

東北大学 工学部 機械知能・航空工学科  
2018年度 クラスC3 D1 D2 D3

# 情報科学基礎 I

## 4. 論理値と論理演算 (教科書2.1節, 2.4節)

大学院情報科学研究科

鏡 慎吾

<http://www.ic.is.tohoku.ac.jp/~swk/lecture/>

# 内容

- **論理演算**
- **論理値 0 と 1**
- **論理演算 AND, OR, NOT**
- **よく使う他の論理演算 NAND, NOR, XOR**
- **論理ゲート回路**
  
- **ビットごと論理演算**

# 論理値と論理演算

## 論理値

- 「3 は 4 より小さい」は真 (true): 数値 1 で表す
- 「カエルは哺乳動物である」は偽 (false): 数値 0 で表す

## 論理演算:

- 論理積 (AND):  $A \cdot B$  あるいは単に  $AB$   
他の記法:  $A \wedge B, A \& B$
- 論理和 (OR):  $A + B$   
他の記法:  $A \vee B, A | B$
- 論理否定 (NOT):  $\overline{A}$   
他の記法:  $\neg A, A', !A, \sim A$

# 真理値表とゲート記号

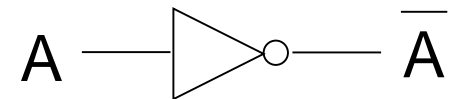
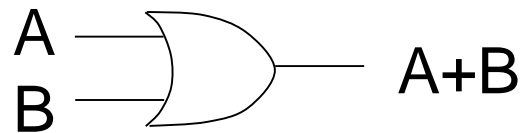
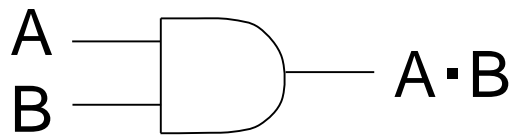
## 真理値表

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

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

A	$\bar{A}$
0	1
1	0

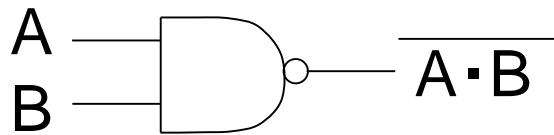
## ゲート記号



# よく使う他の論理演算

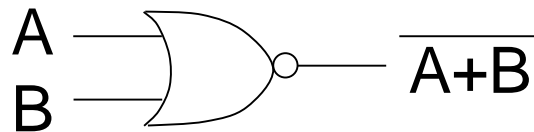
否定論理積  
(NAND)

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



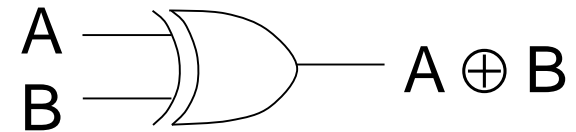
否定論理和  
(NOR)

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



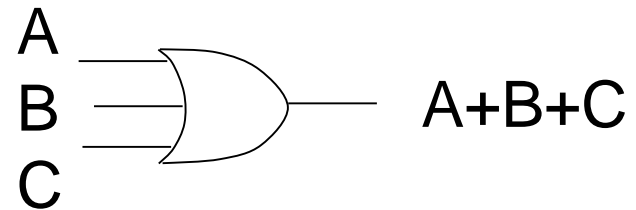
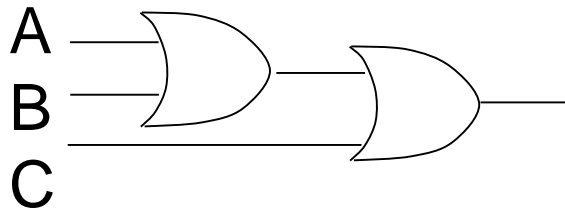
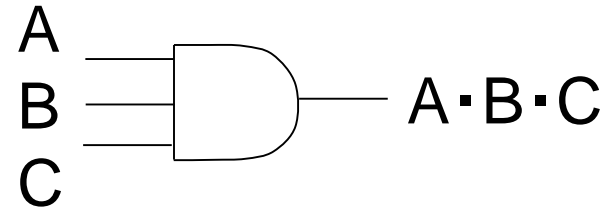
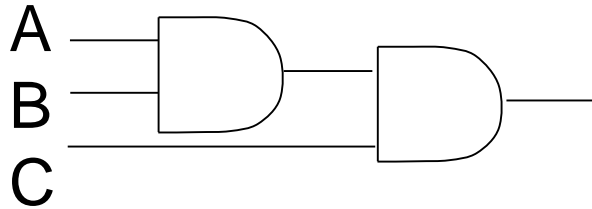
排他的論理和  
(eXclusive OR)  
(XOR)

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



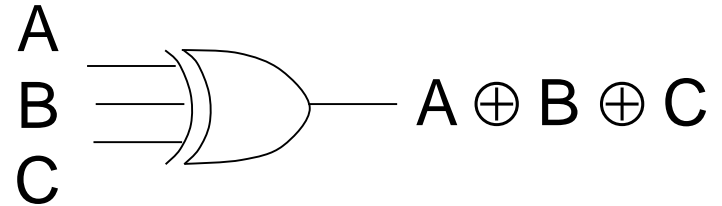
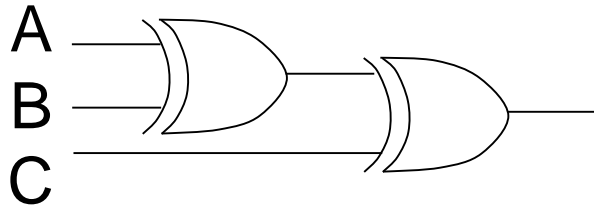
- 片方だけ 1 のときに限って 1
- つまり入力が相異なるときに 1

# 多入力AND, 多入力OR



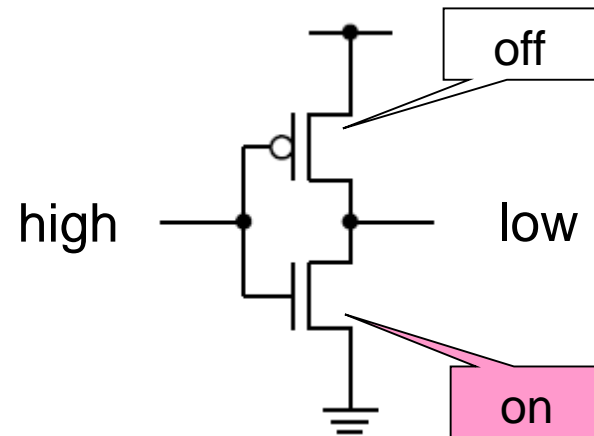
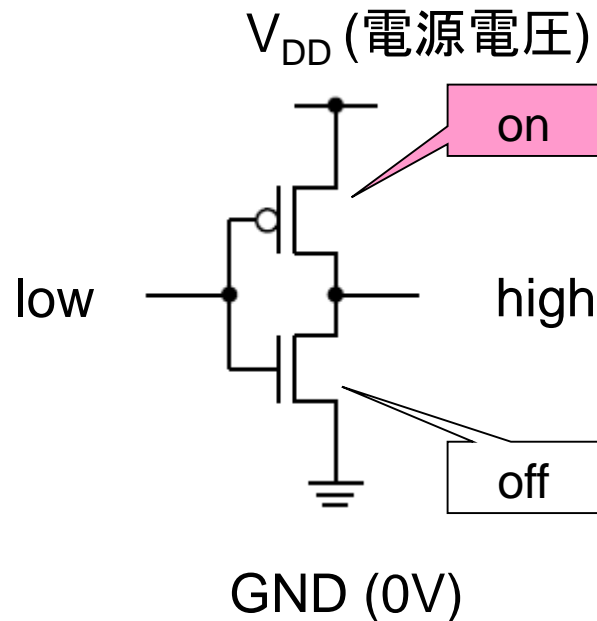
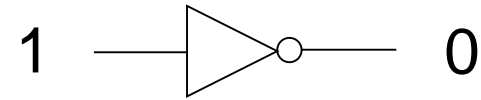
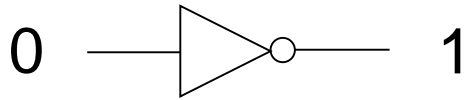
- 論理積も論理和も, 交換則 ( $AB = BA$ ,  $A+B = B+A$ ) と結合則 ( $(AB)C = A(BC)$ ,  $(A+B)+C = A+(B+C)$ ) が成り立つ.
- 入力の順序に関係なく,
  - 多入力ANDは入力が一つでも 0 なら 0
  - 多入力 OR は入力が一つでも 1 なら 1
- NAND, NOR も同様

# 多入力XOR



- $X \oplus 1 = \overline{X}$
- $X \oplus 0 = X$
- A, B, C, ... と順に見ていって, 入力に 1 が現れる度に出力は反転する
- 結局, 多入力 XOR は, 入力のうち 1 の数が奇数なら 1に, 偶数なら 0 になる

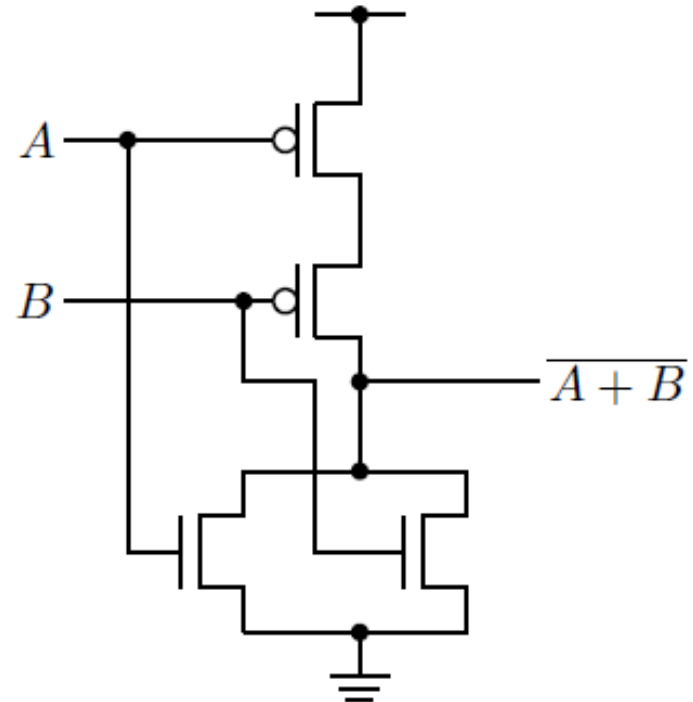
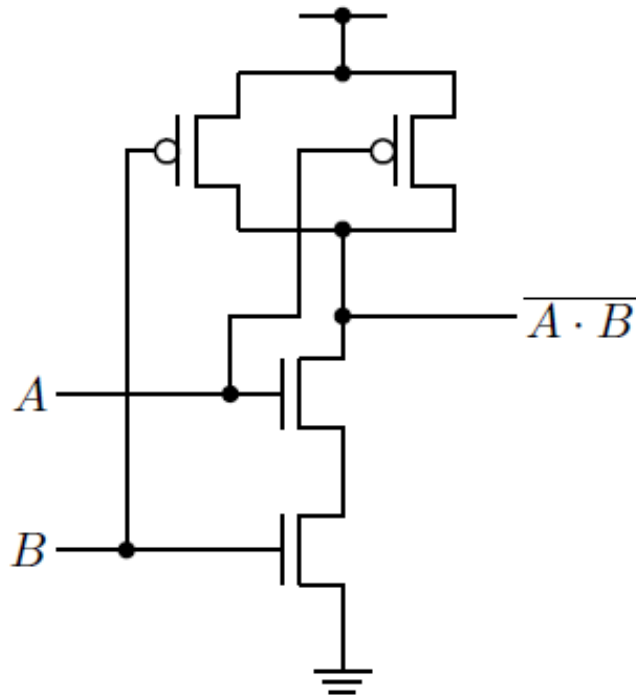
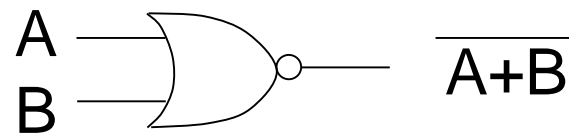
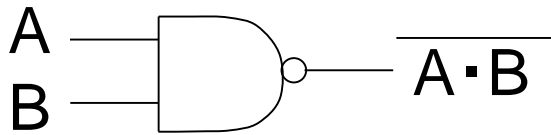
# NOTゲートの回路 (インバータ)



CMOS技術 (Complementary MOS):  
NMOS と PMOS を常に対にして使う (低消費電力)



# NAND, NORゲートの回路



- AND, OR は NAND, NOR, NOT から作れる
- XOR は AND, OR, NOT から作れる

# プログラミングで使う論理演算

```
if (x > 20 && x < 80) {  
    ...  
}
```

C言語では, 比較演算子 (>, >=, <, <=, ==, !=) は真のとき整数 1 を, 偽のとき整数 0 を返す.

```
if (a < 0 || b < 0) {  
    ...  
}
```

演算子 &&, ||, ! が, 論理積, 論理和, 論理否定を行う.

```
if (!isalpha(c)) {  
    ...  
}
```

if や while などの条件部は, 0 を偽, 0 以外を真とみなして判定する.

# ビットごとと論理演算

C言語の演算子  $\&$ ,  $|$ ,  $\sim$ ,  $\wedge$  などは,  $\&\&$ ,  $\|\|$ ,  $!$  とは異なり ビットごとの論理演算を行う

```
a = 0x1234;    // 0001 0010 0011 0100
b = 0xcafe;    // 1100 1010 1111 1110
c = a & b;     // 0000 0010 0011 0100 (AND)
c = a | b;     // 1101 1010 1111 1110 (OR)
c = ~a;       // 1110 1101 1100 1011 (NOT)
c = a ^ b;    // 1101 1000 1100 1010 (XOR)
```

任意のビットは

- 1 との OR を取ると 1 になる; 0 との OR は元のまま不変
  - 0 との AND を取ると 0 になる; 1 との AND は元のまま不変
  - 1 との XOR を取ると反転する; 0 との XOR は元のまま不変
- などを利用して, ビットごとの操作ができる

# 例題

ファミリーコンピュータ用ゲーム「ドラゴンクエストIV 導かれし者たち」((株) エニックス, 1990年) では, 敵との戦闘中に「にげる」操作を8 回行くと, それ以降, 敵へのすべての攻撃が強力なものとなる(会心の一撃と呼ばれる) 現象が生じた. 内部でどのような処理が行われていたか推測して述べよ.

(2010年度 期末試験)



[http://www.youtube.com/watch?v=\\_Ao1ofpl0mE](http://www.youtube.com/watch?v=_Ao1ofpl0mE)

# 例題 解答例

ドラゴンクエストIV (エニックス) の戦闘状態を保持するメモリ領域の一部は、以下のような構成だったと推測されている:

霧フラグ	?	?	?	会心フラグ	「にげる」コマンドカウンタ		
7	6	5	4	3	2	1	0

```
critical_hit_bit = 1 << 3; // 0000 1000
fog_bit = 1 << 7; // 1000 0000
```

```
status = status | critical_hit_bit; // 会心の一撃ON
status = status & ~critical_hit_bit; // 会心の一撃OFF
status = status ^ fog_bit; // 霧状態を反転
```

```
if (status & fog_bit) {
    // 霧状態ならこの処理を実行
}
```

# 練習問題

1. 16ビットの値  $x$  が与えられたとき, 上位 12 ビットと下位 4 ビットを入れ替えた結果を変数  $y$  に得る処理を, シフト演算とビットごと論理演算の組合せで実現せよ. (このような処理は「4 ビット右ローテート」と呼ばれる)
2. 16ビットの値が与えられたとき, 下位から 7 ビット目 (ただし LSB を 1 ビット目と数えることにする) の値を変数  $y$  に得る処理を, シフト演算とビットごと論理演算の組合せで実現せよ. (if 文などは使わない)

# 練習問題 解答例

C言語であれば, 以下のような処理で実現できる.

1. `unsigned short x, y;`

`...`

`y = (x << 12) | (x >> 4);`

2. `y = (x & (1 << 6)) >> 6;`