

論理回路

- **組合せ回路** :
入力 (の組) によって出力が決まる
→ **演算**に用いる

- **順序回路** :
内部状態を保持し、
入力と入力前の状態とによって
出力と出力後の状態とが決まる
→ **データ (bit 情報) の保持**に用いる

論理回路の基本部品 : 論理素子 (論理ゲート)

- NOT: 否定 : $\neg A$
- OR: 論理和 : $A \vee B$
- AND: 論理積 : $A \wedge B$
- XOR: 排他的論理和 :
 $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
- NOR: 論理和の否定 :
 $\neg(A \vee B) = \neg A \wedge \neg B$
- NAND: 論理積の否定 :
 $\neg(A \wedge B) = \neg A \vee \neg B$

論理素子の真理値表

NOT		X
A	0	1
	1	0

OR		B	
		0	1
A	0	0	1
	1	1	1

AND		B	
		0	1
A	0	0	0
	1	0	1

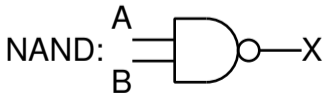
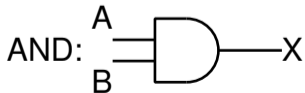
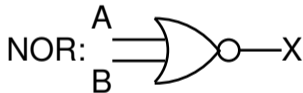
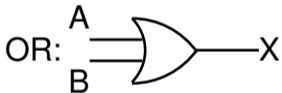
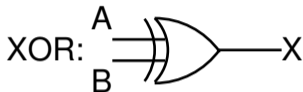
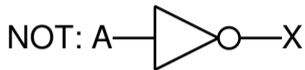
XOR		B	
		0	1
A	0	0	1
	1	1	0

NOR		B	
		0	1
A	0	1	0
	1	0	0

NAND		B	
		0	1
A	0	1	1
	1	1	0

論理素子の MIL 記号

論理素子を回路図で表現するのに用いる記号



論理素子

どんな論理回路も

これらの論理素子の組合せで表せるか？

→ “論理回路・論理素子が表すもの”
の定式化が必要

→ “**Boole 関数**” (真理値 (の組) を値とする関数)

組合せ回路

入力 (の組) によって出力が決まる

n 入力 m 出力の回路は **Boole** 関数

$$f : X_f \longrightarrow \{0, 1\}^m$$

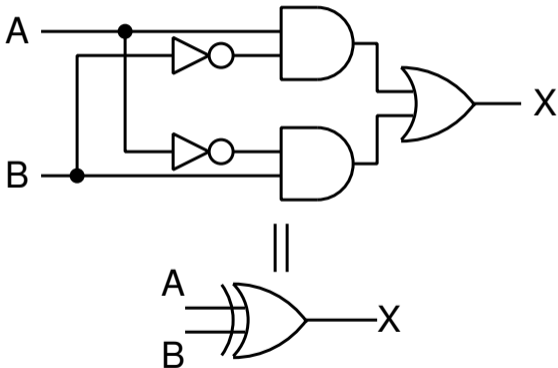
(ここに $X_f \subset \{0, 1\}^n$ は許される入力全体)
を定める

組合せ回路 : **Boole** 関数の論理素子による実現

以下では、NOT, OR, AND を用いた実現を考える

組合せ回路

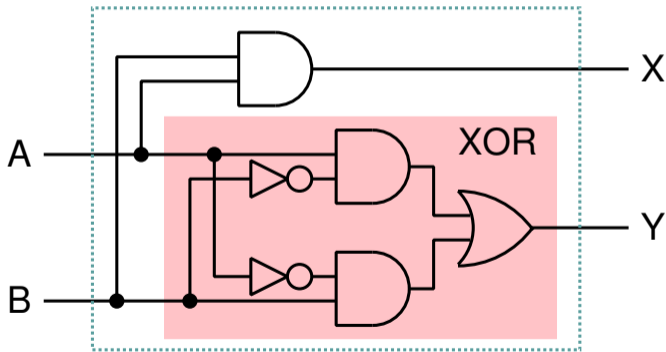
例: XOR を NOT, OR, AND で表す



半加算器 (semi adder, SA)

入力 : **A, B**: 各桁の値

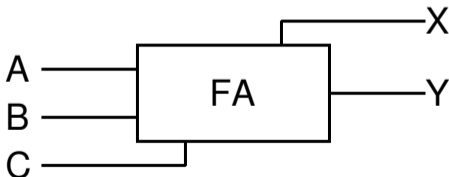
出力 : **X**: 上への繰上がり, **Y**: 当桁の値



演習問題

全加算器 (full adder, FA) を、
NOT, OR, AND を用いて構成せよ

- 入力 : **A, B**: 各桁の値, **C**: 下からの繰上がり
- 出力 : **X**: 上への繰上がり, **Y**: 当桁の値



演習問題

全加算器 (**full adder**) を部品として用いて、
二進 4 桁の符号付き整数値 2 つの
加算を行なう組合せ回路を構成せよ。
但し、桁溢れの発生を判定し、
桁溢れが生じた場合は **overflow flag** を立てよ。

- 入力 : $A_1, A_2, A_3, A_4; B_1, B_2, B_3, B_4$
 : 数値 (A_1, B_1 は符号 bit)
- 出力 :
 X_1, X_2, X_3, X_4 : 加算の結果 (X_1 は符号 bit)
 Y : **overflow flag**
 (桁溢れが発生したら 1、しなければ 0)

(再掲) 論理回路

- データ (bit 情報) の保持 → 順序回路
- 基本的な演算 → 組合せ回路

組合せ回路

入力 (の組) によって出力が決まる (演算回路)

n 入力 m 出力の回路は **Boole** 関数

$$f : X_f \longrightarrow \{0, 1\}^m$$

(ここに $X_f \subset \{0, 1\}^n$ は許される入力全体)

順序回路

- 内部状態を保持し、
- 入力と入力前の状態とによって、
- 出力と出力後の状態が決まる

許される内部状態 : $Q_f \subset \{0, 1\}^B$

n 入力 m 出力の順序回路は **Boole** 関数

$$f : Q_f \times X_f \longrightarrow Q_f \times \{0, 1\}^m$$

(ここに $X_f \subset \{0, 1\}^n$ は許される入力全体)
を定める

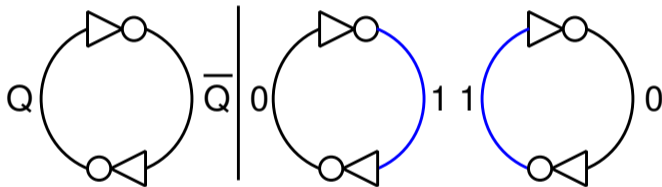
順序回路

内部状態を計算機内に如何に保持するか

- コンデンサに電荷を蓄える
- 複数の安定状態を持つ論理回路で実現
例: フリップフロップ

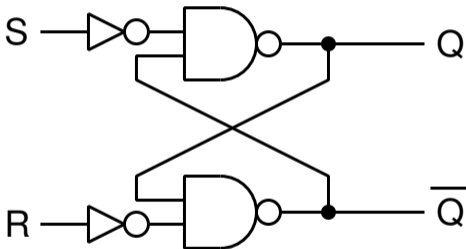
フリップフロップ (flip-flop)

原理図:



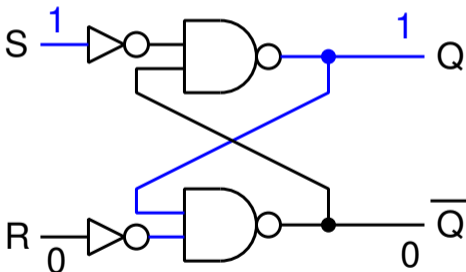
これに入出力端子を付ける

SR-フリップフロップ (Set-Reset)



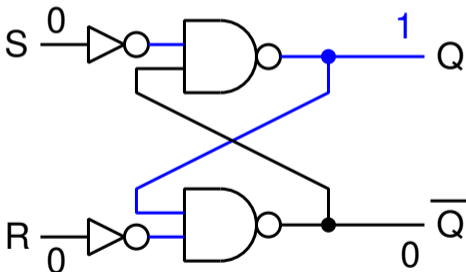
SR-フリップフロップ (Set-Reset)

$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$



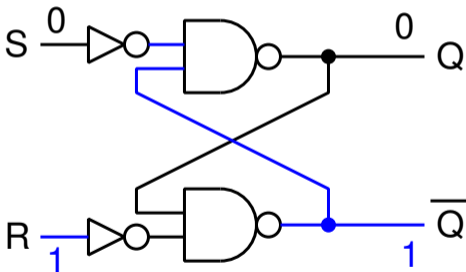
SR-フリップフロップ (Set-Reset)

$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$



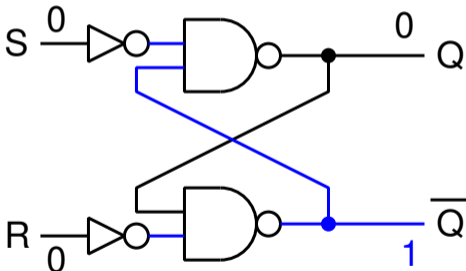
SR-フリップフロップ (Set-Reset)

$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$



SR-フリップフロップ (Set-Reset)

$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$



SR-フリップフロップ (Set-Reset)

$(S, R) = (1, 1)$ は禁止入力

$(X_f = \{(0, 0), (0, 1), (1, 0)\})$

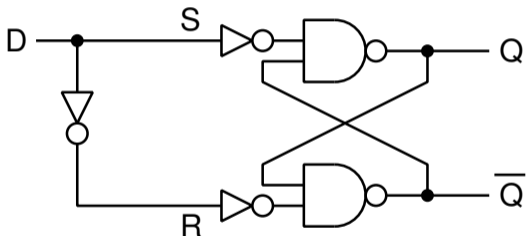
S	R	Q	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	x

SR-フリップフロップ (Set-Reset)

入力が $(S, R) = (1, 1)$ とならない回路設計

→ $S = \bar{R}$ となるようにしてみる



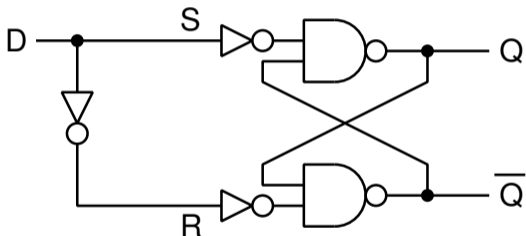
→ 入力 D を保持・出力

但し、これだと D をそのまま流しているのと同じ

SR-フリップフロップ (Set-Reset)

入力が $(S, R) = (1, 1)$ とならない回路設計

→ $S = \bar{R}$ となるようにしてみる



→ 入力 D を保持・出力

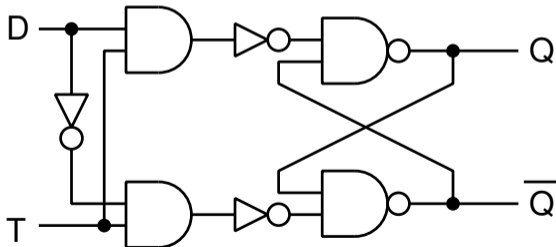
但し、これだと D をそのまま流しているのと同じ

SR-フリップフロップ (Set-Reset)

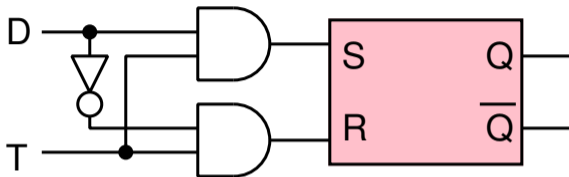
これだと D をそのまま流しているのと同じ

→ 書込スイッチを付けよう

- T : スイッチ入力
 - ★ T = 0 : そのまま
 - ★ T = 1 : D を取り込む



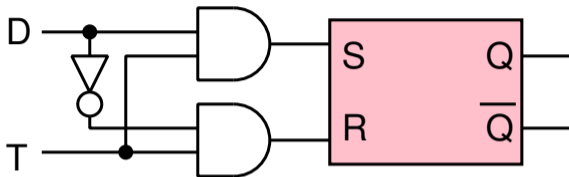
SR-フリップフロップ (Set-Reset)



T	D	S	R	Q
0	0	0	0	Q
0	1	0	0	Q
1	0	0	1	0
1	1	1	0	1

T	D	S	R	Q
0	*	0	0	Q
1	*	D	\overline{D}	D

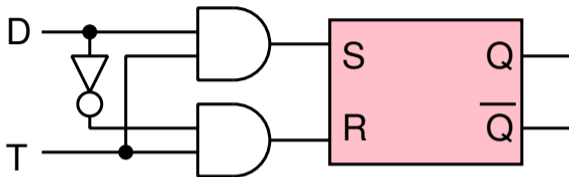
SR-フリップフロップ (Set-Reset)



- Q からの出力により他の値を計算
- 他からの入力により D を計算

→ Q が D に影響を与えることにより、
状態が変わってしまう

SR-フリップフロップ (Set-Reset)



- Q からの出力により他の値を計算
- 他からの入力により D を計算

→ Q が D に影響を与えることにより、
状態が変わってしまう

SR-フリップフロップ (Set-Reset)

回路の他の部分とタイミングを合わせる必要あり

→ 出力を一旦せき止めて、
一段階づつ計算を進める

→ D-フリップフロップ・クロックパルスの利用

SR-フリップフロップ (Set-Reset)

回路の他の部分とタイミングを合わせる必要あり

→ 出力を一旦せき止めて、
一段階づつ計算を進める

→ D-フリップフロップ・クロックパルスの利用

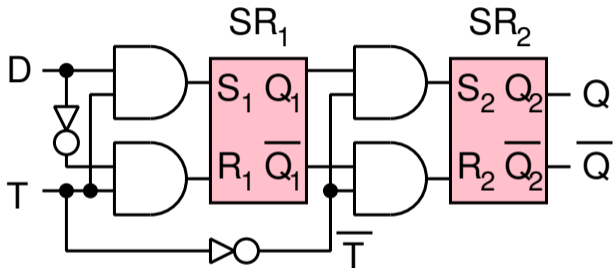
SR-フリップフロップ (Set-Reset)

回路の他の部分とタイミングを合わせる必要あり

→ 出力を一旦せき止めて、
一段階づつ計算を進める

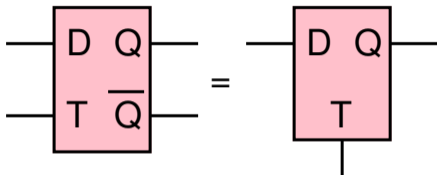
→ **D-フリップフロップ**・**クロックパルス**の利用

D-フリップフロップ (Double, Delay)



- $T = 1, \overline{T} = 0 \implies SR_1$: 取込、 SR_2 : 保持
- $T = 0, \overline{T} = 1 \implies SR_1$: 保持、 SR_2 : 取込

D-フリップフロップ (Double, Delay)



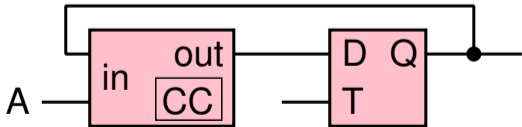
- $T = 1$: 入力 D を内部に取り込む
(出力 Q は変わらない)
- $T = 0$: 内部状態を Q に送り出す
(入力 D は受け取らない)

計算の実行

例: (CC は適当な組合せ回路・A は外部入力)

- $T = 1$: 入力 D を内部に取り込む
- $T = 0$: Q に出力 \rightarrow CC の入力へ
 \rightarrow CC での計算結果が D に到達
- $T = 1$: 新たな D を内部に取り込む

$\rightarrow T = 1 \rightarrow 0 \rightarrow 1$ と変化する度に、
計算が一段階進む



クロックパルス

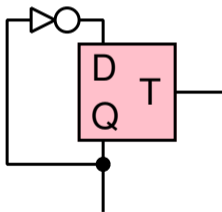
一定の周期で $1 \rightarrow 0 \rightarrow 1$ を繰り返す信号
(クロックパルス, **clock pulse**) を

水晶発振器などで発生させることにより、
回路全体での同期を取って、計算を進める

クロックパルスの周波数 \longrightarrow 計算機の動作の速さ
(動作 **clock** **Hz**)

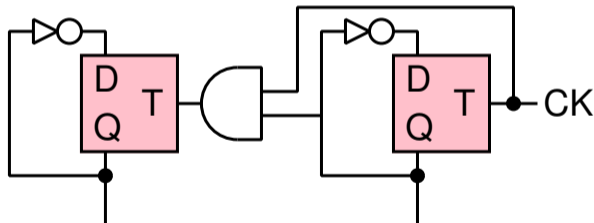
(二進) カウンタ

T が一周期変化する度に
内部状態が $1 \rightarrow 0 \rightarrow 1$ と変化する



四進カウンタ

T が一周期変化する度に、内部状態が
 $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ と変化する



計算機の機能

ここまでで紹介した

基本的な論理回路の部品を組合わせて、
以下の機能を実現する計算機を構成する

- データの保持・書込・読出 → 順序回路
- データの処理 (演算) → 組合せ回路
- (データの入出力)
- 実行の制御
 - ★ プログラムの読出
 - ★ 順次実行
 - ★ 条件分岐

「計算」の定式化

計算機に於ける「計算」の各ステップ
(= 命令の実行) は、

- 外部からの入力
- 内部状態 (メモリ・レジスタ) の現在の値
に従って、
- 外部への出力
- 内部状態 (メモリ・レジスタ) の値の変更
を行なうこと