論理回路の働き:

- データ (bit 情報) の保持
- 基本的な演算　　を計算機で実現

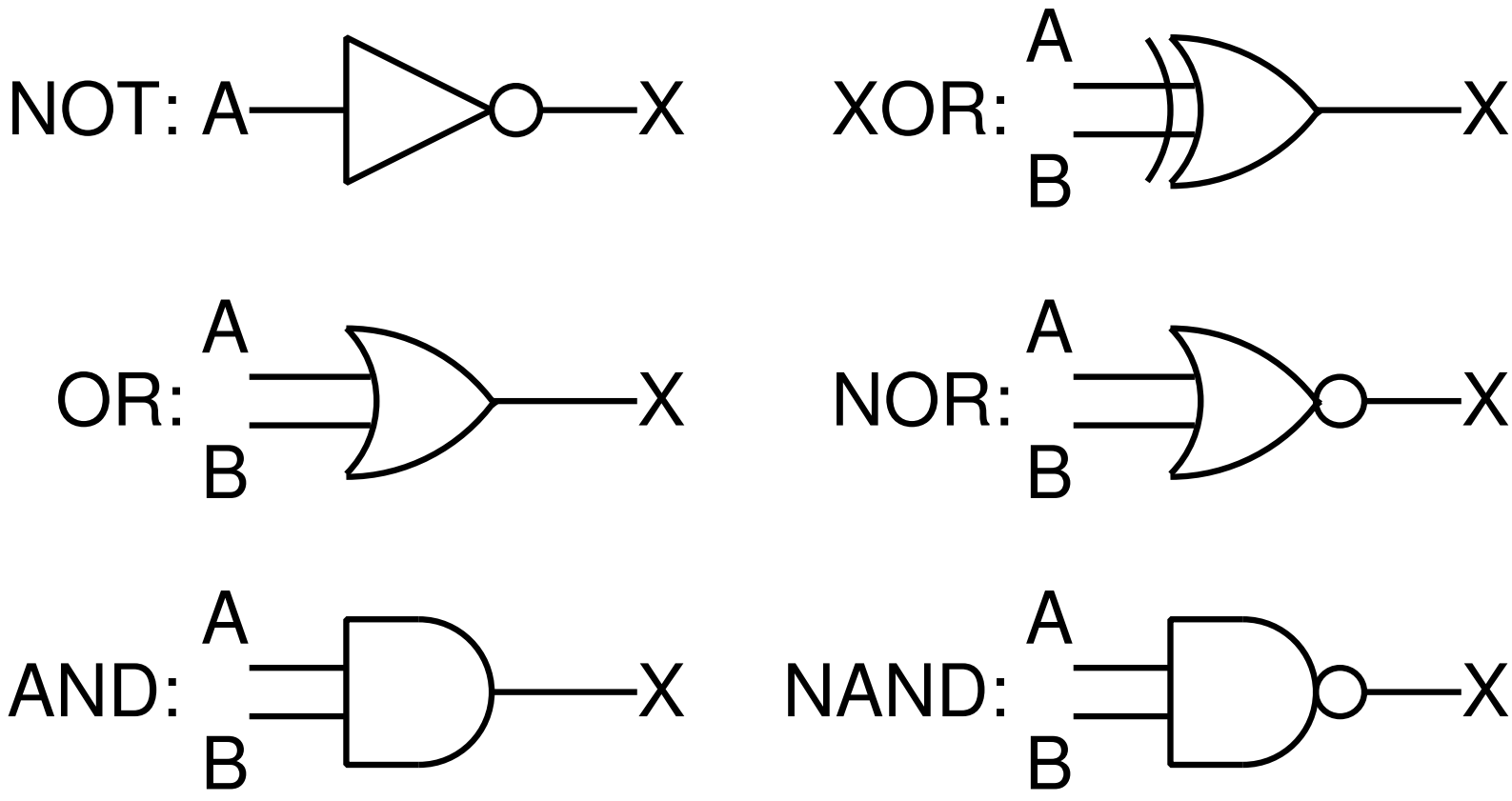
- 組合せ回路:
  入力 (の組) によって出力が決まる
  $\longrightarrow$ 演算に用いる

- 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
  $\longrightarrow$ 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 MIL 記号



論理素子の真理値表

NOT	X
0	1
1	0

OR	B	
	0	1
0	0	1
1	1	1

AND	B	
	0	1
0	0	0
1	0	1

XOR	B	
	0	1
0	0	1
1	1	0

NOR	B	
	0	1
0	1	0
1	0	0

NAND	B	
	0	1
0	1	1
1	1	0

論理回路の定式化

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?

$\longrightarrow$ “論理回路・論理素子が表すもの”
の定式化が必要

$\longrightarrow$ “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

組合せ回路

NOT, OR, AND **のみで、**
全ての Boole 関数が実現できる

$$f(A_1, \ldots, A_n) = (X_{11} \wedge \cdots \wedge X_{1t_1}) \vee \cdots \vee (X_{s1} \wedge \cdots \wedge X_{st_s})$$

(各 X_{ij} **は** A_k **または** $\neg A_k$**)**

$\cdots$ **論理和標準形・選言標準形**
(disjunctive normal form, DNF)

組合せ回路

双対的に、次の形でも書ける。

$$\begin{aligned} f(A_1, \ldots, A_n) = & (X_{11} \vee \cdots \vee X_{1t_1}) \\ & \wedge \cdots \\ & \wedge (X_{s1} \vee \cdots \vee X_{st_s}) \end{aligned}$$

(各 X_{ij} は A_k または $\neg A_k$)

$\cdots$ **論理積標準形・連言標準形**
(conjunctive normal form, CNF)

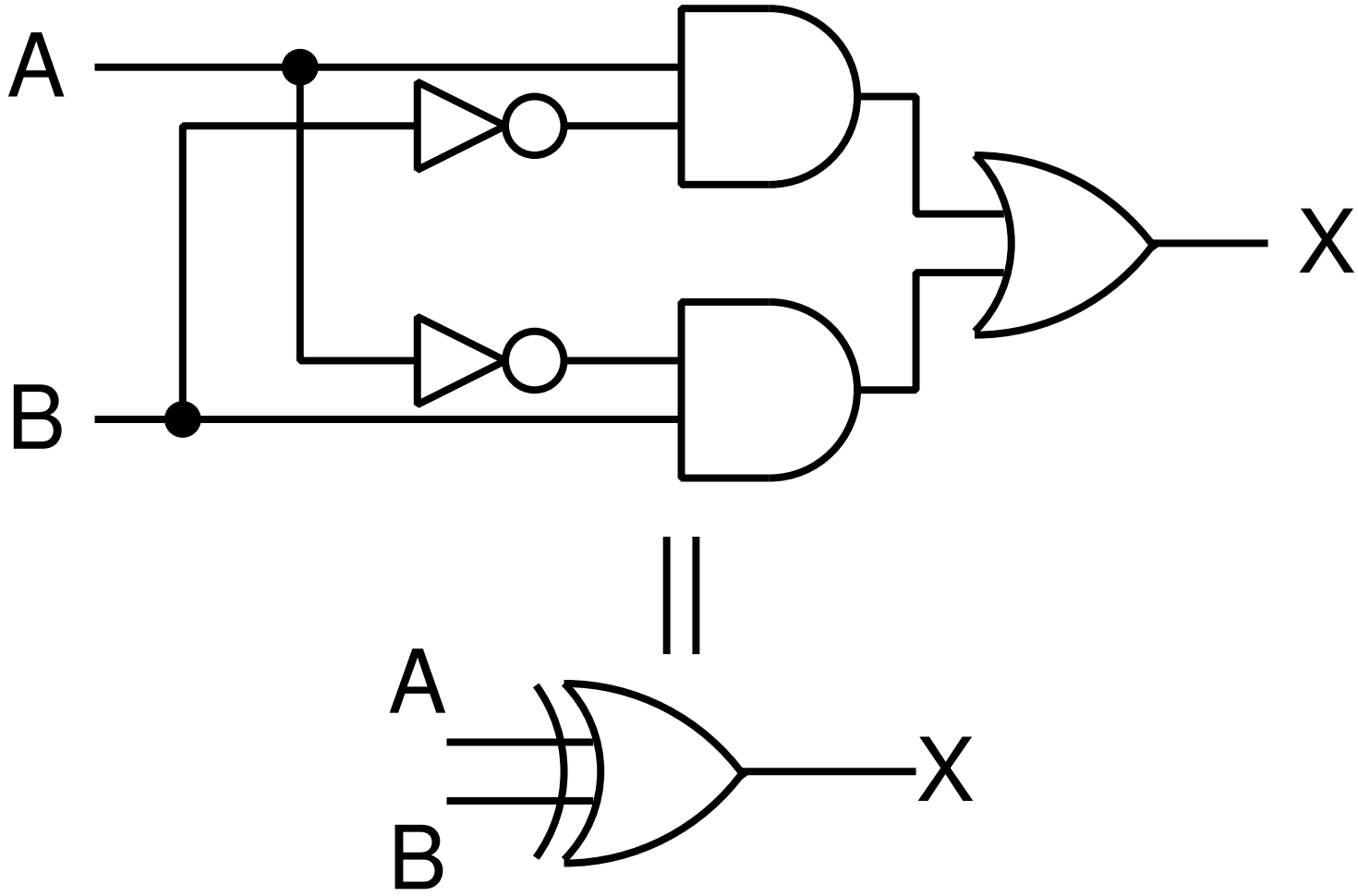
組合せ回路

- NOT, OR, AND のみで、
  全ての Boole 関数が実現できる
- NOT があれば OR, AND は片方で良い
- OR, AND だけでは
  全ての Boole 関数は実現できない
- NOR または NAND 一種類だけで、
  全ての Boole 関数が実現できる

以下では、
NOT, OR, AND を用いた実現を考える

組合せ回路

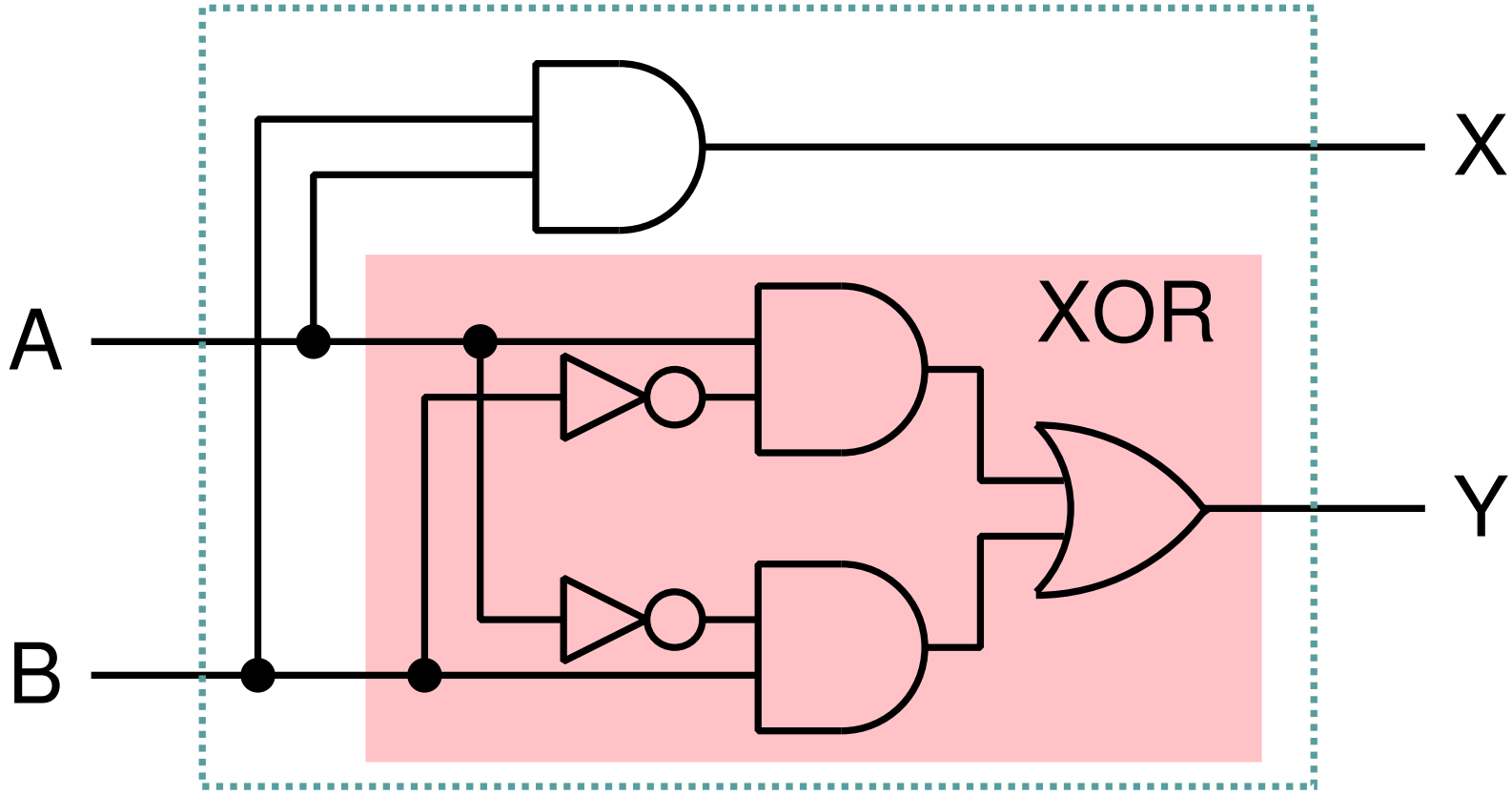
例: XOR **を** NOT, OR, AND **で表す**



半加算器 (semi adder, SA)

入力: A,B:各桁の値

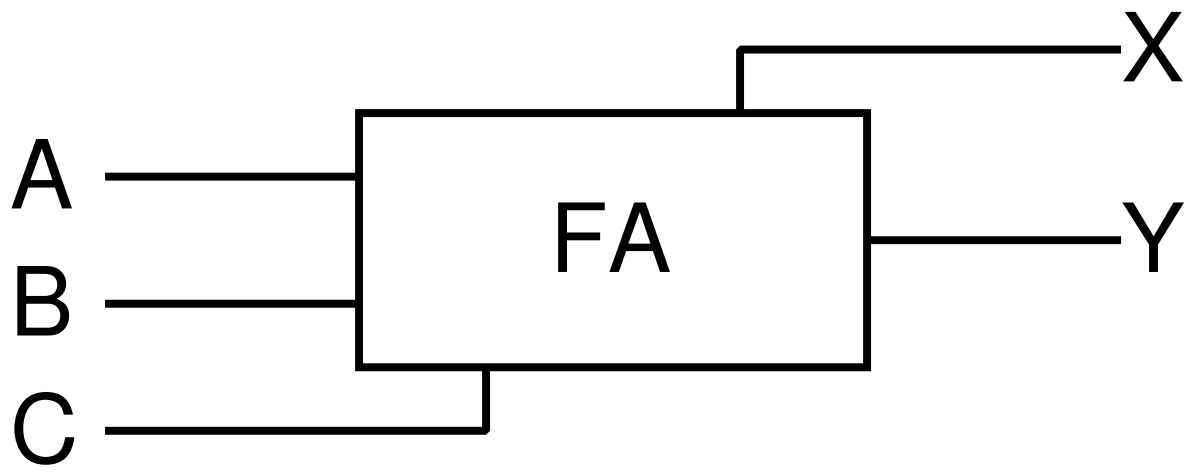
出力: X: 上への繰上がり, Y: 当桁の値



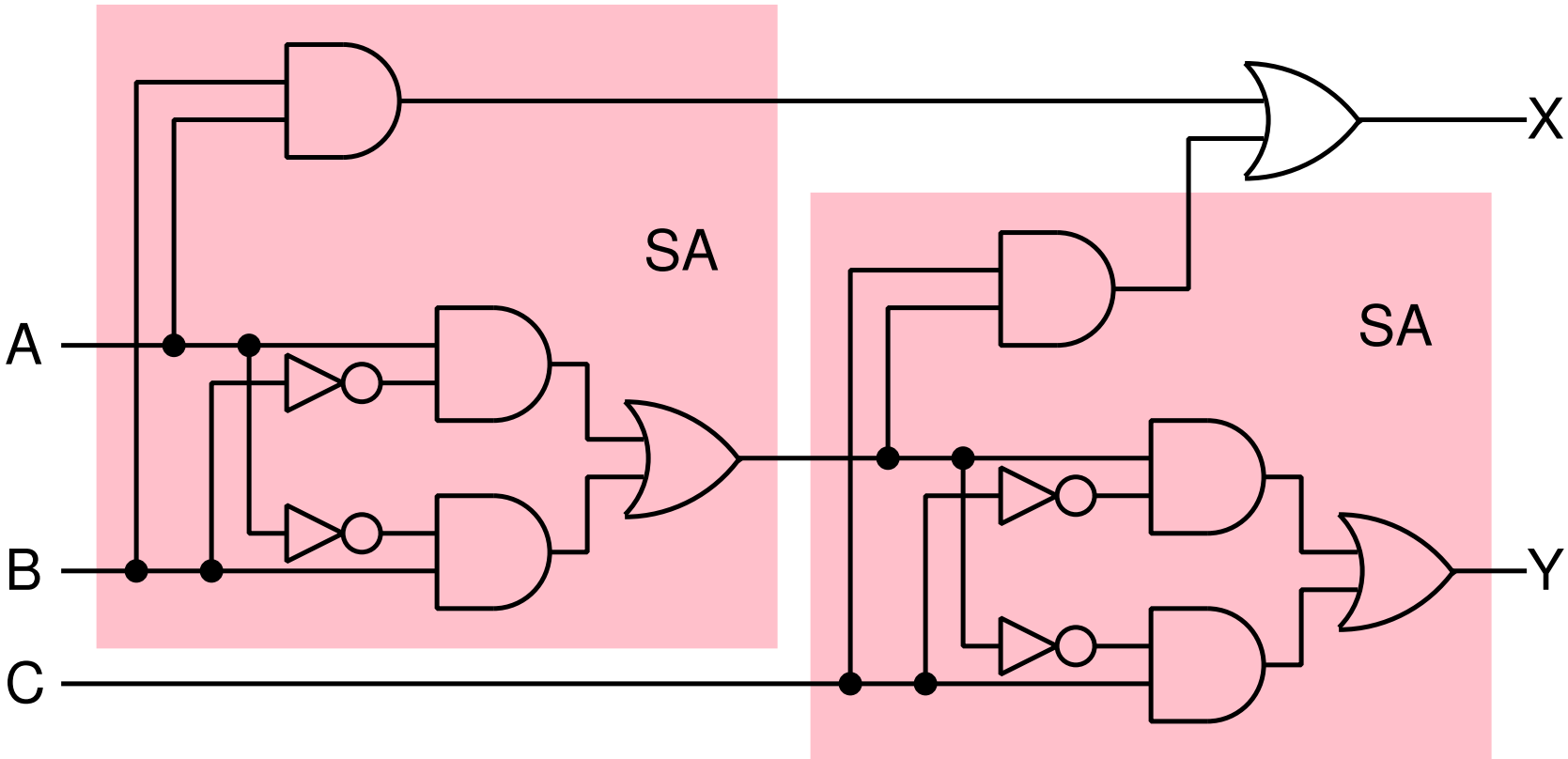
問題:

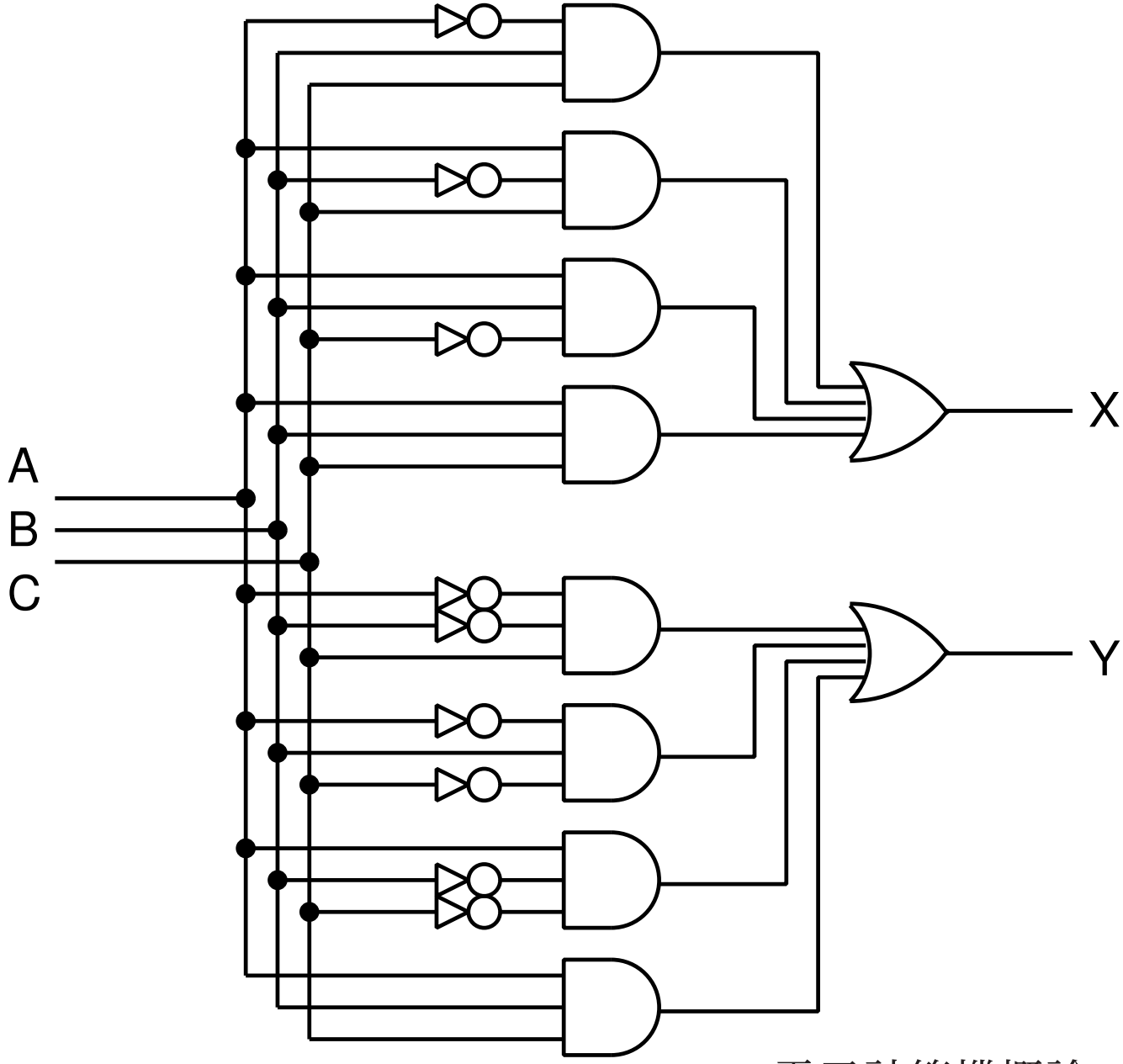
全加算器 (full adder, FA) を、
NOT, OR, AND **を用いて構成せよ。**

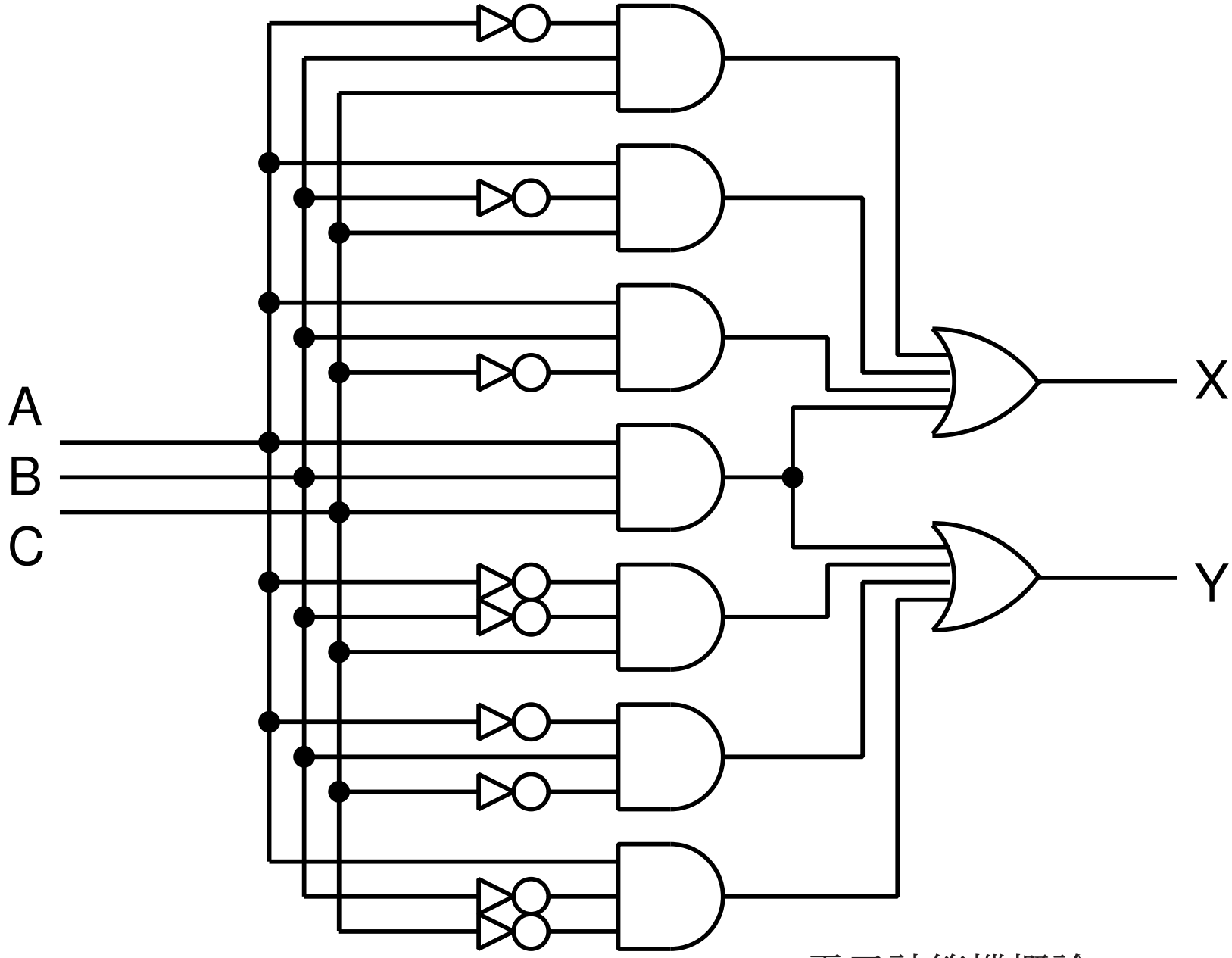
- **入力: A,B:各桁の値, C:下からの繰上がり**
- **出力: X: 上への繰上がり, Y: 当桁の値**



解答例:




A
B
C
X
Y


A
B
C
X
Y

(再掲) 論理回路

- データ (bit 情報) の保持 $\longrightarrow$ 順序回路
- 基本的な演算 $\longrightarrow$ 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)

(再掲) 論理回路

- データ (bit 情報) の保持 $\longrightarrow$ 順序回路
- 基本的な演算 $\longrightarrow$ 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

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

(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

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

(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

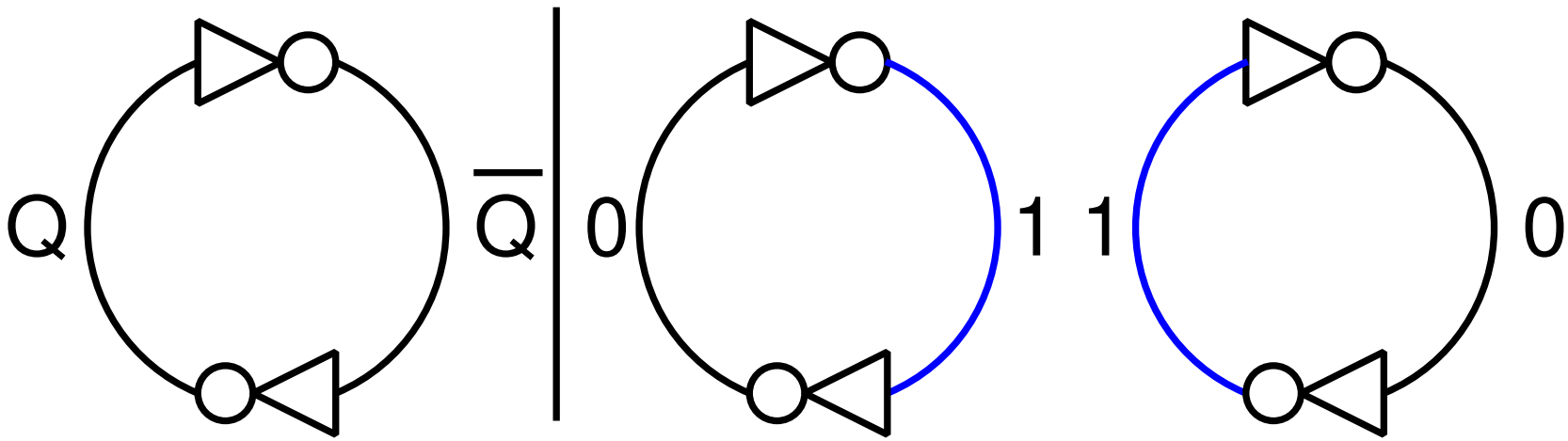
順序回路

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

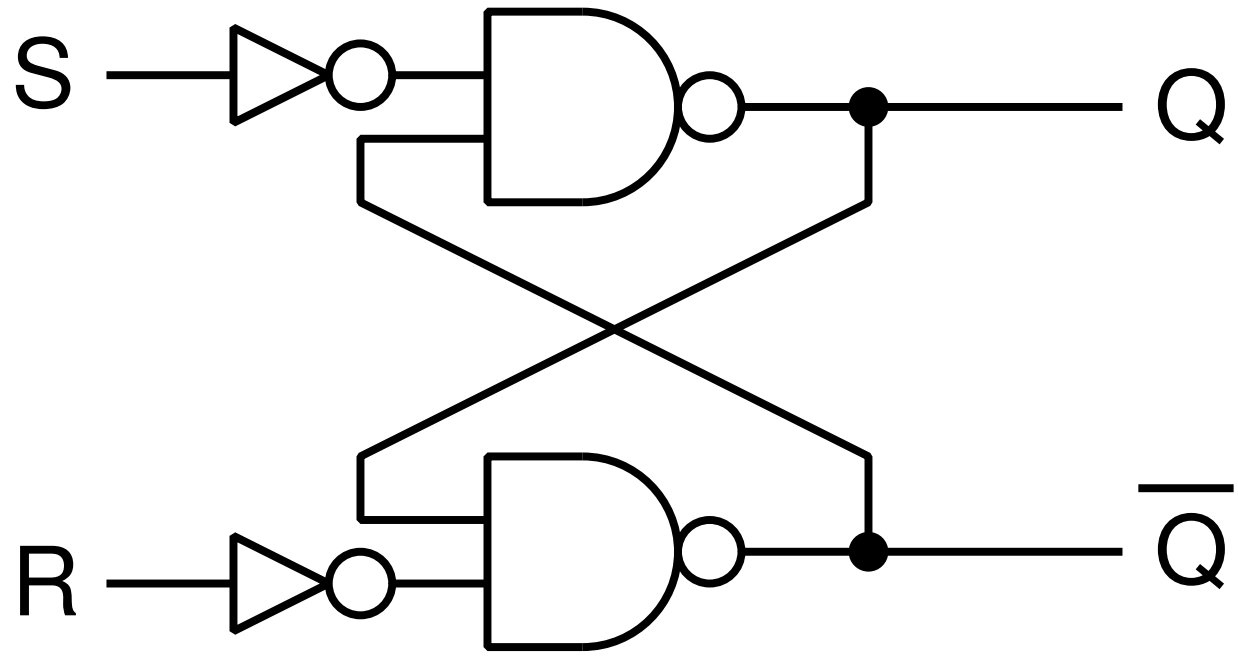
フリップフロップ (flip-flop)

原理図:



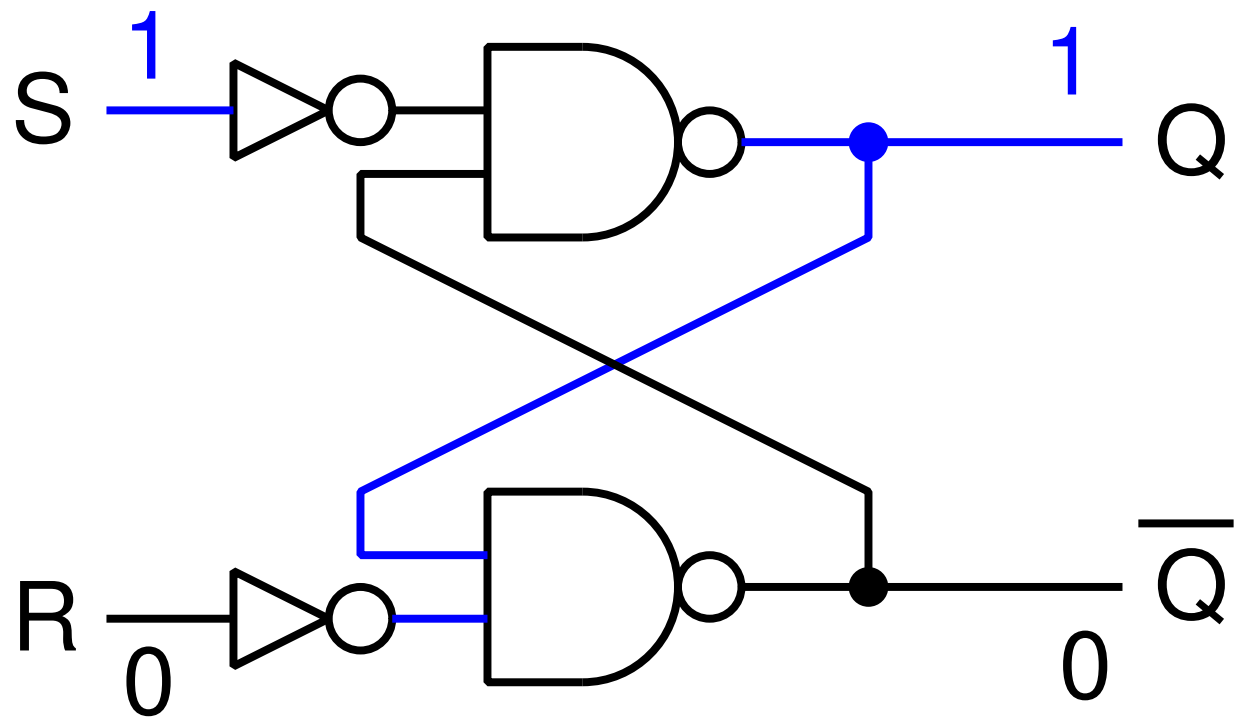
これに入出力端子を付ける

SR-フリップフロップ (Set-Reset)



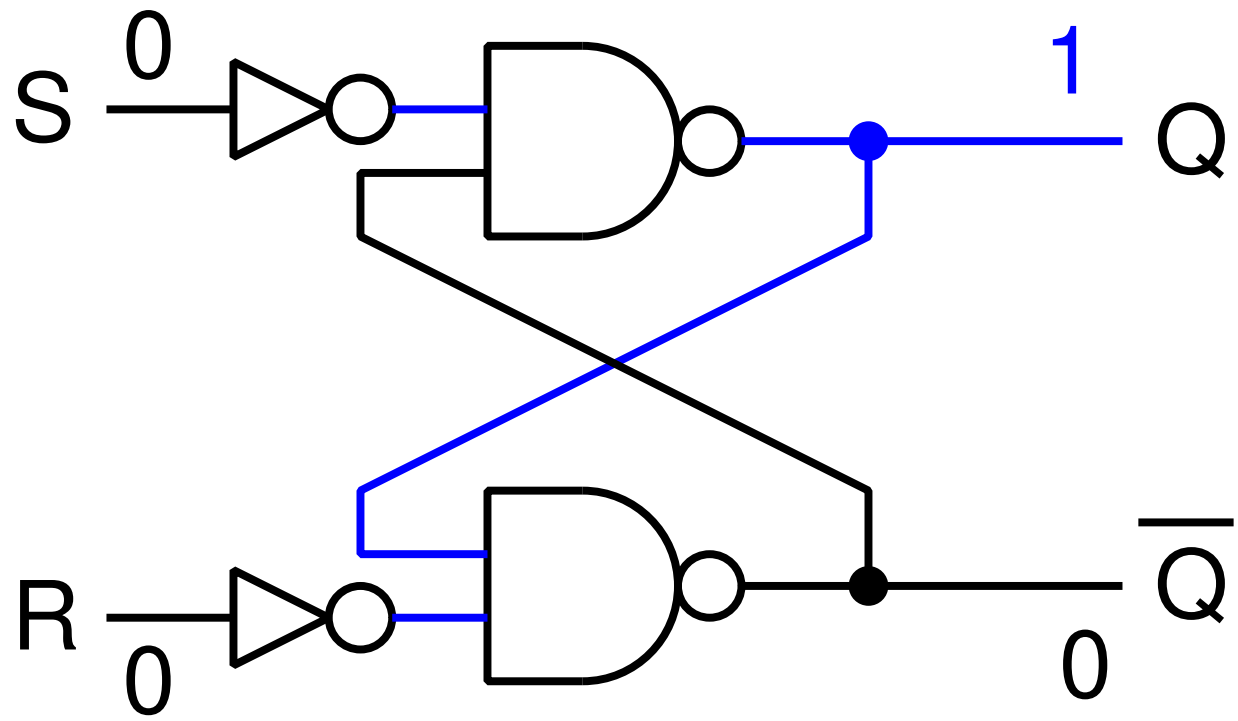
SR-フリップフロップ (Set-Reset)

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



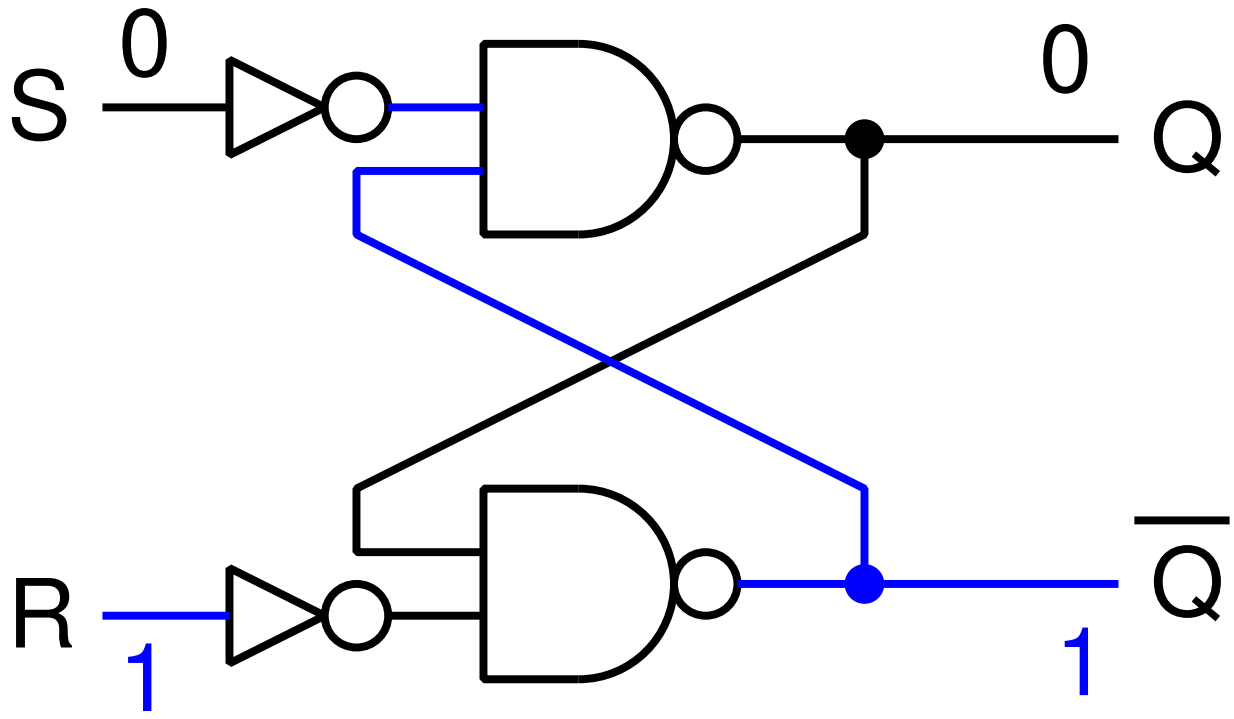
SR-フリップフロップ (Set-Reset)

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



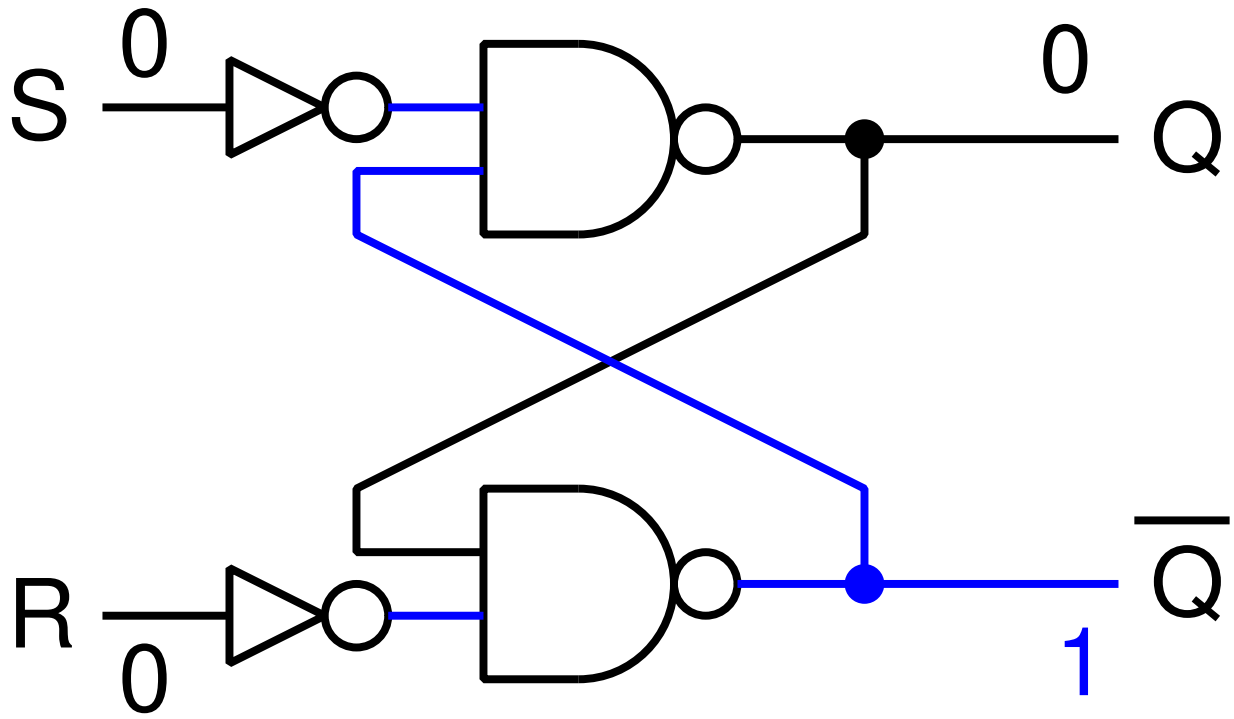
SR-フリップフロップ (Set-Reset)

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



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**

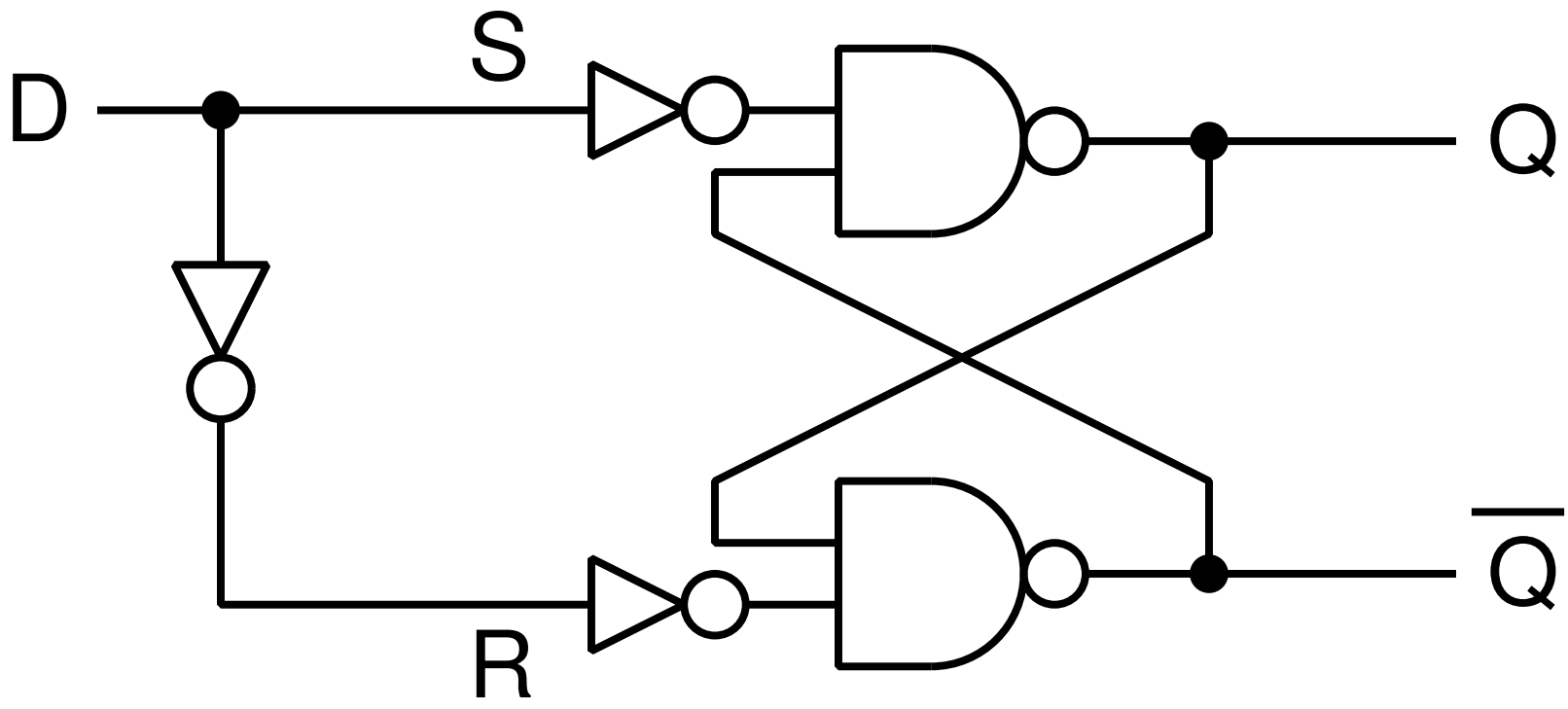
($X_f = \{(0,0), (0,1), (1,0)\}$**)**

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

入力が $(S, R) = (1, 1)$ とならない回路設計

$\longrightarrow$ **$S = \overline{R}$ となるようにしてみる**



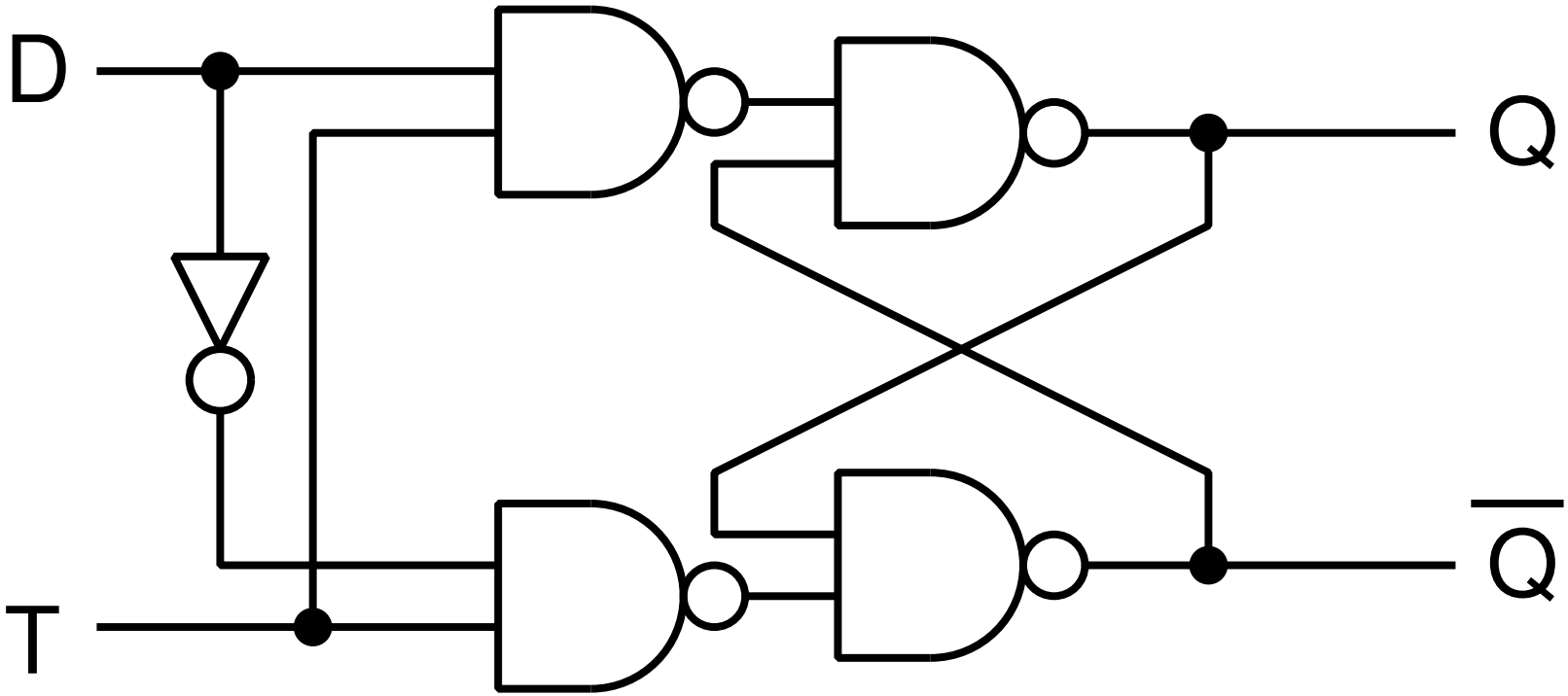
$\longrightarrow$ **入力 D を保持・出力**

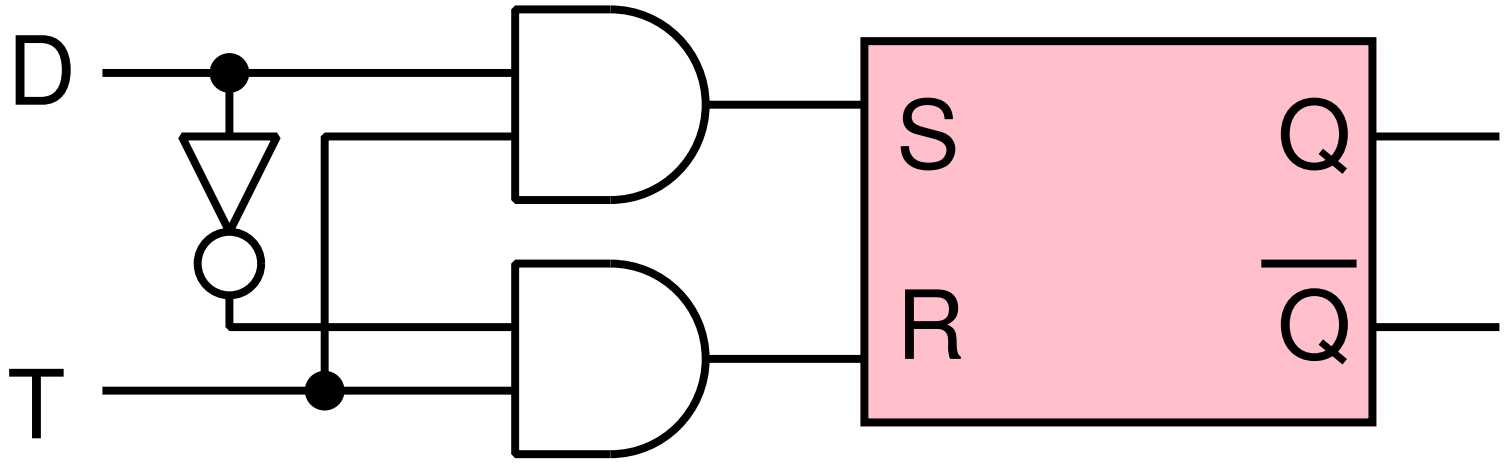
これだと D をそのまま流しているのと同じ

$\longrightarrow$ 書込スイッチを付けよう

T：スイッチ入力

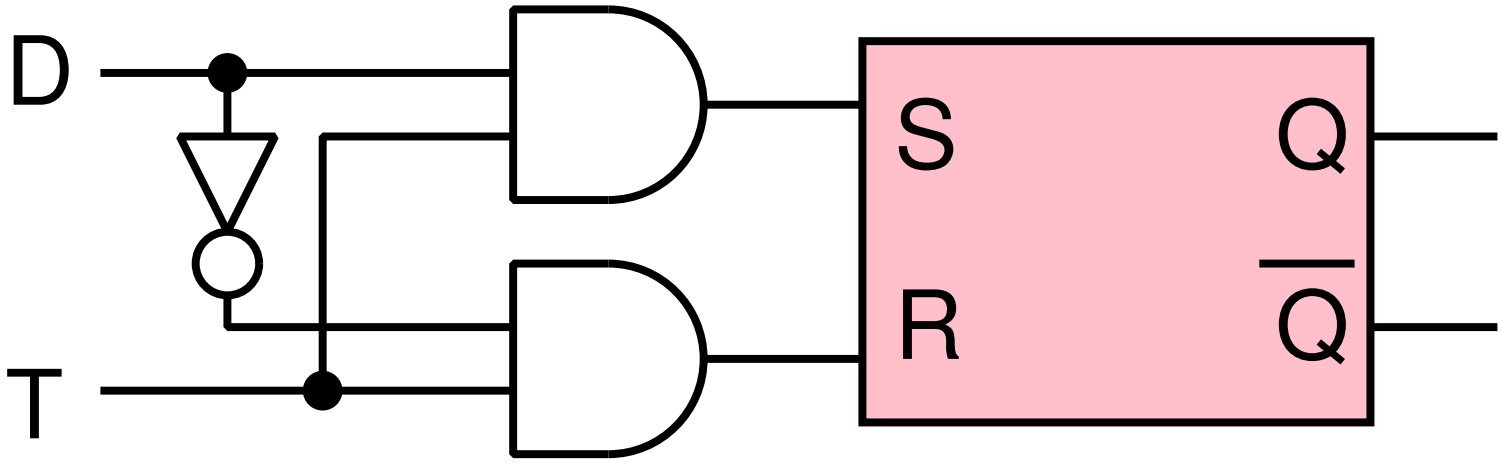
- **$T = 0$：そのまま**
- **$T = 1$：D を取り込む**





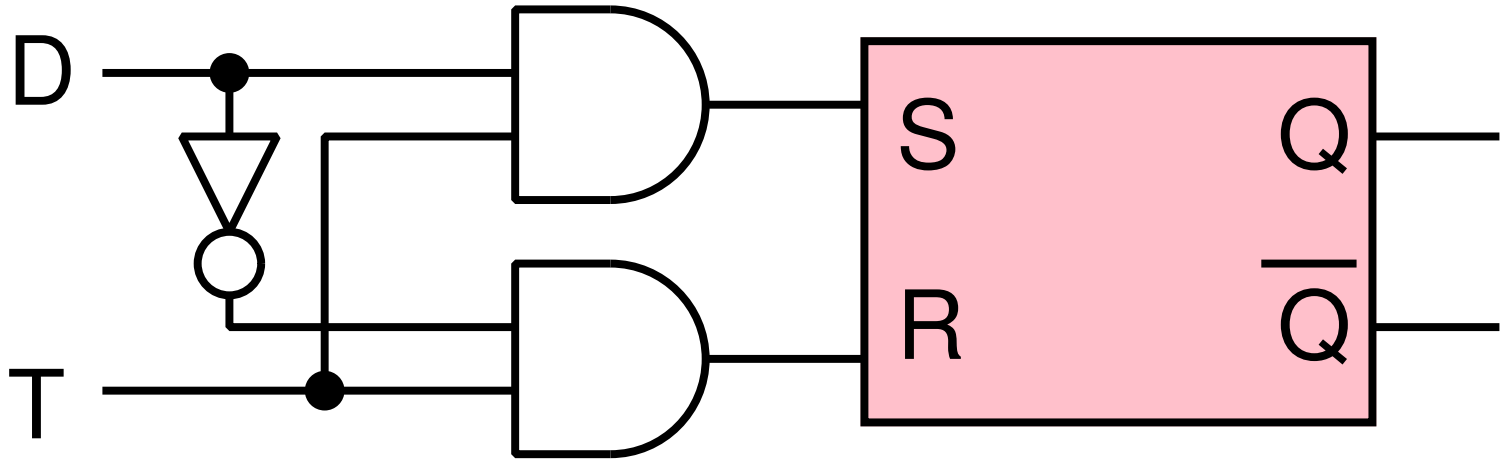
T	D	S	R	Q
0	0	0	0	Q
0	1	0	0	Q
1	0	0	1	0
1	1	1	0	1

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



- 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

$\longrightarrow$ 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



- 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

$\longrightarrow$ 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

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

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

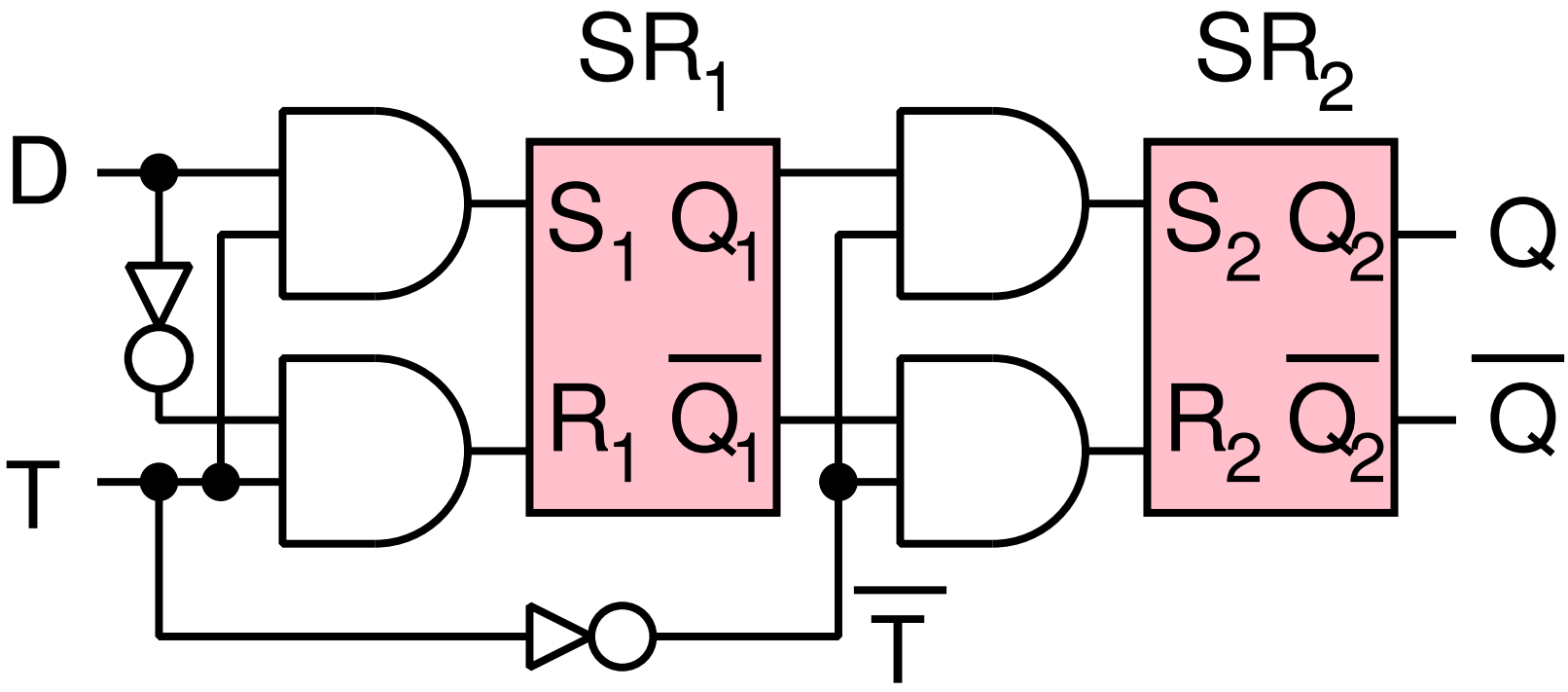
⟶ D-フリップフロップ・クロックパルス
の利用

回路の他の部分と
タイミングを合わせる必要あり

$\longrightarrow$ **出力を一旦せき止めて、**
一段階づつ計算を進める

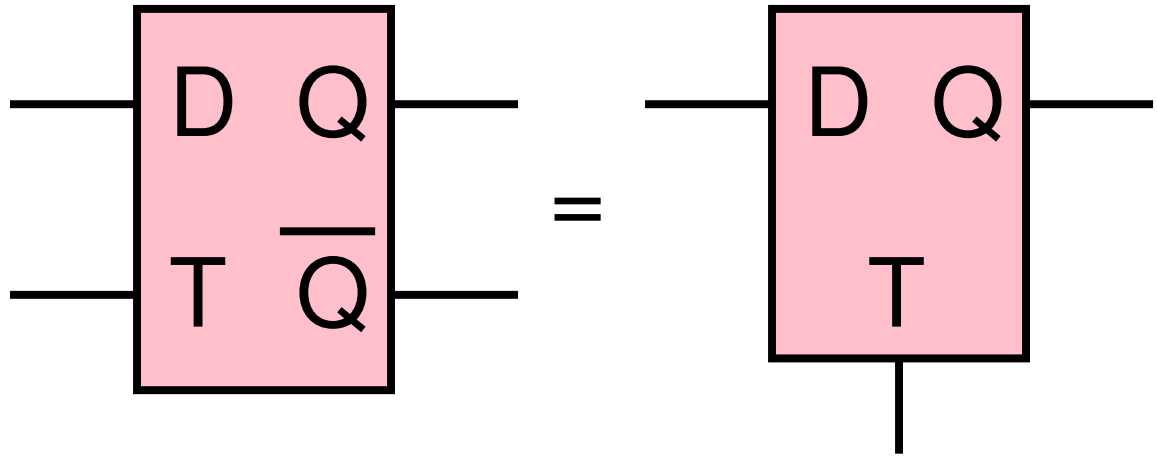
$\longrightarrow$ **D-フリップフロップ・クロックパルス**
の利用

D-フリップフロップ (Double, Delay)



- $T = 1, \overline{T} = 0 \Longrightarrow$ **SR$_1$: 取込、SR$_2$: 保持**
- $T = 0, \overline{T} = 1 \Longrightarrow$ **SR$_1$: 保持、SR$_2$: 取込**

D-フリップフロップ (Double, Delay)



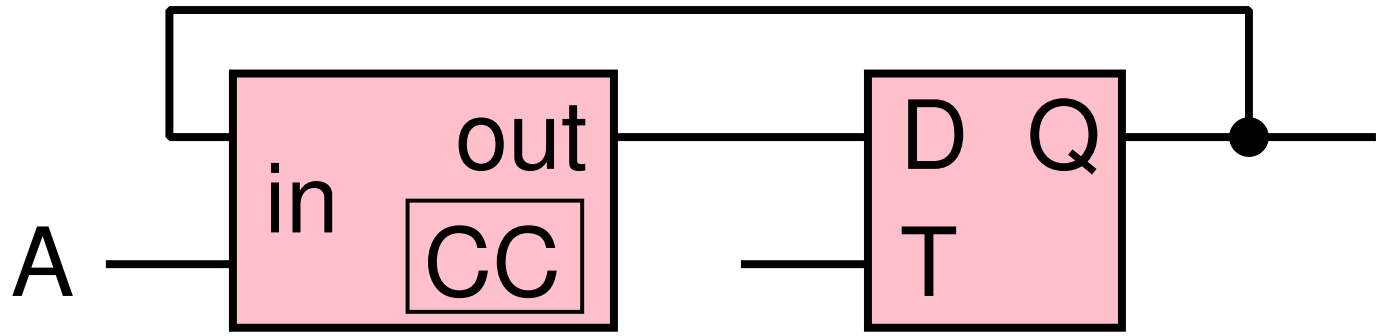
- $T = 1$: **入力 D を内部に取り込む (出力 Q は変わらない)**
- $T = 0$: **内部状態を Q に送り出す (入力 D は受付けない)**

例: (CC は適当な組合せ回路)

- $T = 1$: **入力** D **を内部に取り込む**
- $T = 0$: Q **に出力** $\longrightarrow$ CC **の入力へ**

  $\longrightarrow$ 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**

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



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、

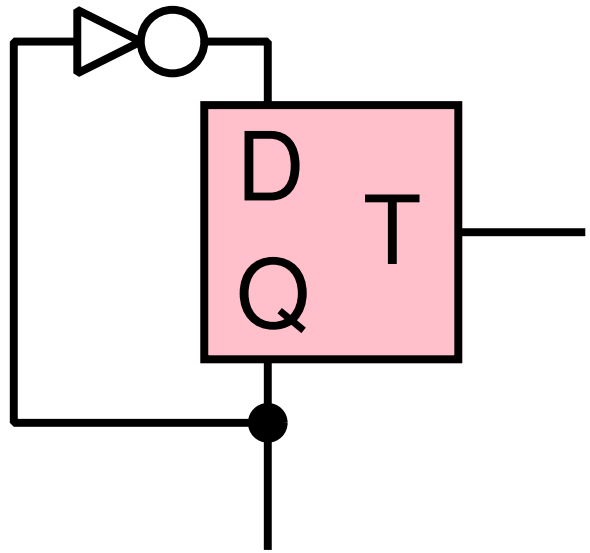
回路全体での同期を取って、計算を進める。

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

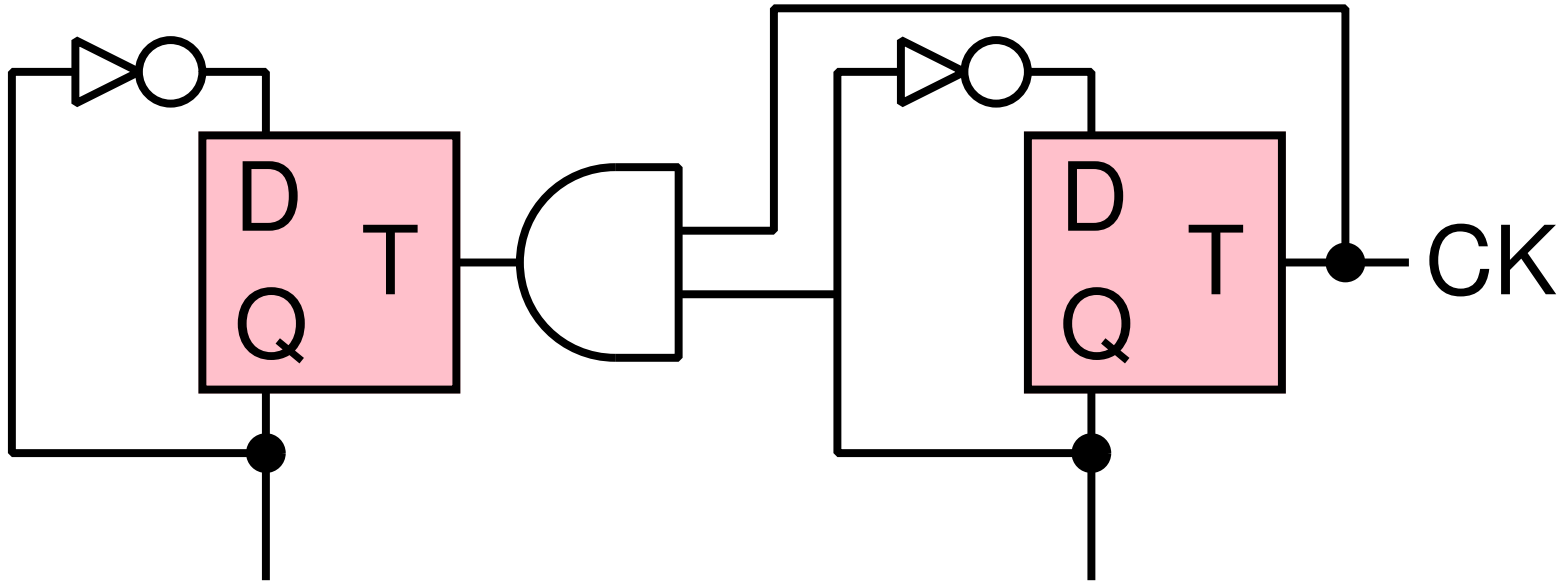
T が一周期変化する度に
内部状態が $1 \to 0 \to 1$ と変化する。



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。**



ここまでで、

計算機を構成する基本的な論理回路の

部品については紹介した。

⇓

どのような部品をどう組み合わせれば

計算機の機能を実現できるか

計算機が実現すべきこと

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