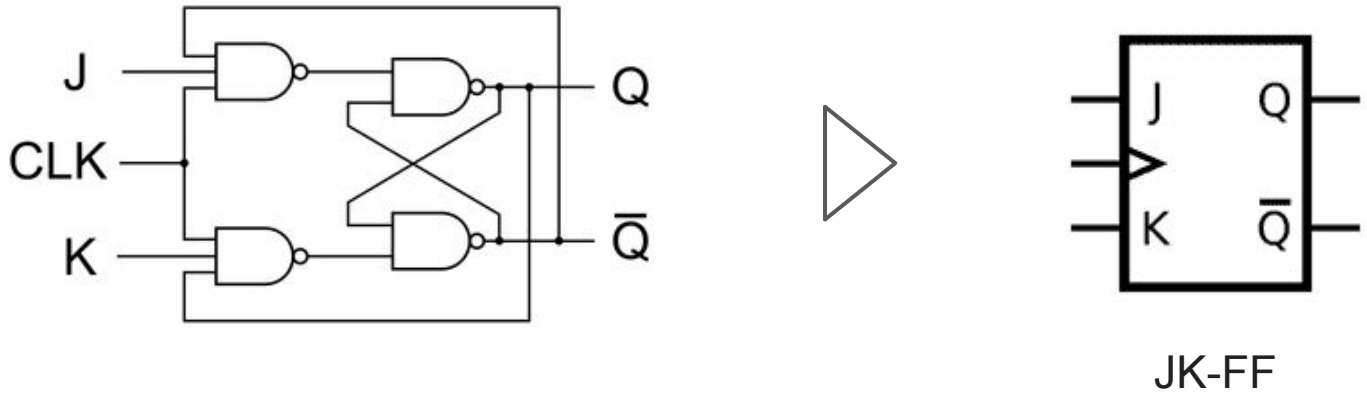
下位4bitのみ取り出す

$Y = 00001010_{(2)}$

フリップフロップ（FF：Flip-Flop）

フリップフロップ回路とは、最も基本的な構造の論理回路の一つで、二つの状態（通常「0」および「1」に対応付けられる）のいずれかを保持することができるもの。現在の入力と共に過去の入力も利用する順序回路の一種で、**SRAM**やマイクロプロセッサ（CPU/MPU）内部のレジスタ、キャッシュメモリなどに応用されている。

（フリップフロップ回路 - IT用語辞典 e-Words より）



JK-FF

2進数での足し算（＋）

$$10011010_{(2)} + 00101001_{(2)}$$

2進数での足し算（＋）

$$10011010_{(2)} + 00101001_{(2)}$$

A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**

2進数での足し算（＋）

$$10011010_{(2)} + 00101001_{(2)}$$

				1				
A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**

1 + 1 = 10

2進数での足し算（＋）

$$10011010_{(2)} + 00101001_{(2)}$$

			1		1			
A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**

1 + 1 = 10

2進数での足し算（＋）

$$10011010_{(2)} + 00101001_{(2)}$$

		1		1		1		
A	1	0	0	1	1	0	1	0
B	0	0	1	0	1	0	0	1
Y	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**

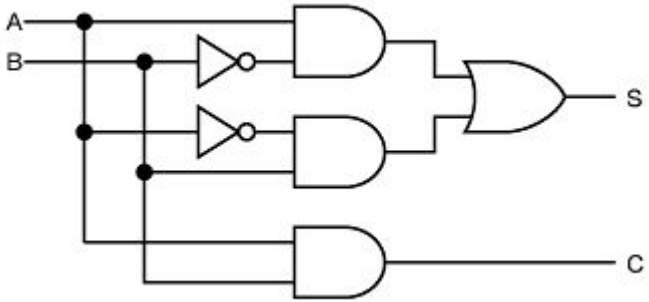
1 + 1 = 10

加算機

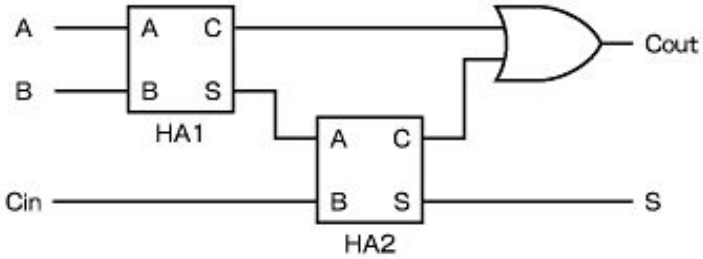
半加算器は2つの入力ビットを足し算して桁上がり(C)と解(S)を出すもの（2in-2out）

全加算器は2つの入力ビットと前段の桁上がりを考慮した加算機（3in-2out）

半加算器（Half-Adder）



全加算器（Full-Adder）



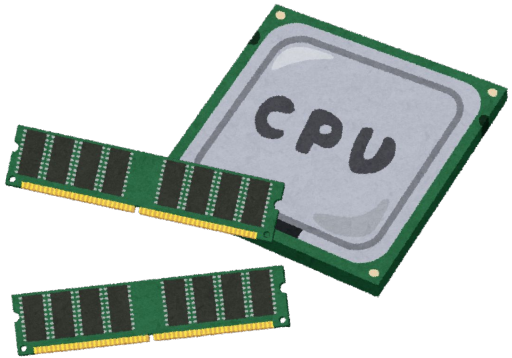
図：【問題9】ゲート回路の簡単化：完全マスター！ 電子回路ドリル II（9） - MONOist
https://monoist.atmarkit.co.jp/mn/articles/0803/13/news126.html

まとめ

電気信号と2進数の関係について学んだ

ビット演算はコンピュータの中でうまくビット列を扱うために使われる

コンピュータの実体はクソデカ論理回路



8進数と16進数（余談）

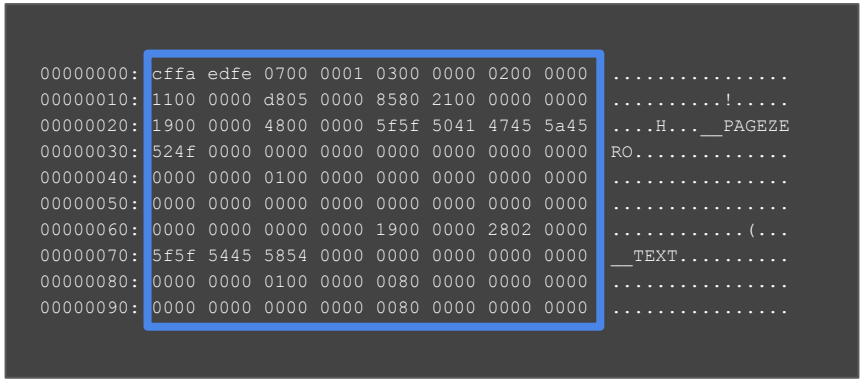
2進数や10進数以外にも，いくつかよく使われる進数表現がある

- **8進数**
  - 0~7が使える
  - n桁目の重みは8^(n-1)
  - 000 000 000 … のように，2進数を3桁ごとに区切ると変換しやすい（2進3桁は0~7を表現可能）
  - 011 のように表記する（これは 8進数で11の意味）（OctalのO→0）

- **16進数**
  - 0~9,A~Fが使える（A:11, B:12, C:13, D:14, F:15）
  - n桁目の重みは16^(n-1)
  - 0000 0000 0000 … のように，2進数を4桁ごとに区切ると変換しやすい（2進4桁は0~15を表現可能）
  - 0x1Fのように表記する（これは 16進数で1Fの意味）（HexadecimalのX）
  - 人間が結構読めるのでバイナリ形式とかはこれで表現したのをよく見る

8進数と16進数（余談）-「コンピュータとは」より

.exeファイル（実行ファイル）の実体



この部分が16進数

ビット（bit）とバイト（Byte）（余談）

	1 B(Byte) = 8 bit
キロ	1 KB = 10^3 B = 1000 B
メガ	1 MB = 10^6 KB = 1000000 B
ギガ	1 GB = 1000 MB = 10^9 B = 10000000000 B
テラ	1 TB = 1000 GB = 10^12 B = 1000000000000 B
ペタ	1 PB = 1000 TB = 10^15 B = 1000000000000000 B
エクサ	1 EB = 1000 PB = 10^18 B = 1000000000000000000 B
ゼタ	1 ZB = 1000 EB = 10^21 B = 1000000000000000000000 B
ヨタ	1 YB = 1000 ZB = 10^24 B = 1000000000000000000000000 B

KB（キロバイト）とKiB（キビバイト）（余談）

10進ベースだと 1 KB = 10^3 B = 1000 B

2進ベースだと 1 KiB = 2^10 B = 1024 B

コンピュータは2進ベースで動いている

メビバイト	1 MiB = 1024 KB = 2^20 B
ギビバイト	1 GiB = 1024 MB = 2^30 B
ティビバイト	1 TiB = 1024 GB = 2^40 B