論理回路の働き:

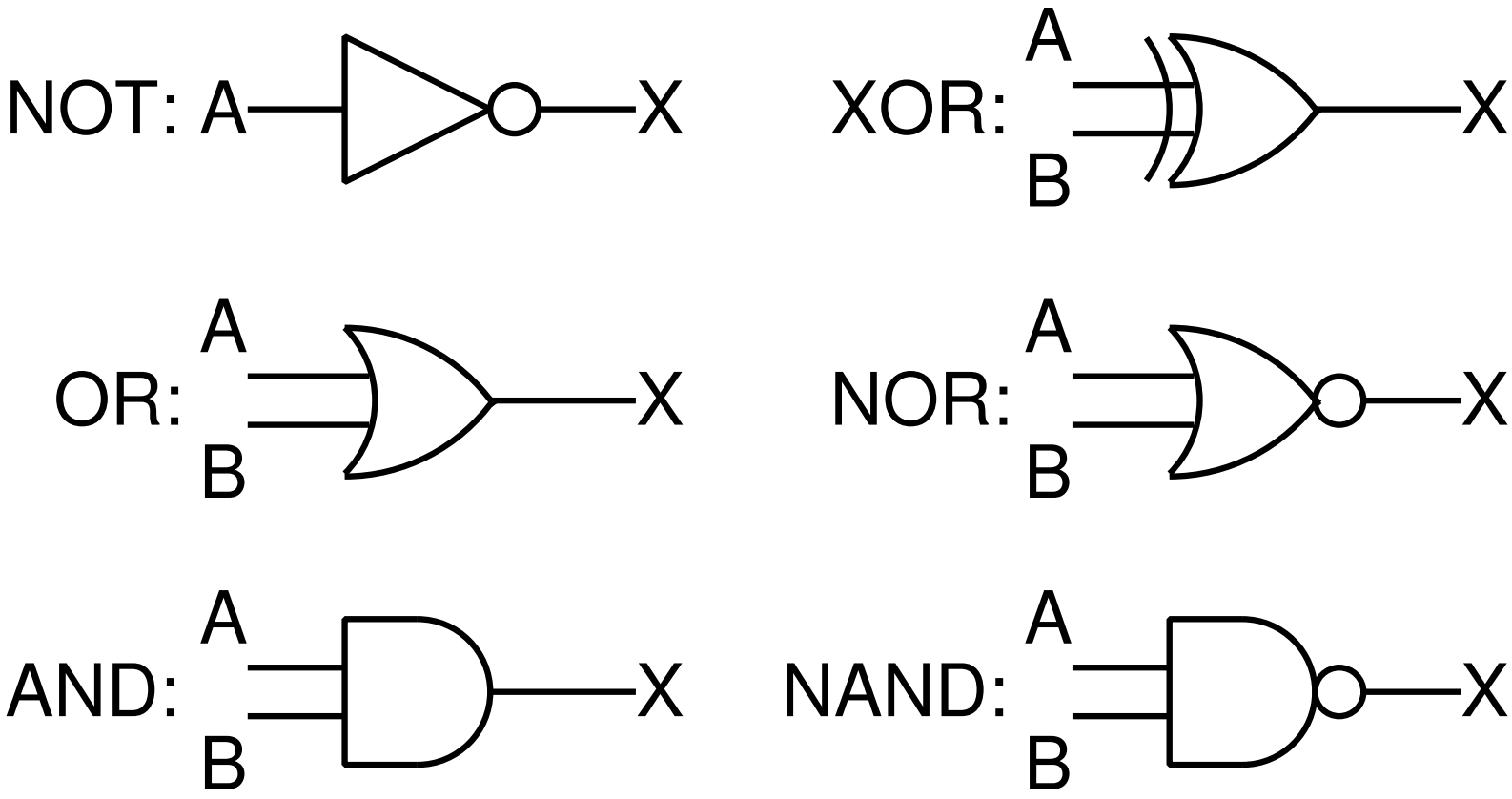
- データ (bit 情報) の保持
- 基本的な演算

論理回路の種類:

- 組合せ回路:
  入力 (の組) によって出力が決まる
- 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

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**：
  $\neg(A \wedge B) = \neg A \vee \neg B$


NOT: A
X
XOR:
A
B
X
OR:
A
B
X
NOR:
A
B
X
AND:
A
B
X
NAND:
A
B
X

真理値表

NOT	X
A 0	1
A 1	0

OR	B 0	B 1
A 0	0	1
A 1	1	1

AND	B 0	B 1
A 0	0	0
A 1	0	1

XOR	B 0	B 1
A 0	0	1
A 1	1	0

NOR	B 0	B 1
A 0	1	0
A 1	0	0

NAND	B 0	B 1
A 0	1	1
A 1	1	0

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

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)

双対的に、次の形でも書ける。

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

(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)

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

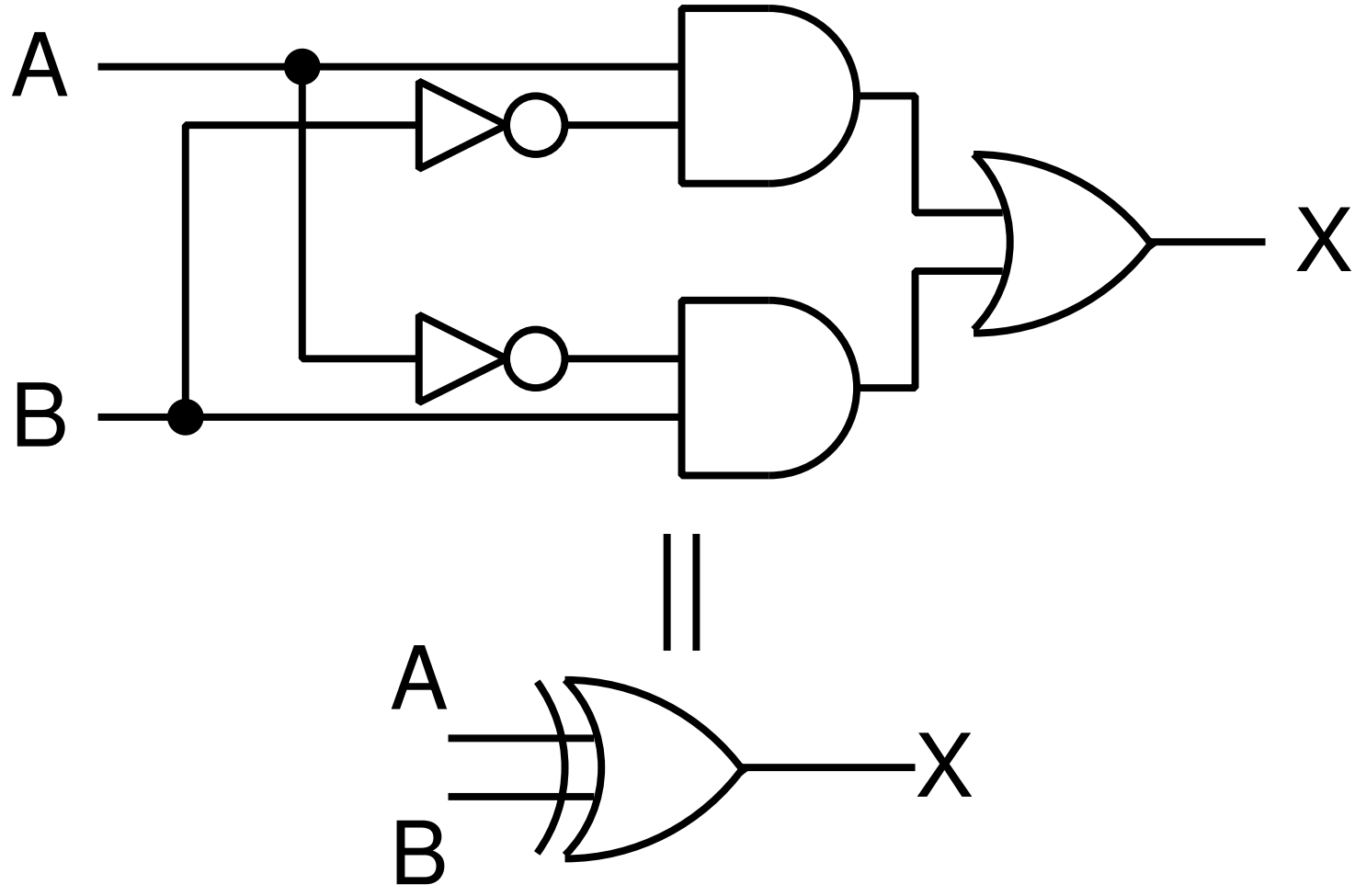
- NOT **があれば** OR, AND **は片方で良い**

- OR, AND **だけでは**
  全ての Boole 関数は実現できない

- **一種類の論理素子** NOR **または** NAND **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。**

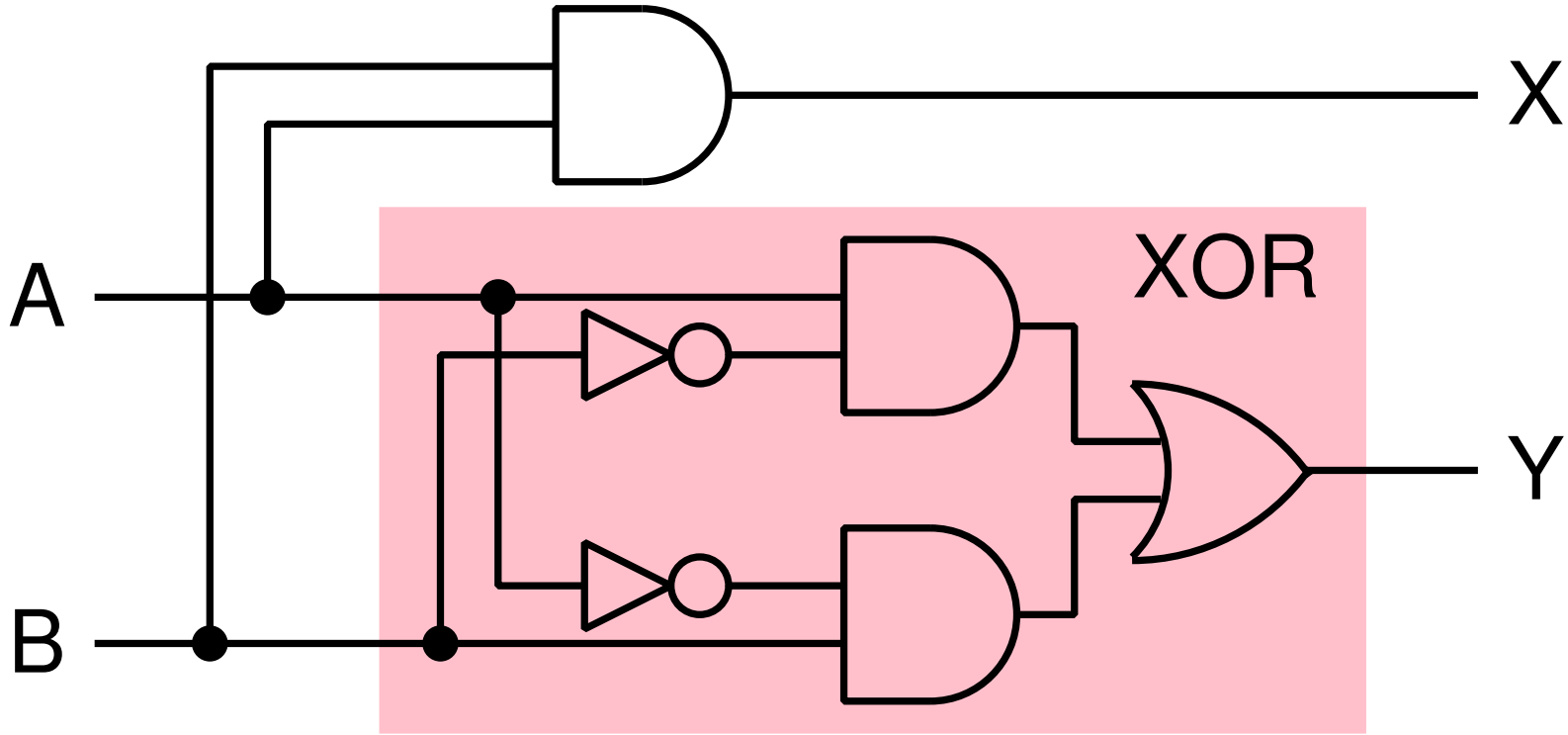
例: XOR



例: 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

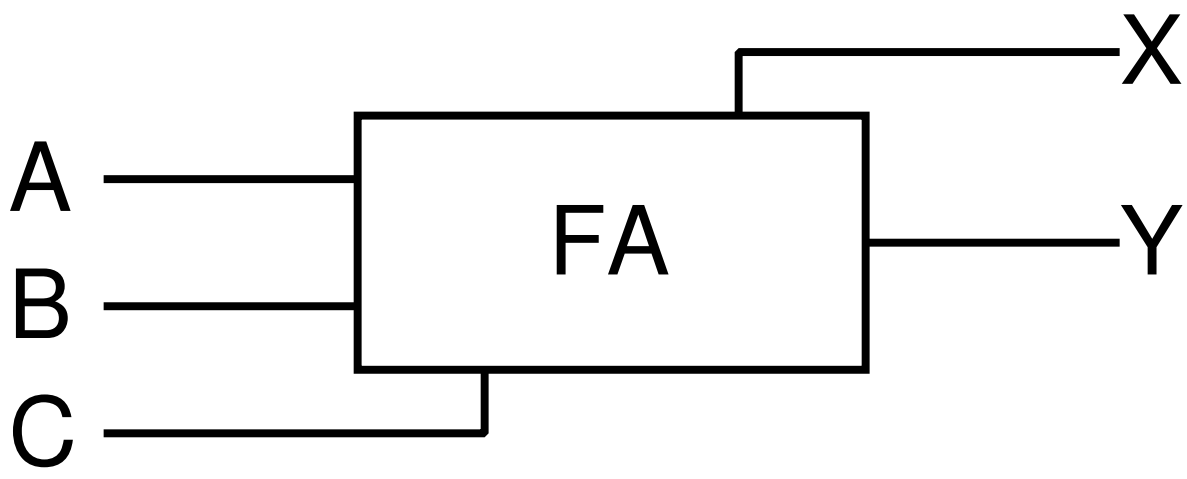


問題:

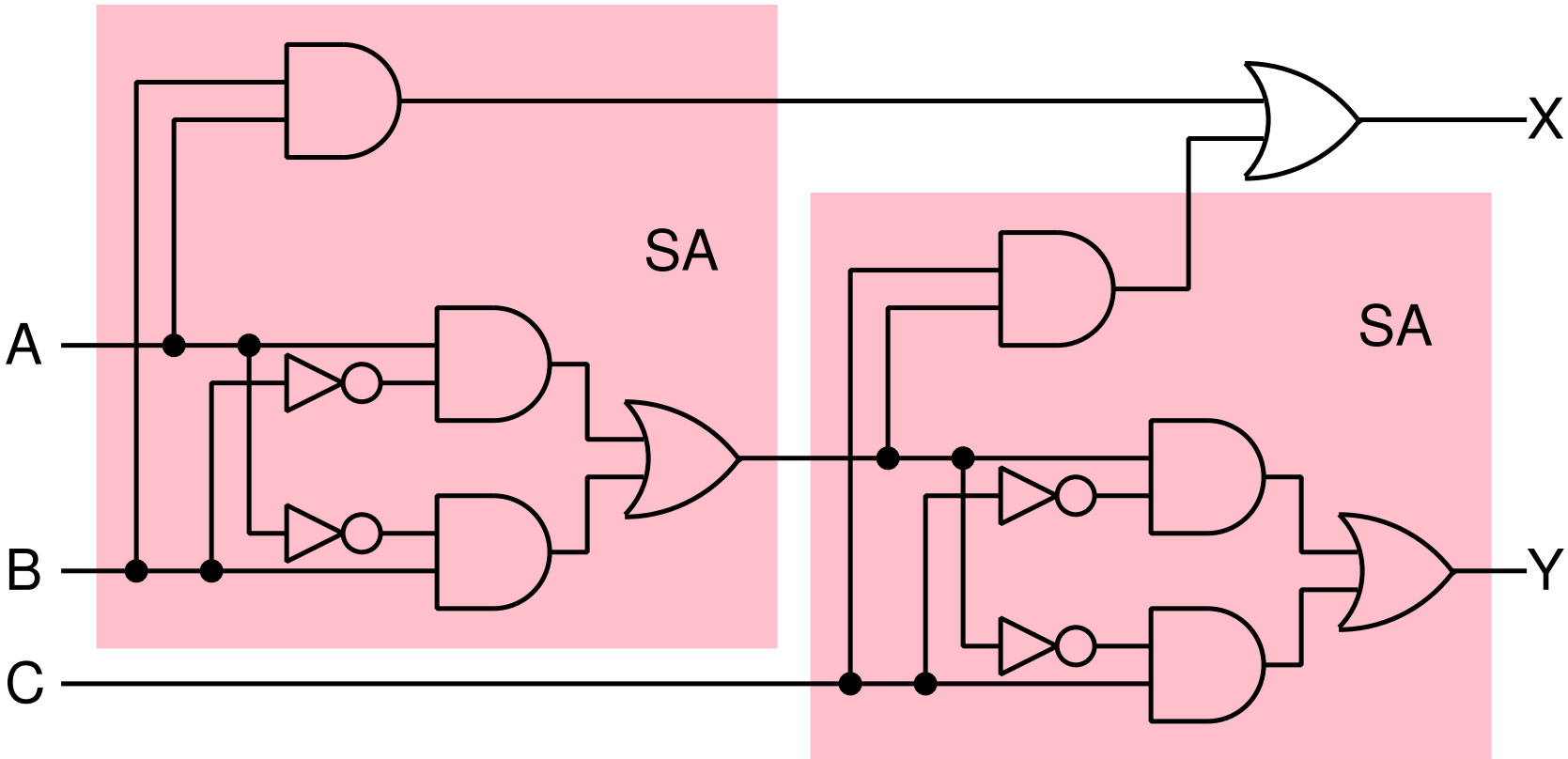
全加算器 (full adder,FA) を、
NOT, OR, AND **を用いて構成せよ。**

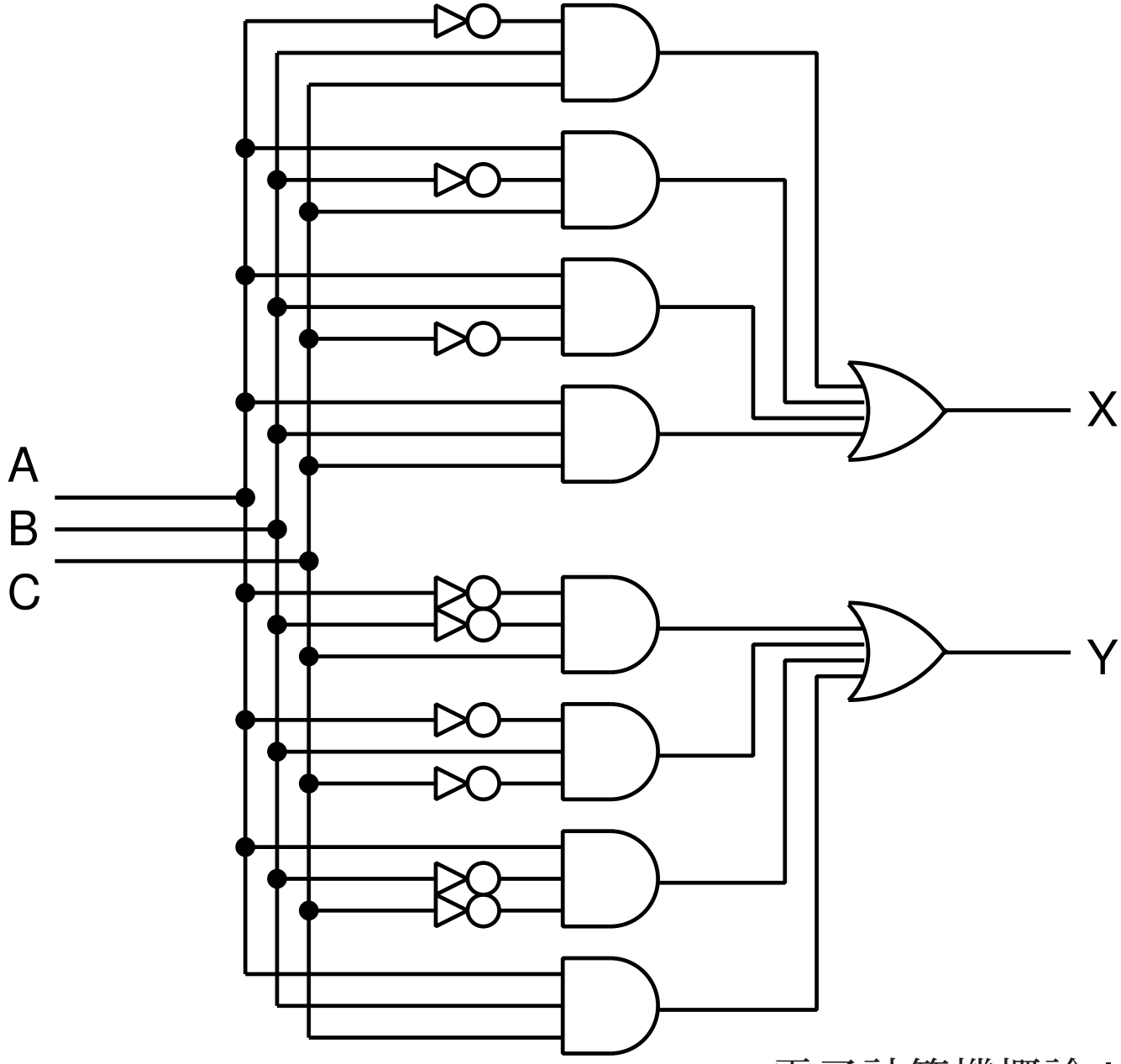
入力: A,B:各桁の値,C:下からの繰上がり

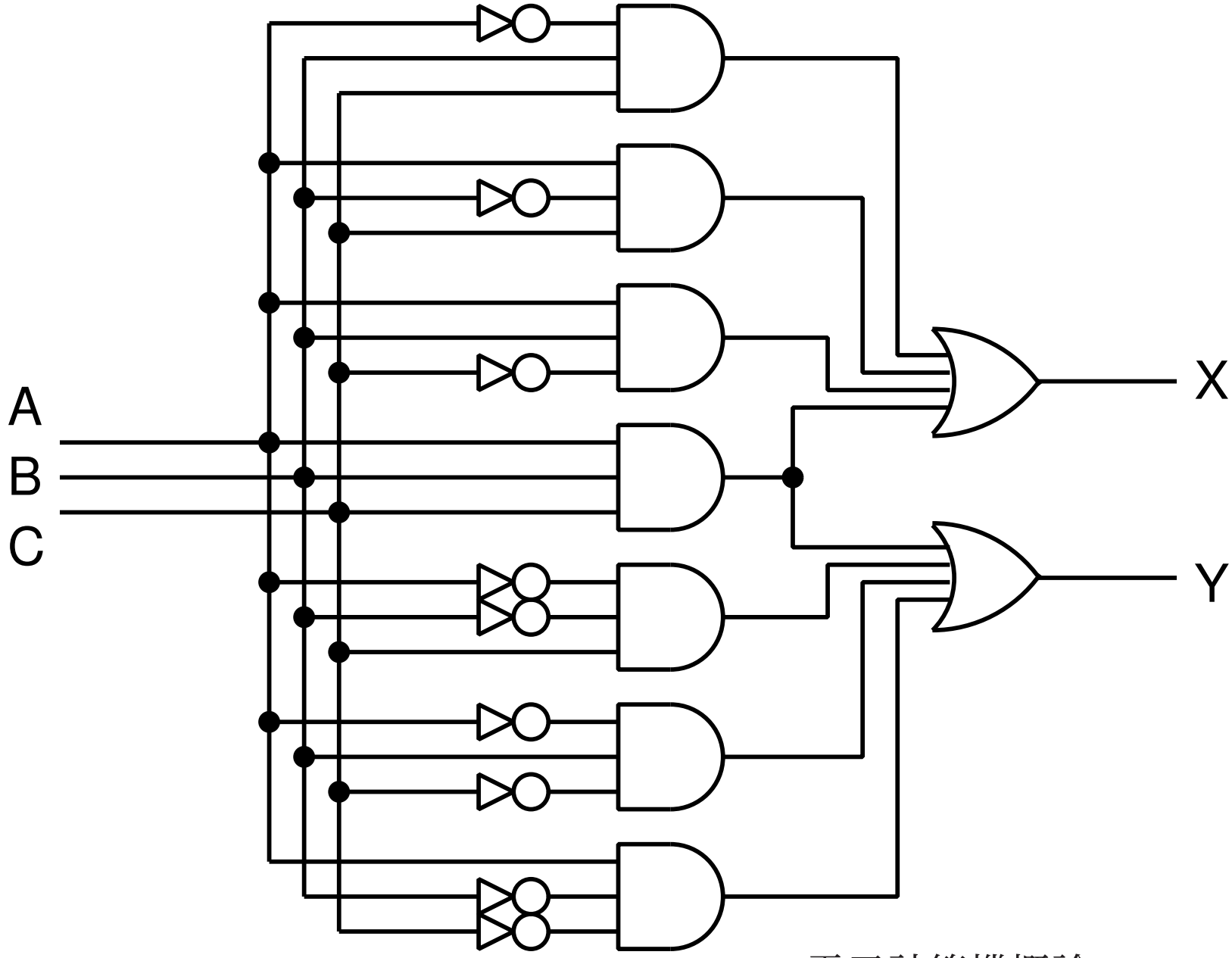
出力: X: 上への繰上がり, Y: 当桁の値



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**
- **基本的な演算**

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)

データ (bit 情報) の保持をするのは ？

$\longrightarrow$ 順序回路

(再掲) 論理回路:

- **データ (bit 情報) の保持**
- **基本的な演算**

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)

データ (bit 情報) の保持をするのは ?

$\longrightarrow$ **順序回路**

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: $Q \subset \{0,1\}^B$

n **入力** m **出力の順序回路は Boole 関数**

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

(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: $Q \subset \{0,1\}^B$

n **入力** m **出力の順序回路は Boole 関数**

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

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

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

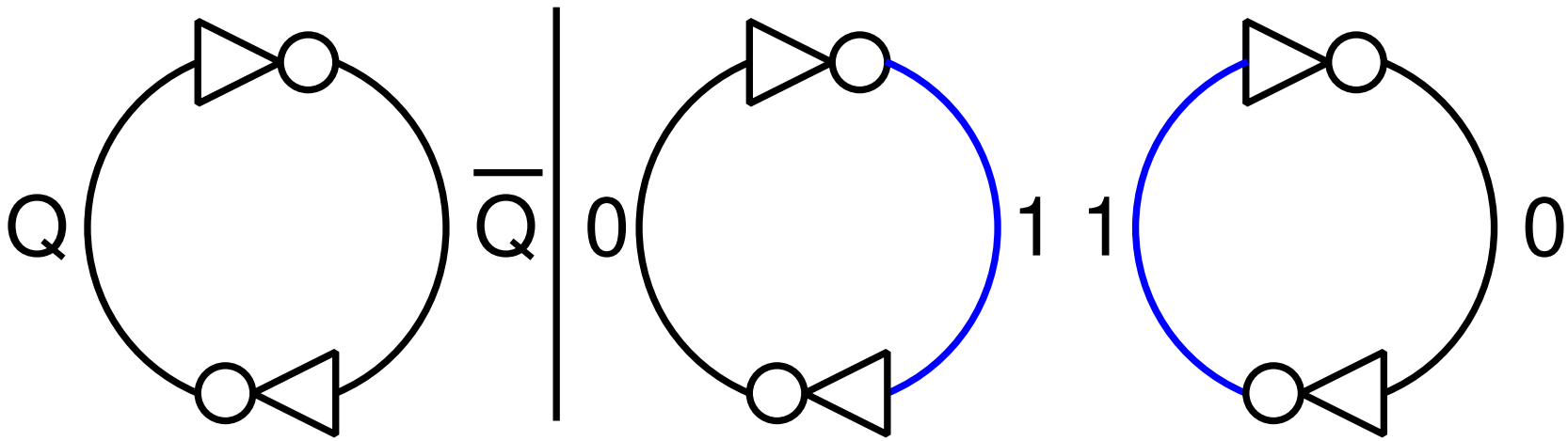
内部状態を計算機内に如何に保持するか

- 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

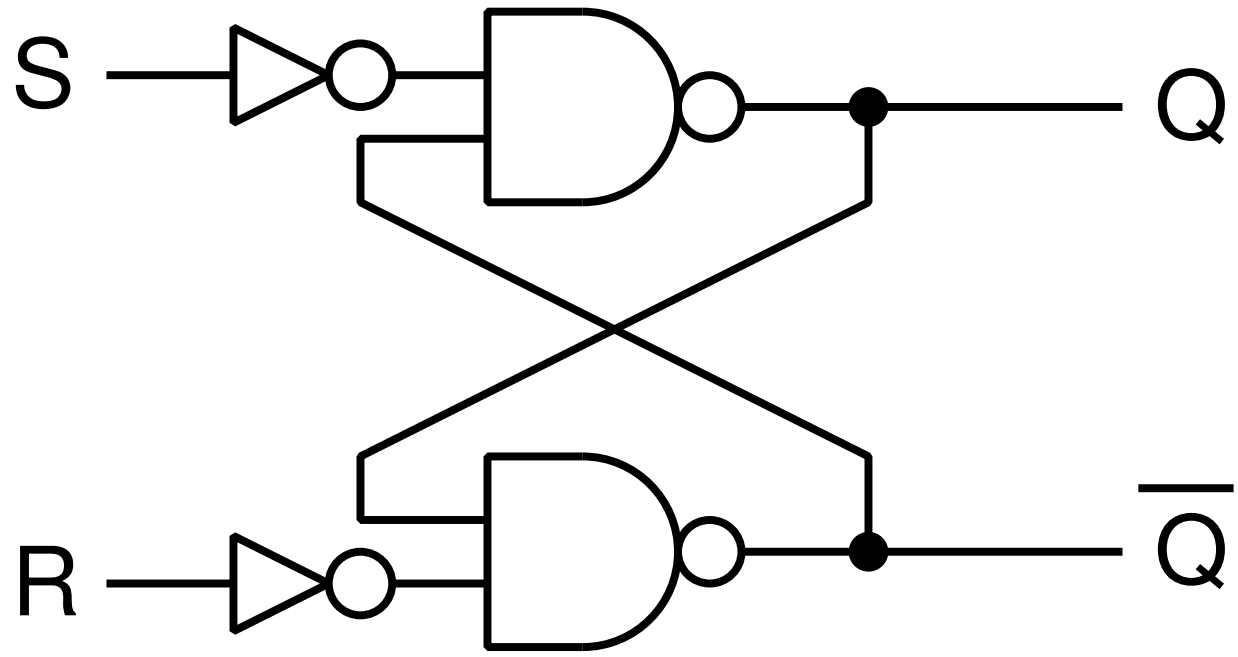
フリップフロップ (flip-flop)

原理図:

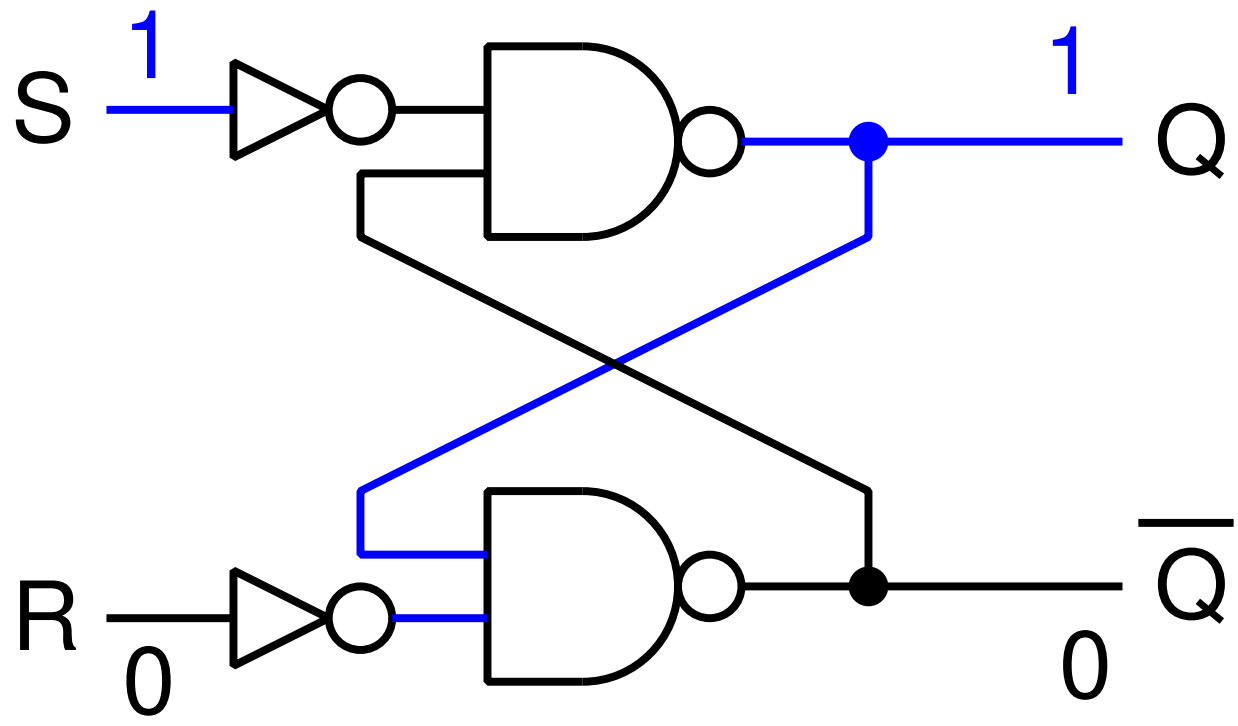


これに入出力端子を付ける

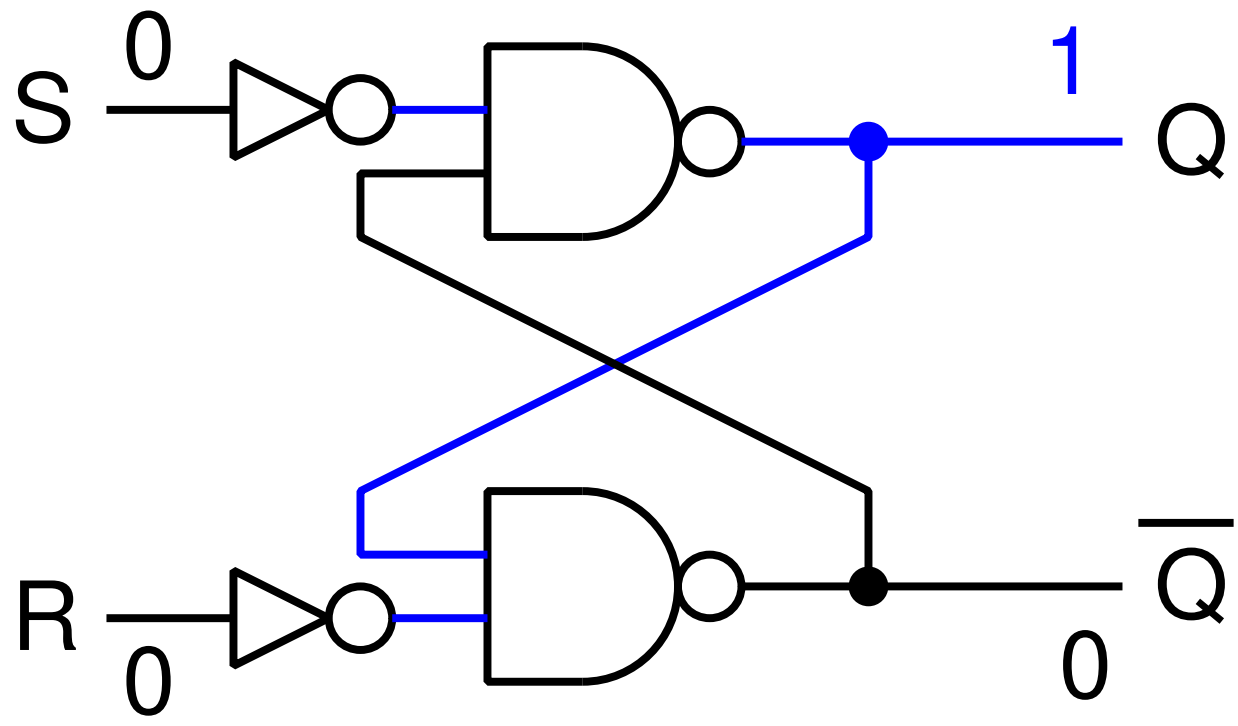
SR-フリップフロップ (Set-Reset)



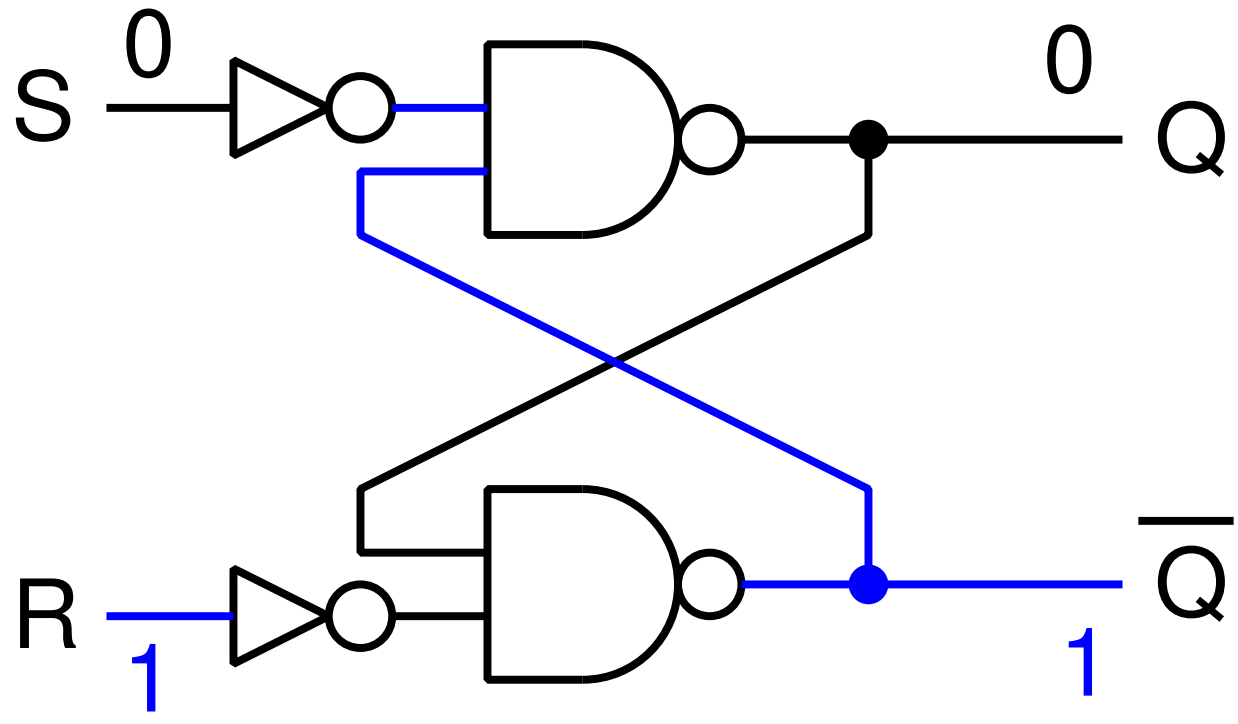
$$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$$



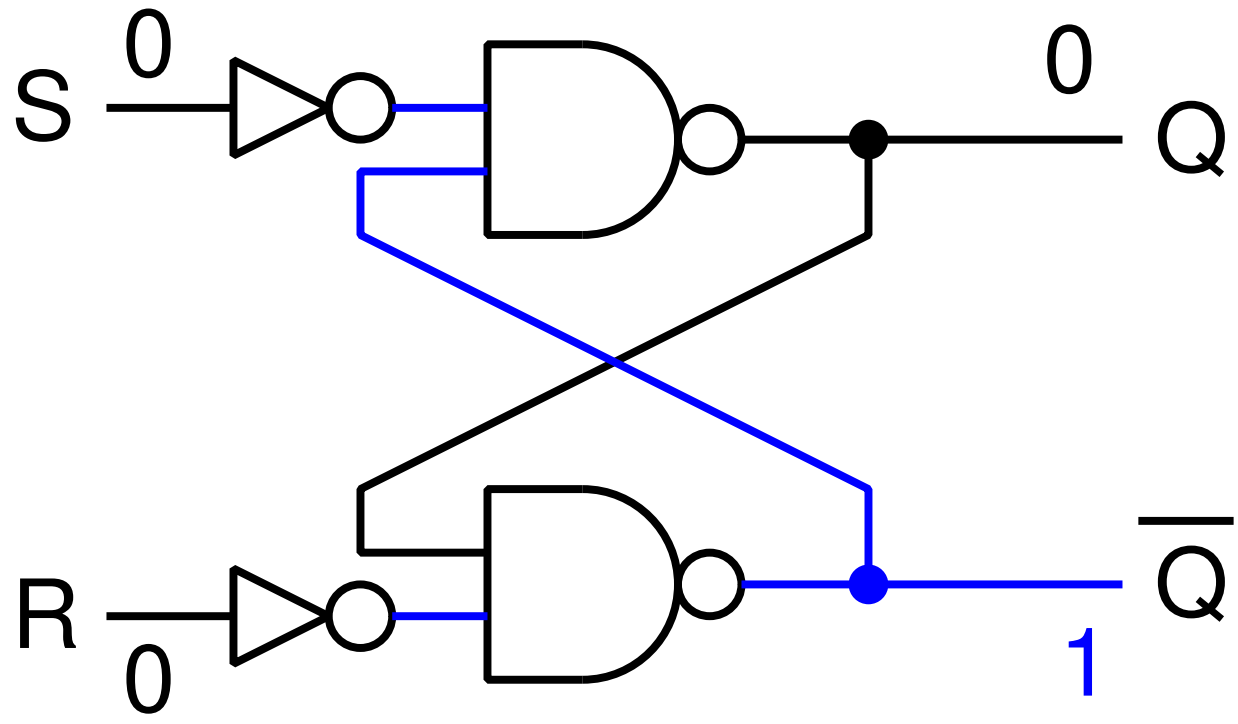
$$(S, R) = (1, 0) \longrightarrow (0, 0) \longrightarrow (0, 1) \longrightarrow (0, 0)$$



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



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



$(S, R) = (1, 1)$ **は禁止入力**

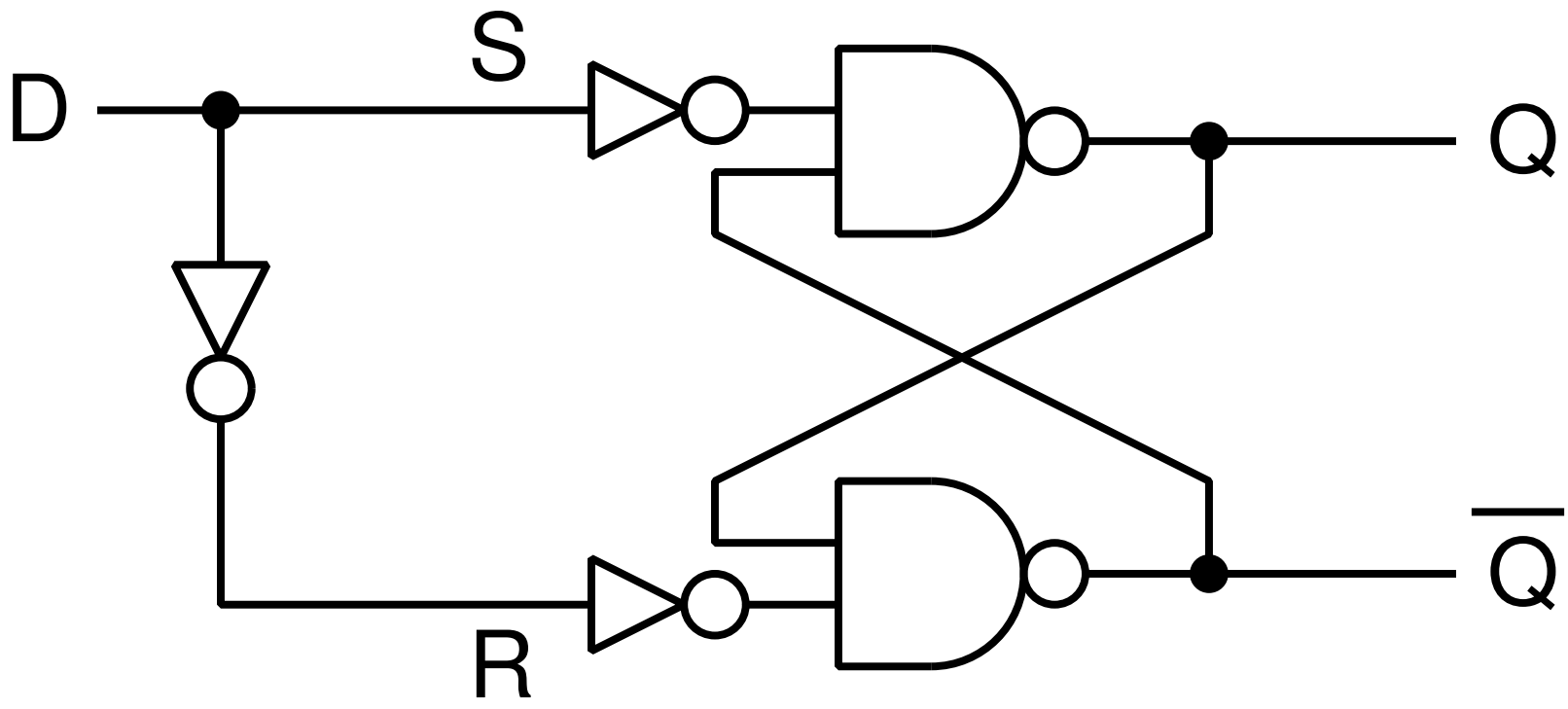
($X_f = \{(0,0), (0,1), (1,0)\}$**)**

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

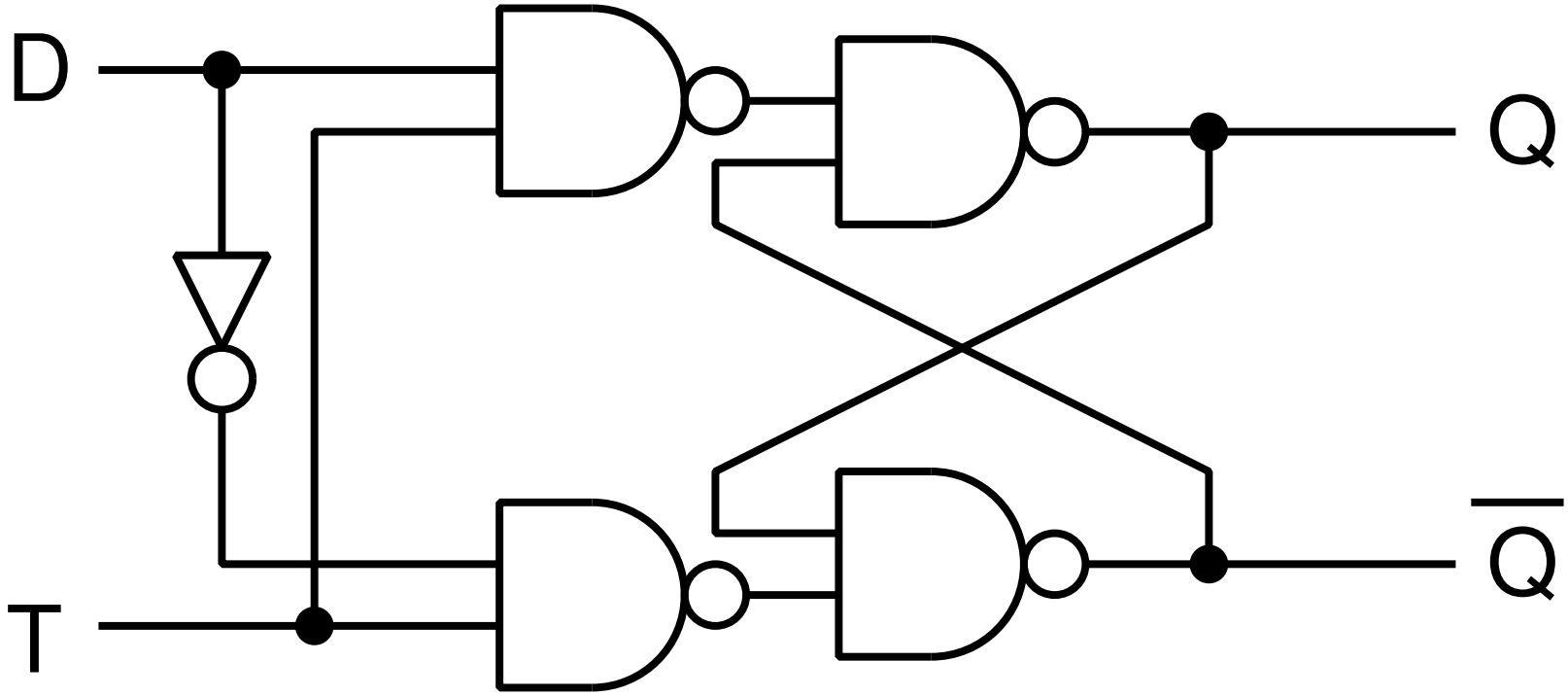
$\longrightarrow$ **$S = \overline{R}$ となるようにしてみる**

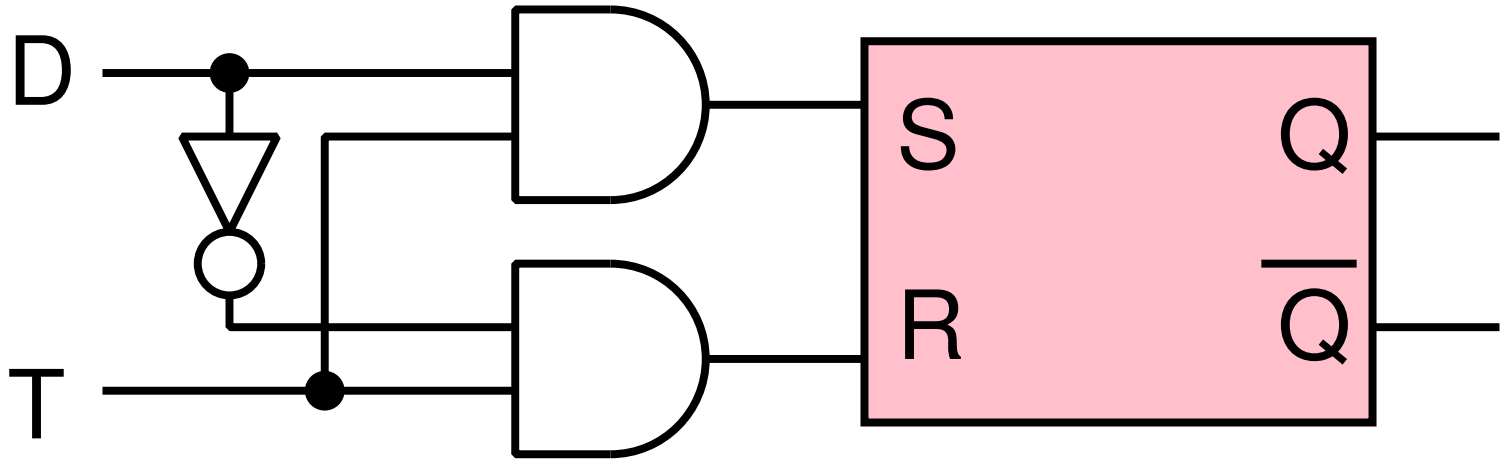


$\longrightarrow$ **入力 D を保持・出力**

T：**スイッチ入力**

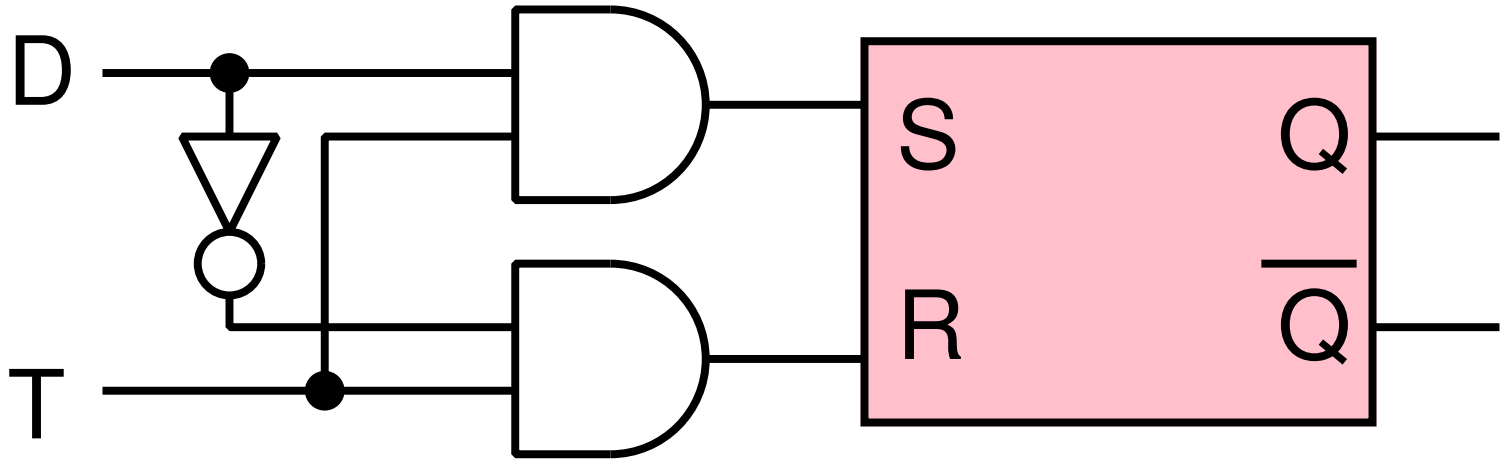
- $T = 0$：**そのまま**
- $T = 1$：D **を取り込む**





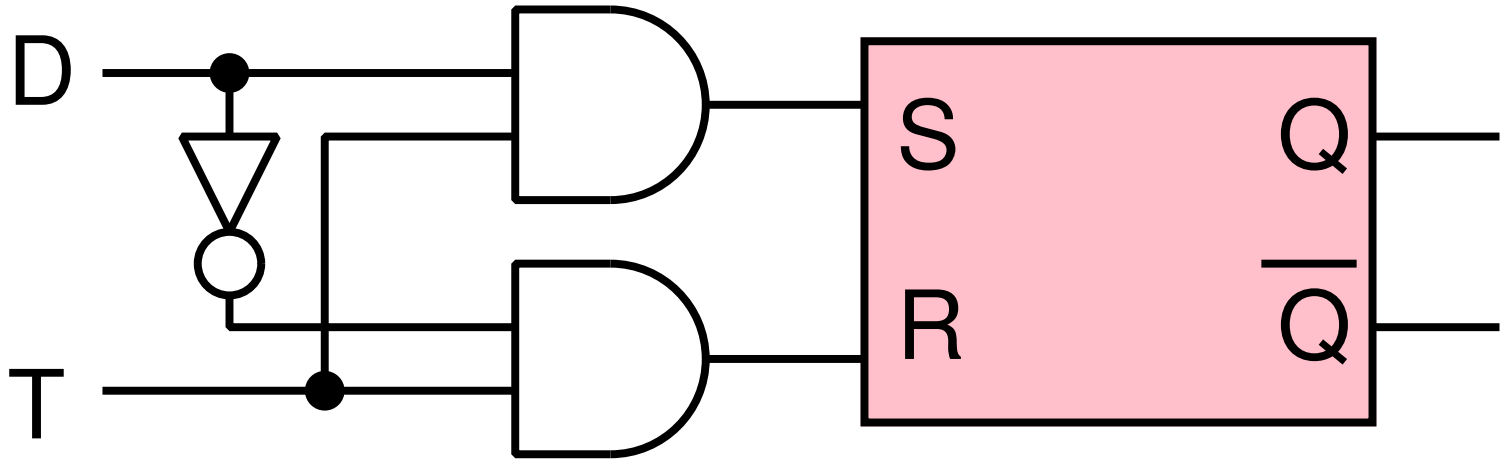
T	D	S	R	Q
0	0	0	0	Q
0	1	0	0	Q
1	0	0	1	0
1	1	1	0	1

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