

KYUSHU UNIVERSITY · SYS-103

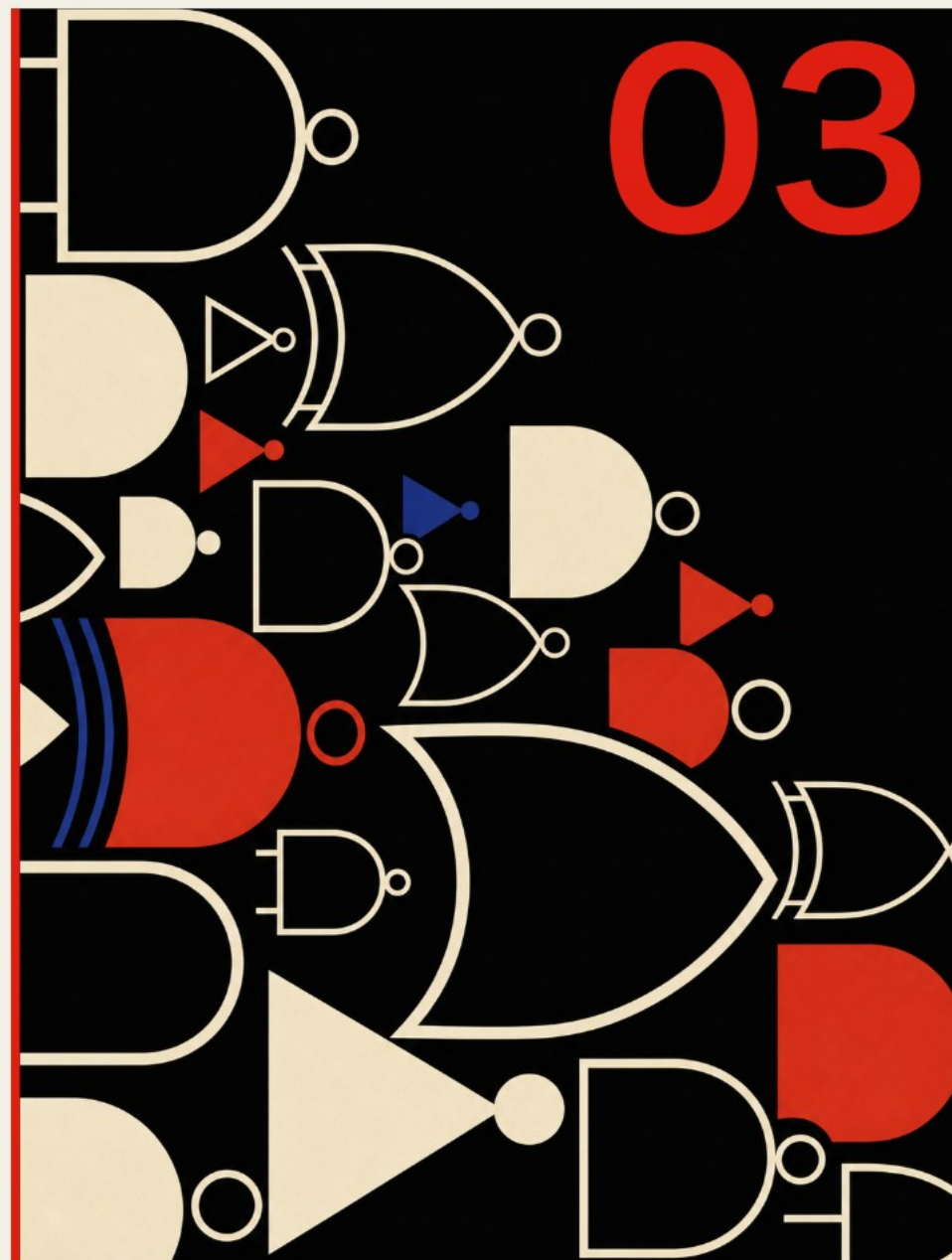
デジタル回路の 基礎 (1)

0 と 1 の規則を、計算する回路へ

2026 年 9 月

Olivia Chen Toyama · C2LAB

LECTURE 3



前回は、0 と 1 の「規則」を書いた。 今日は、その規則を回路にする

RULE

紙の上の加算

$$1 + 1 = 10$$

PHYSICAL STATE

電気の ON / OFF



THIS LECTURE

規則を実行する回路



前回の最後の問い

この「？」の中を、どう作る？

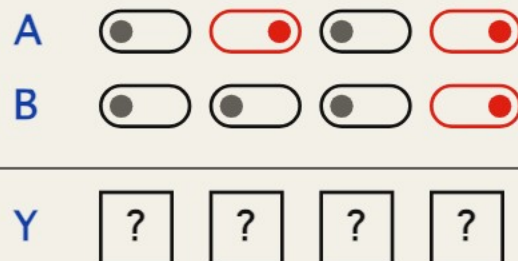
0101 + 0001 を、 スイッチだけで計算できるか

答えは計算できる

$$\begin{array}{r} 0101 \\ + 0001 \\ \hline 0110 \end{array}$$



では、答えを生む仕組みは？



答えではなく、**答えを生む仕組み**を作ります。

今日のゴール：0 / 1 のスイッチから、0110 が出るまでを説明する

01 / READ



電気の状態を読む

OFF / ON を
0 / 1 として扱う

02 / BUILD

$$1 + 1 \rightarrow 10$$

一桁の加算を作る

四つの入力パターンを
一つずつ回路にする

03 / CONNECT

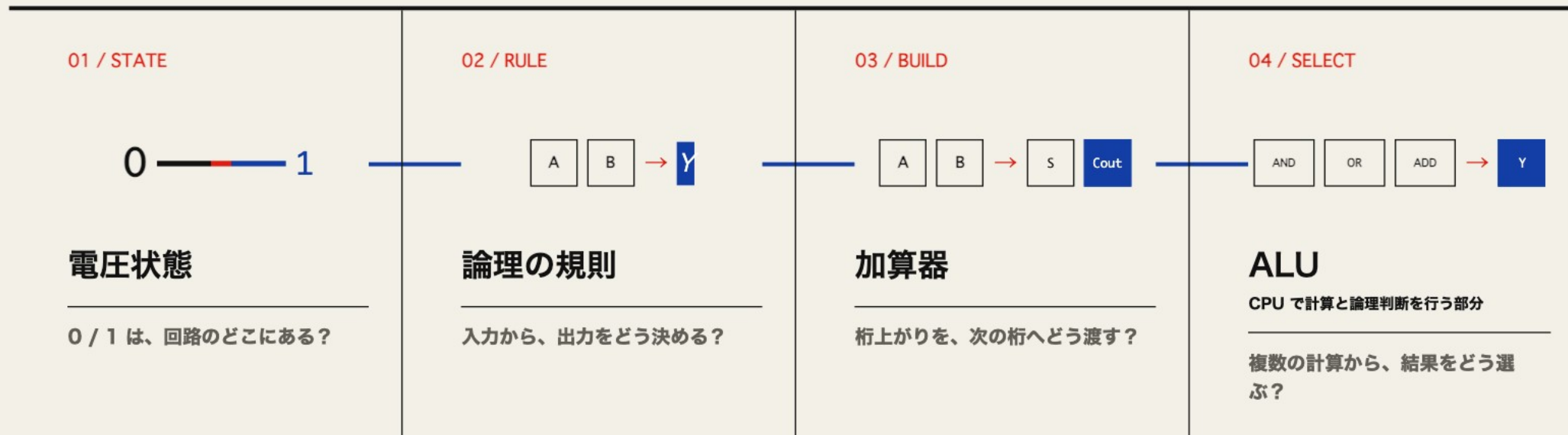
$$\begin{array}{r} 0101 \\ + 0001 \\ \hline \end{array} \rightarrow 0110$$

四桁をつなぐ

下位桁から上位桁へ
桁上がりを渡す

名前を先に暗記せず、**信号の変化を一步步追う。**

電圧状態から、ALU まで



最初の問い

0 と 1 は、実際の回路では何だろう？

紙の上は 0 / 1。電線の中は連続した電圧。

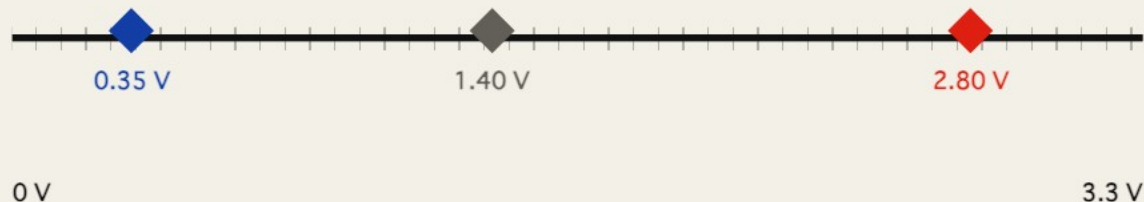
人が使う記号

0 / 1

答えは 0 か 1 のどちらか



実際の電線を測ると



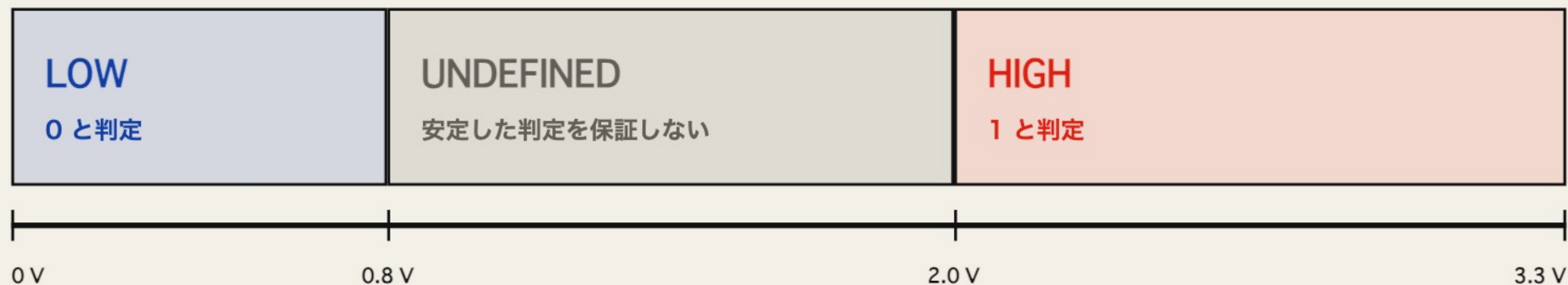
0 V から 3.3 V まで、途中の値も測れる

では、1.40 V は 0? それとも 1?

電圧の範囲を **LOW** と **HIGH** に分け、 0 と 1 に対応させる

V_{IN}

3.3 V ロジックの一例



電圧そのものではなく、**どの範囲に入ったか**を読む。

判定境界から離れた電圧なら、 多少のノイズにも耐えられる



送信値から判定境界まで：

$$0.80 - 0.35 = 0.45 \text{ V}$$

$$2.80 - 2.00 = 0.80 \text{ V}$$

次に必要なのは、電圧を判定する電気のスイッチ。

接点から真空へ、そしてチップの中へ

[INTERACTIVE TIMELINE](#) [詳しい年表を見る](#)



STEP 5 / 5 ・ 集積回路の内部

現在

IC 内の MOS

一つのチップに数百億個

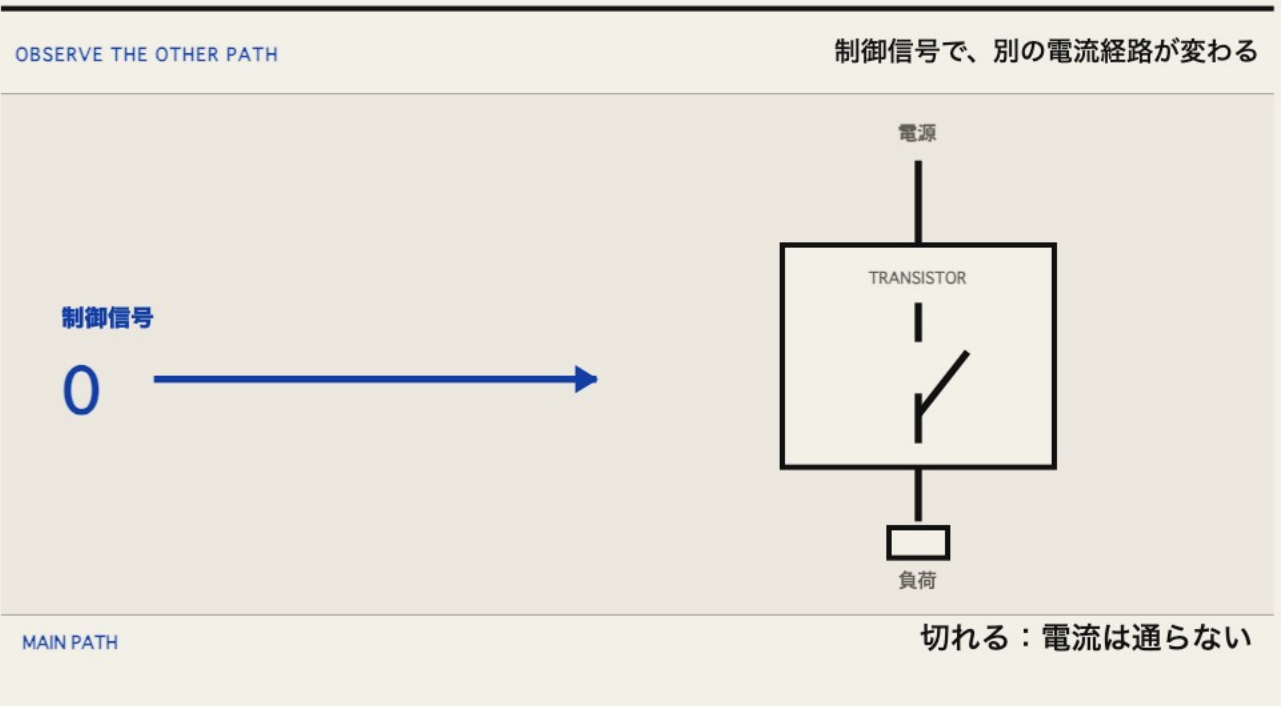
一個は肉眼では見えない。
配線と層を重ねて機能を作る。

SIZE	INTEGRATION
nm 級	数百億個 / chip

機械接点 → 真空 → 半導体 → IC 役割は同じ：電流を通す／止める → 次は、一つのトランジスタへ

[戻る](#)[1 リレー](#)[2 真空管](#)[3 トランジスタ](#)[4 MOSFET](#)[5 IC 内の MOS](#)[次へ](#)

トランジスタは、電気で動くスイッチ



TRY / CONTROL INPUT

0 / LOW

1 / HIGH

SWITCH

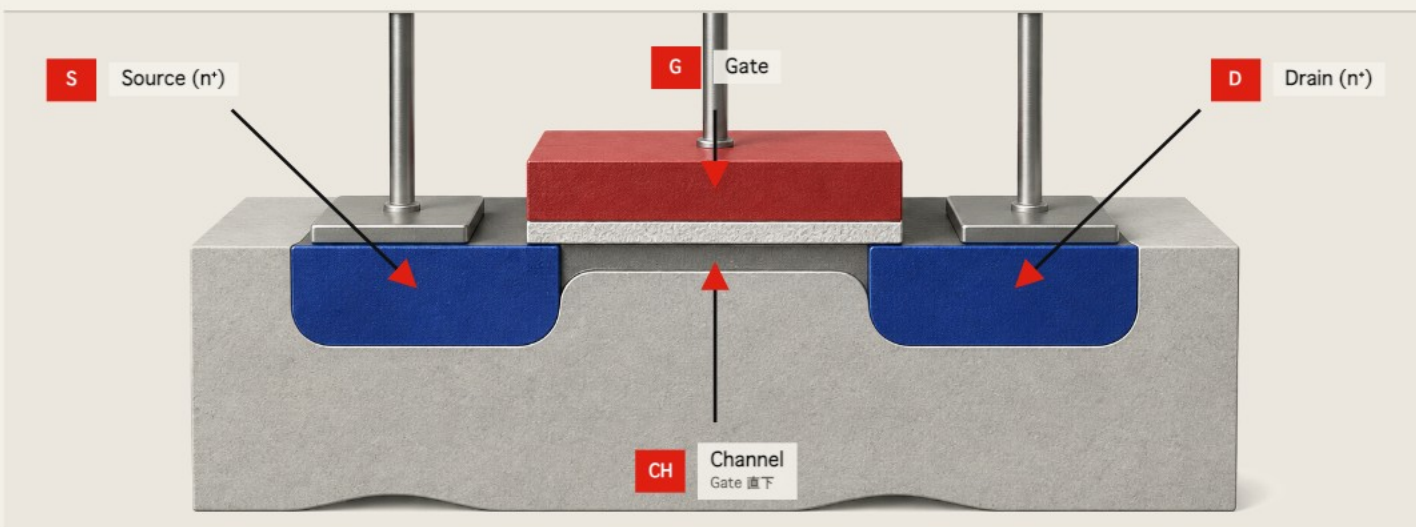
OFF

- 1 制御信号を切り替える
- 2 主回路がつながる
- 3 負荷へ電流が流れる

小さな制御 → 別の経路を ON / OFF

ゲート電圧が、ソースとドレインの間のチャンネルを制御する

01 / PHYSICAL STRUCTURE

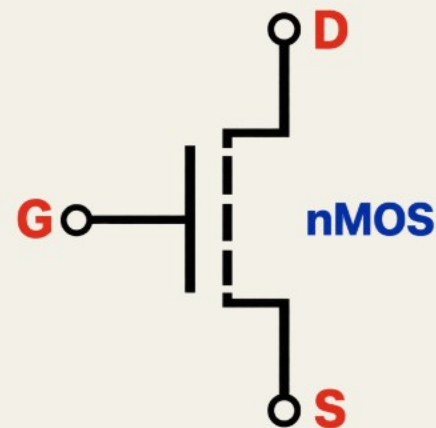


Gate = HIGH の電界で

Source と Drain の間に Channel ができる

02 / CIRCUIT SYMBOL

3 端子の簡略記号

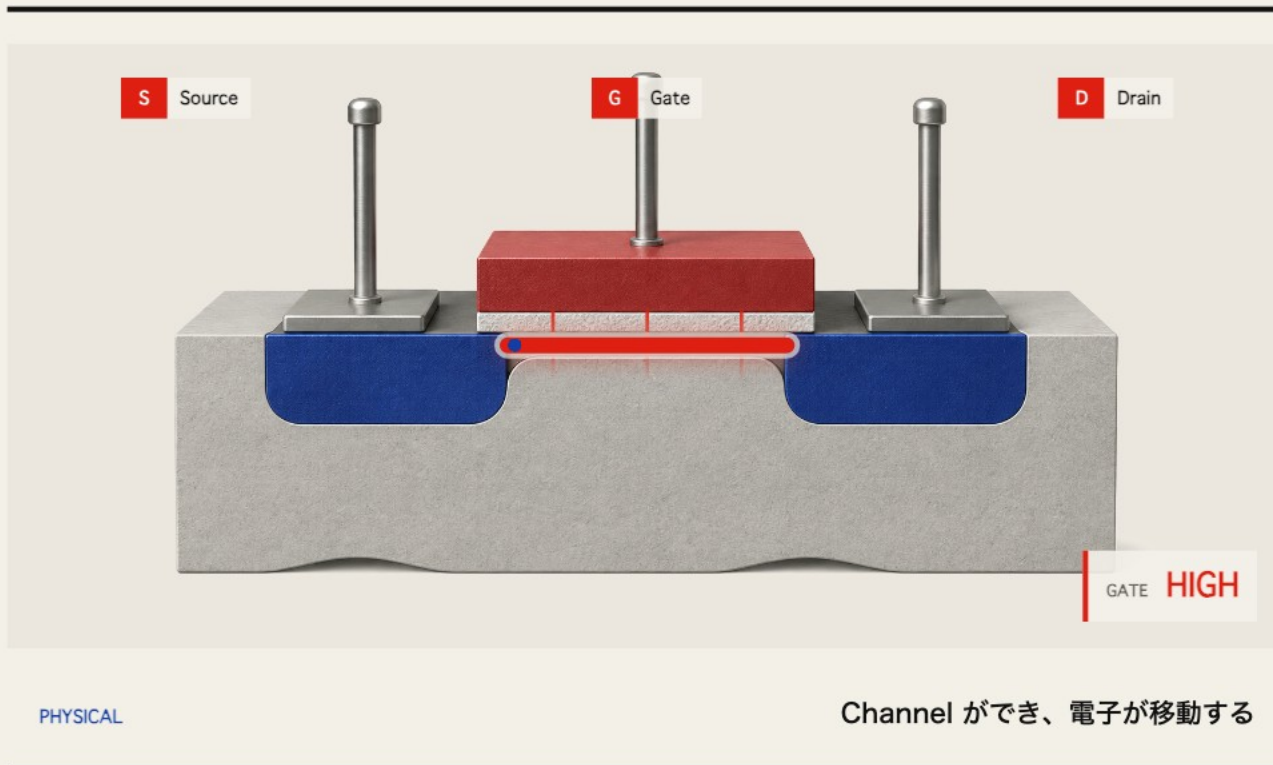


Body / 内蔵ダイオードは省略

G
電圧で制御する端子

D / S
電流が通る両端

導通する／しないを、ON/OFF として扱う



GATE INPUT

0 / LOW

1 / HIGH

CHANNEL

ON

- 1 Gate に電圧を与える
- 2 Channel が形成される
- 3 Source → Drain に電子が移動

電子の動き

遅い

速い

物理 : Channel の有無 → デジタル : ON

nMOS と pMOS は、反対の入力で ON になる

SAME GATE INPUT / A

0 / LOW

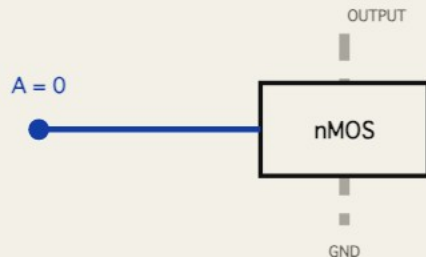
1 / HIGH

同じ入力を、両方に与える

N-TYPE MOSFET

nMOS

1 で ON



GATE LOW

OFF

0 → OFF

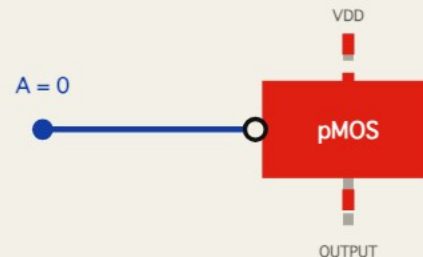
1 → ON

主に出力を GND 側 (0 V) へつなぐ

P-TYPE MOSFET

pMOS

0 で ON



GATE LOW

ON

0 → ON

1 → OFF

主に出力を VDD 側 (電源) へつなぐ

OPPOSITE



動きが逆

安定した 0 / 1 の理想モデルでは、片方だけが ON。これが **Complementary** (相補的)。

二つを組み合わせると、反対の値が出る

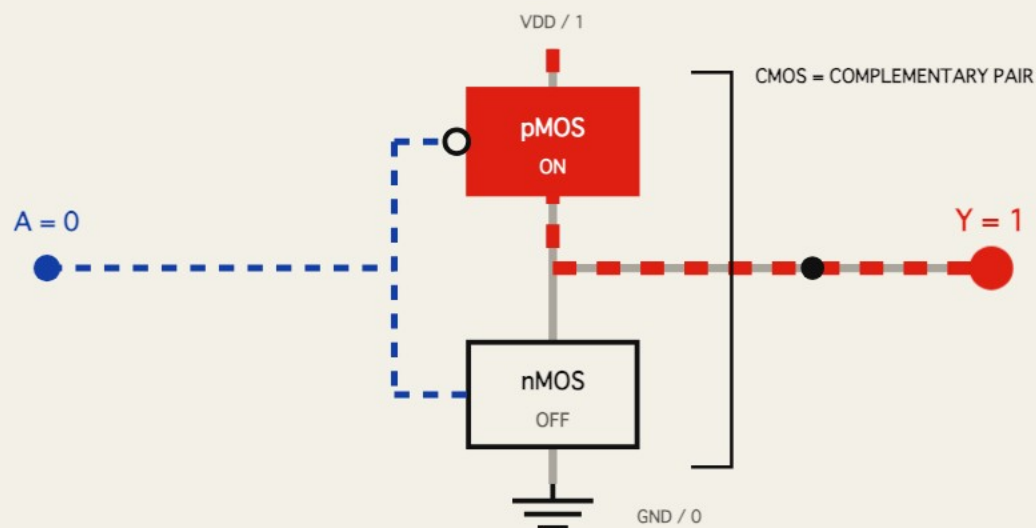
INPUT / A

0 / LOW

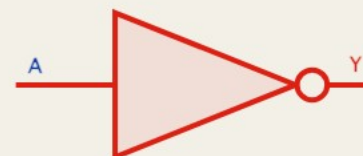
1 / HIGH

入力を切り替えて、赤い経路を追う

PHYSICAL CIRCUIT / CMOS INVERTER



DIGITAL RULE

 $A = 0 \rightarrow Y = 1$


NOT
 $Y = \text{NOT } A$

A	Y
0	1
1	0

1 A を二つへ

2 pMOS が ON

3 Y が VDD へ

4 NOT として使う

トランジスタから、最初の論理ゲートへ

PHYSICAL WORLD

電圧と電流の世界 DIGITAL WORLD

0 / 1 の規則の世界

01 / DEVICE

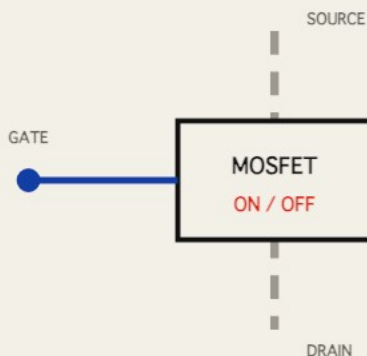
TRANSISTOR

02 / CIRCUIT

CMOS

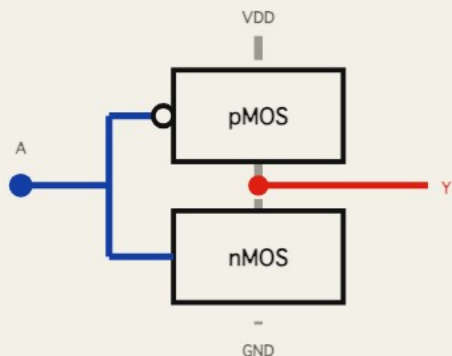
ここが境界 LOGIC

NOT GATE



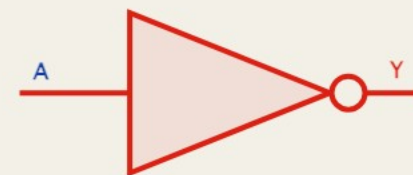
一つの電気スイッチ

組み合わせる



nMOS + pMOS の回路

抽象化する



$A = 0 \rightarrow Y = 1$

$A = 1 \rightarrow Y = 0$

一つの論理規則

DEVICE → CIRCUIT

中がどう動くかを見る LOGIC

入力と出力だけを見る

部品を組み合わせ、動きを 0 / 1 の規則として名前づけると、論理ゲートになる。

トランジスタを、レゴのように組み合わせる



01 / TRANSISTOR

一つのスイッチ

02 / COMBINATION

いくつかを接続

03 / FUNCTION

役割をもつまとまり

04 / CPU

膨大な回路の集合

小さな部品も、組み合わせることで「計算する装置」になる。

増え方には、何か規則があるのか



REPRESENTATIVE PROCESSOR

1971

Intel 4004

TRANSISTORS

2,300

PROCESS

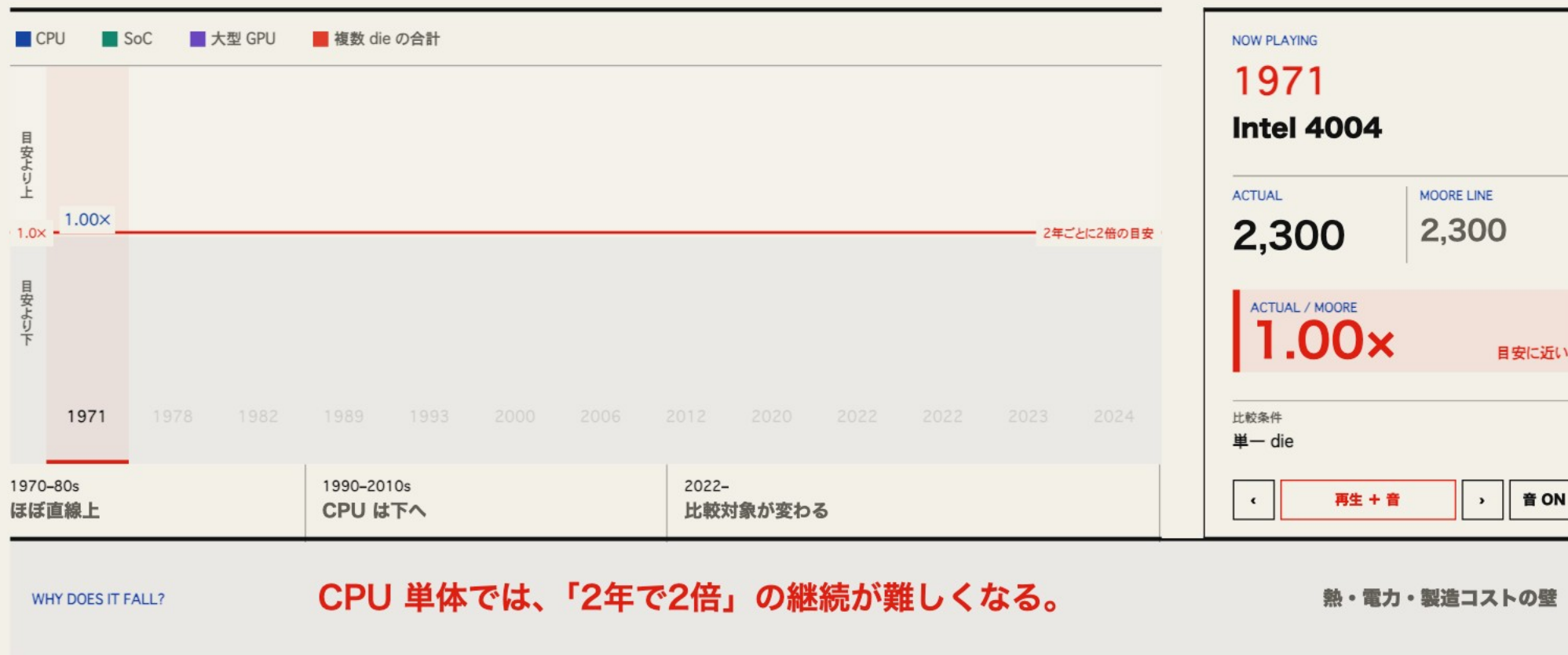
10 μ m

この増え方に、
規則はあるだろうか。

◀ 再生 ▶ 全体

2,300 → 14億。増え方そのものに注目する。

「2年で2倍」の直線から、次第に外れていく

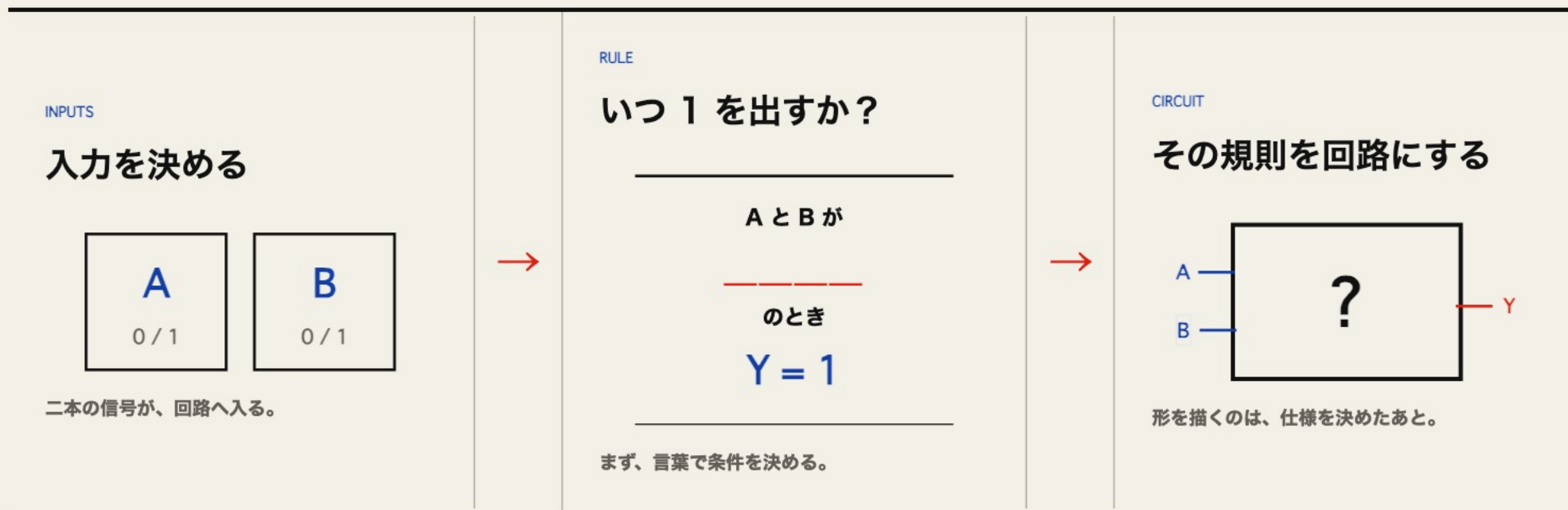


ここからは、0 / 1 の「規則」を見る



内部の物理をいったん置き、回路の「ふるまい」を見る。

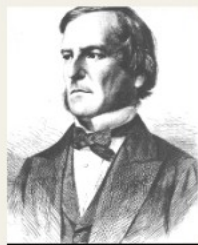
回路を描く前に、出力の規則を決める



先に決めるのは、形ではなく**入力と出力の関係**。

その関係を、どうすれば漏れなく確かめられる？

ブール論理が、規則と回路をつないだ



1854 / LOGIC

言葉の論理を
真 / 偽で書く

GEORGE BOOLE
1815-1864

「かつ」「または」「ではない」を、計算できる規則として表した。

COMMON LANGUAGE

ブール論理

真 / 偽を、二つの値として扱う論理

真 1 偽 0

「A と B が、ともに真」

$A = 1, B = 1 \rightarrow Y = 1$



1937 / CIRCUIT

0 / 1 の規則を
開 / 閉で実装

CLAUDE SHANNON
1916-2001



OPEN



CLOSED

スイッチの接続が、規則を実行する回路になる。

論理を 0 / 1 の規則 に翻訳できるから、回路として作れる。

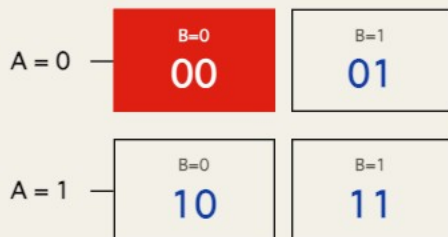
では、すべての入力を漏れなく確かめるには？

0 / 1 の規則を、漏れなく一行ずつ並べる

1 / ENUMERATE

入力を枝分かれさせる

A が 2 通り。各 A に対して、B も 2 通り。

 $2 \times 2 = 4$ 通り

2 / ONE CASE PER ROW

一組ずつ、一行にする

A	B	Y
0	0	?
0	1	?
1	0	?
1	1	?

3 / READ ONE ROW

1 行目

A = 0

B = 0

のとき

Y = ?

<

1 / 4

>

THIS IS A

真理値表

入力の全組合せと、
それぞれの出力を並べる。

入力はそろった。次は「どのとき Y = 1 にするか」を決める。

では、ログインを許可する $Y = 1$ の条件を決めよう

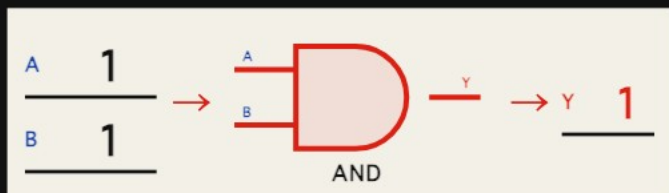
ログインできる $Y = 1$ は、A と B のどの組合せ？

AND は、「すべての条件」を確認する

AND / 論理積

すべての条件が 1

そのときだけ、出力 Y は 1



真理値表・赤は Y = 1 となる条件

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

一つでも 0 があれば、Y は 0。

WHERE AND APPEARS

「全部そろった？」を確認する場所



認証・権限

ID が一致 AND パスワードが一致
ログインや入室を許可する。

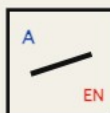
許可



安全インターロック

扉が閉じた AND 非常停止を解除
条件がそろってから機械を動かす。

運転



デジタル回路

信号 A AND ENABLE = 1
必要なときだけ信号を先へ通す。

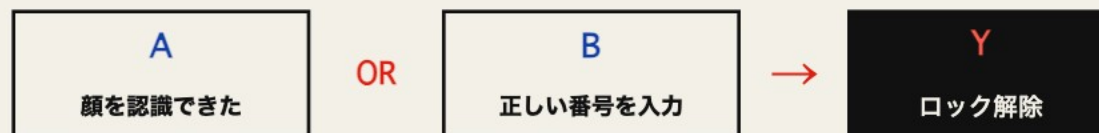
通す

スマートフォンを開く方法は、一つとは限らない



EVERYDAY DECISION

二つの方法を、
両方成功させる必要はない

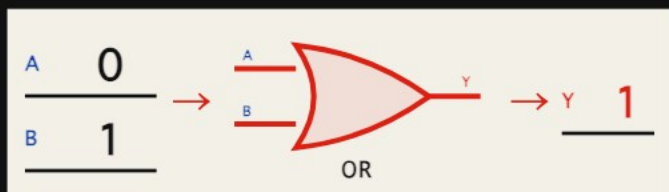


どの組合せなら $Y = 1$ になる？

OR は、「どれか一つでも」を確認する

OR / 論理和

少なくとも一つが 1

A、B のどちらか、または両方が 1 なら $Y = 1$ 真理値表・赤は $Y = 1$ となる条件

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

両方が 1 の場合も、Y は 1。

WHERE OR APPEARS

「どれか一つでも？」を確認する場所

FACE
OR
PIN

代替手段

顔認証 OR パスコード

どちらかが成功すればロックを解除。

解除

SMOKE
OR
HEAT

警報・監視

煙を検知 OR 温度が上昇

どれか一つの異常から警報を出す。

警報

REQ A
OR
REQ B

デジタル回路

要求 A OR 要求 B

複数の要求を一つの信号へまとめる。

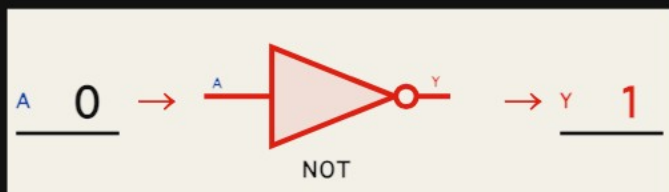
通知

NOT は、入力の意味を反対にする

NOT / 論理否定

入力の意味を反対にする

0 は 1 に、1 は 0 に変わる



真理値表・赤は Y = 1 となる条件

A	Y
0	1
1	0

入力の一つ。小さな丸が 反転 の印。

WHERE NOT APPEARS

「反対なら？」を作る場所

LIGHT
NOT
DARK

自動照明

明るいを NOT して「暗い」へ
暗くなったら照明を点灯する判断。

点灯

/RESET
NOT
RESET

active-low 信号

0 で有効 NOT → 1 として扱う
反対向きの信号を、読みやすく変換する。

反転

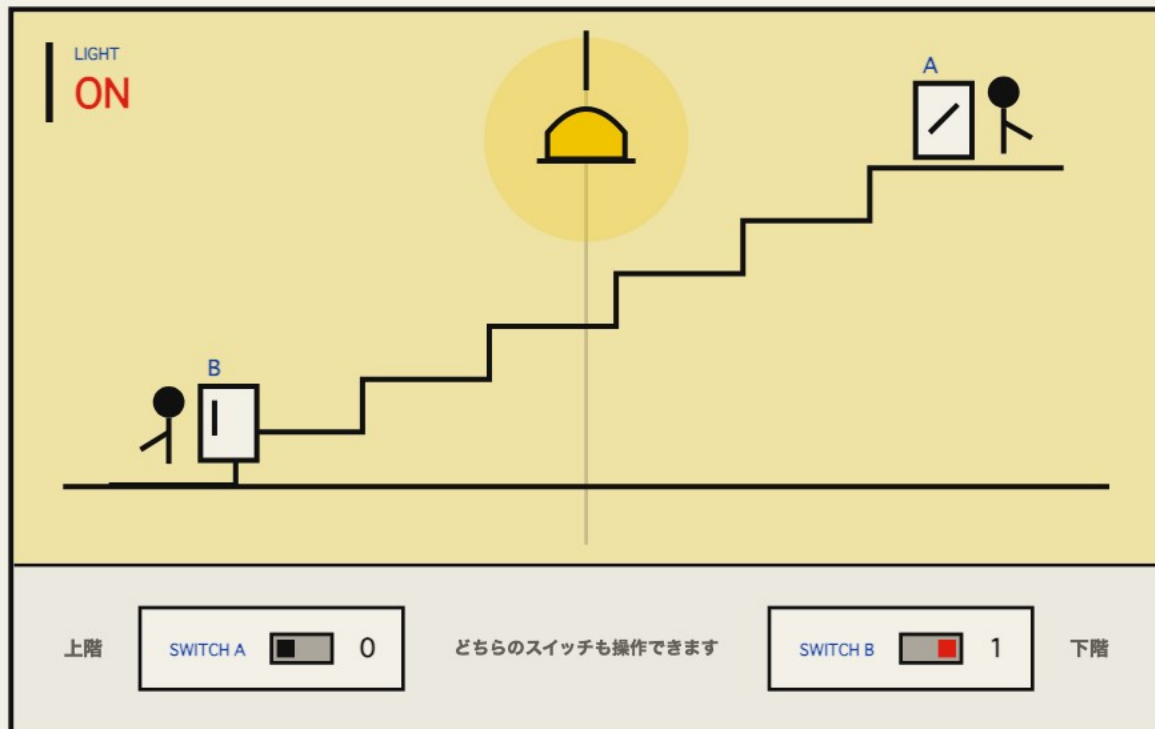
TRUE
NOT
FALSE

プログラム

NOT logged_in
ログインしていなければ案内を表示する。

表示

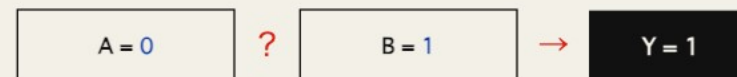
階段の照明を、二つの場所から切り替える



TWO-WAY LIGHTING

上と下、どちらからでも
照明を切り替えたい

二つのスイッチを押して、
点灯する組合せを探してみよう。

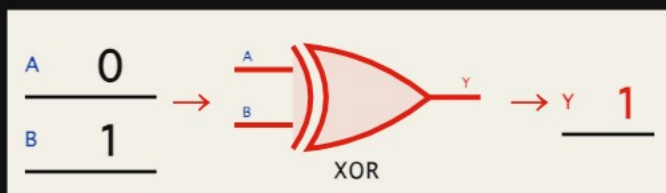


この配線では、二つの向きが異なる。

XOR は、「二つが違う」を確認する

XOR / 排他的論理和

二つの入力異なる

0,1 または 1,0 のときだけ $Y = 1$ 真理値表・赤は $Y = 1$ となる条件

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

同じなら 0、異なれば 1。

WHERE XOR APPEARS

「違いがある？」を確認する場所

OLD
XOR
NEW

違いの検出

以前の値 XOR 現在の値
状態が変化するときだけ 1 を出す。

変化

SW A
XOR
SW B

双方向の操作

スイッチ A XOR スイッチ B
この配線では向きが異なると点灯する。

点灯

A
XOR
B

1 bit 加算

$A \text{ XOR } B \rightarrow \text{和 } S$
桁上がりを除いた一桁の和になる。

和 S

同じ規則を、四つの方法で表す

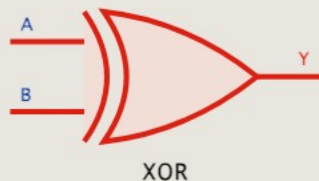
01 / MEANING



二つの値が異なる

まず、言葉で意味を決める。

02 / SYMBOL



配線図で使う

形を見れば、回路を追える。

03 / EXPRESSION

$$Y = A \oplus B$$

式で短く書く

計算や設計を簡潔に表す。

04 / TRUTH TABLE

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

全ケースを確認する

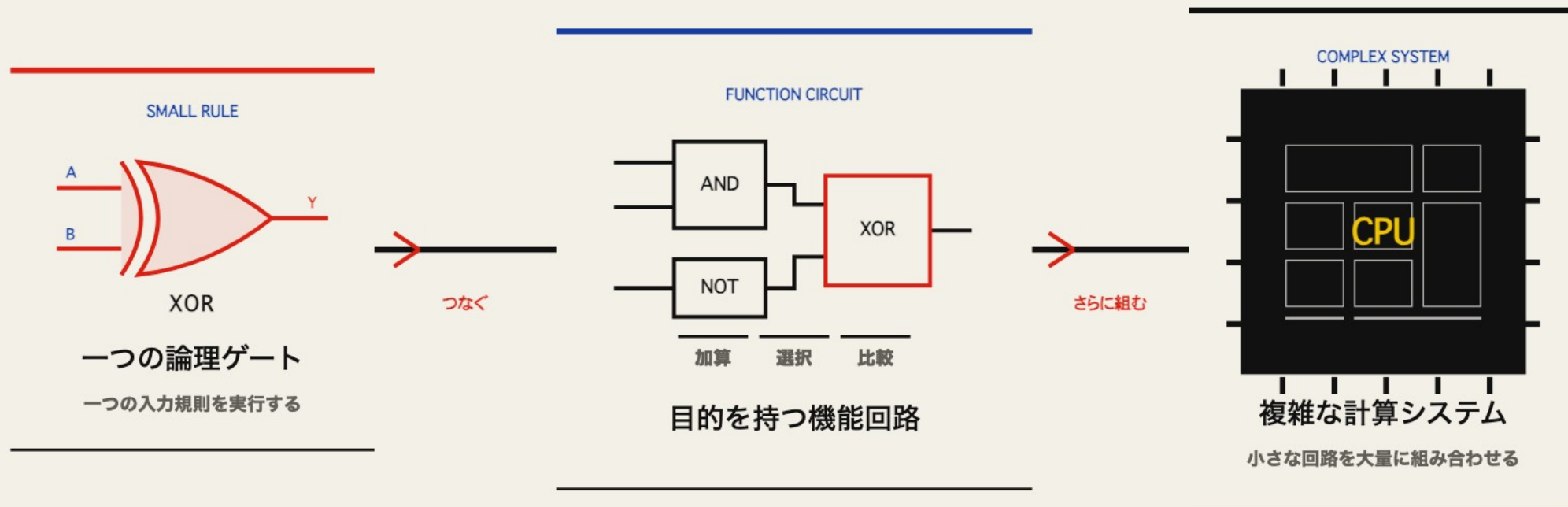
漏れなく仕様を決められる。

意味

同じ規則を、目的に応じて翻訳している。

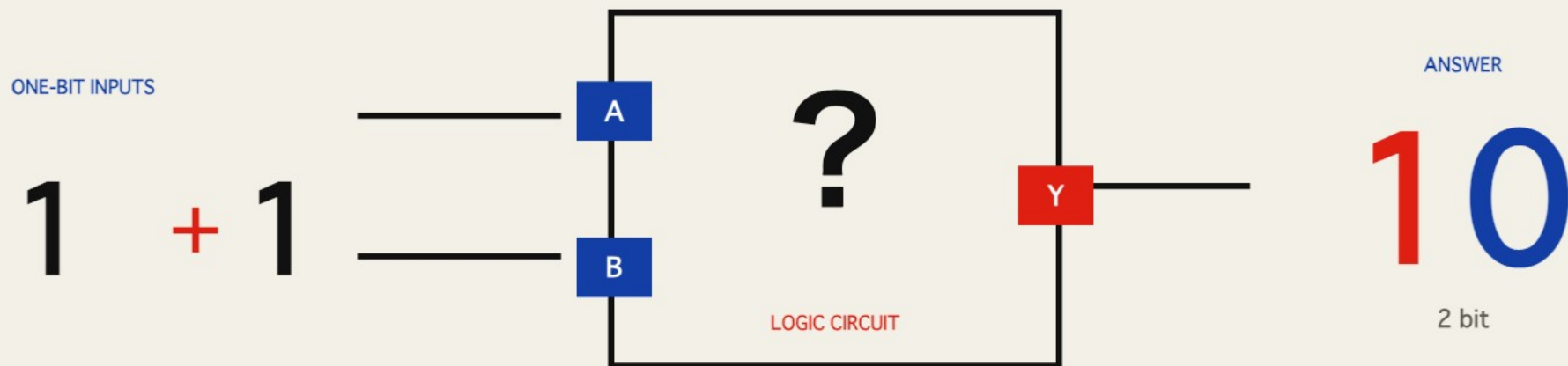
実装

論理ゲートは、組み合わせて使う



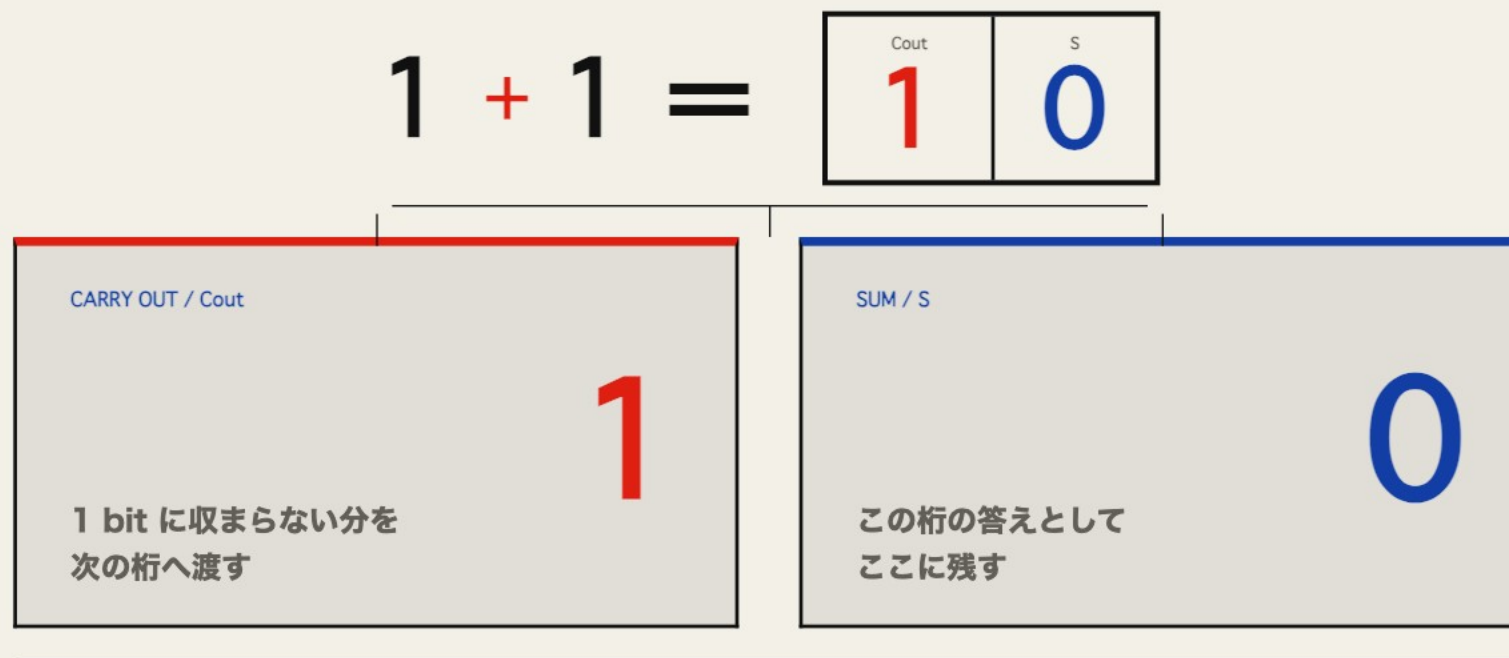
小さな 0 / 1 の規則を接続すると、単独の gate にはできない機能が生まれる。

「1 + 1」を、論理回路で計算できるか



出力が一本だけでは、答えの 10 を表せない。

答えの「10」には、二本の出力が必要



10 を二桁の数字として眺めず、役割の違う二本の出力信号として読む。

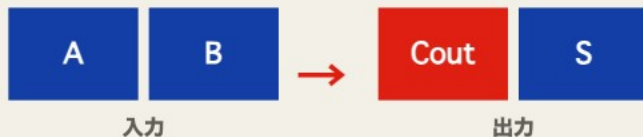
必要な入出力から、真理値表を作る

01 / 06

STEP 1 / DEFINE COLUMNS

必要な列を決める

入力 は A と B。出力 は、その桁に残す S と、次の桁へ渡す Cout。



INPUT		OUTPUT	
A	B	Cout	S
"	"	"	"
"	"	"	"
"	"	"	"
"	"	"	"
"	"	"	"

四つの結果が、加算器の仕様になる

A	B	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Cout

0 0 0 1

いつ 1 になる？

S

0 1 1 0

いつ 1 になる？

S の並びは、XOR と一致する

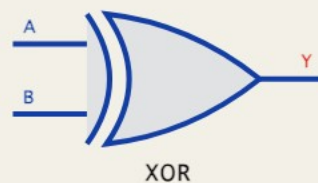
ADDITION OUTPUT

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

同じ並び



XOR TRUTH TABLE



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

$$S = A \text{ XOR } B$$

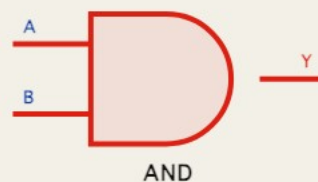
Cout の並びは、AND と一致する

ADDITION OUTPUT

A	B	Cout
0	0	0
0	1	0
1	0	0
1	1	1

AND TRUTH TABLE

同じ並び

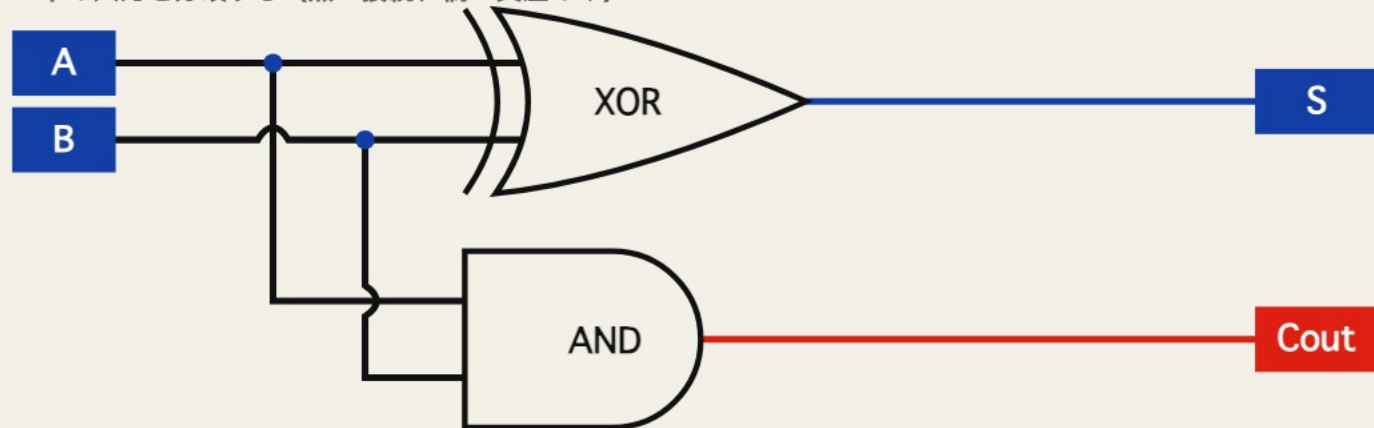


A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{Cout} = A \text{ AND } B$$

XOR と AND で、半加算器を作る

二本の入力を分岐する（点＝接続、橋＝交差のみ）



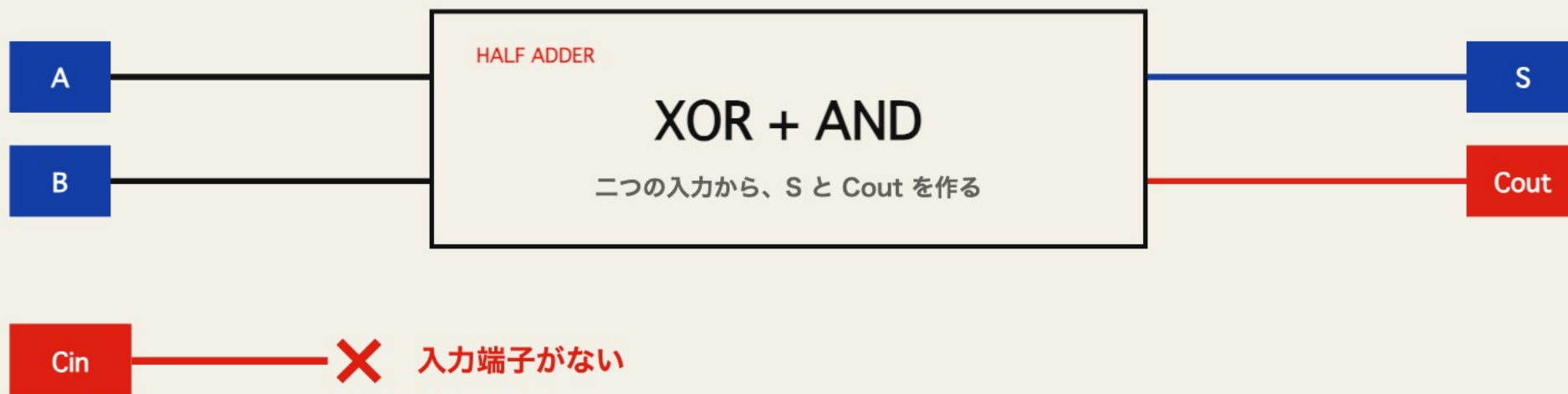
$$S = A \text{ XOR } B$$

半加算器 / Half Adder

$$\text{Cout} = A \text{ AND } B$$

二つの 1 bit を足し、和と桁上がりを同時に出す最小の組合せ回路。

Half Adder は、下位桁からの **Cin** を受け取れない



A、B、Cin の三入力には、 八つの組合せがある

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

INPUT PATTERNS

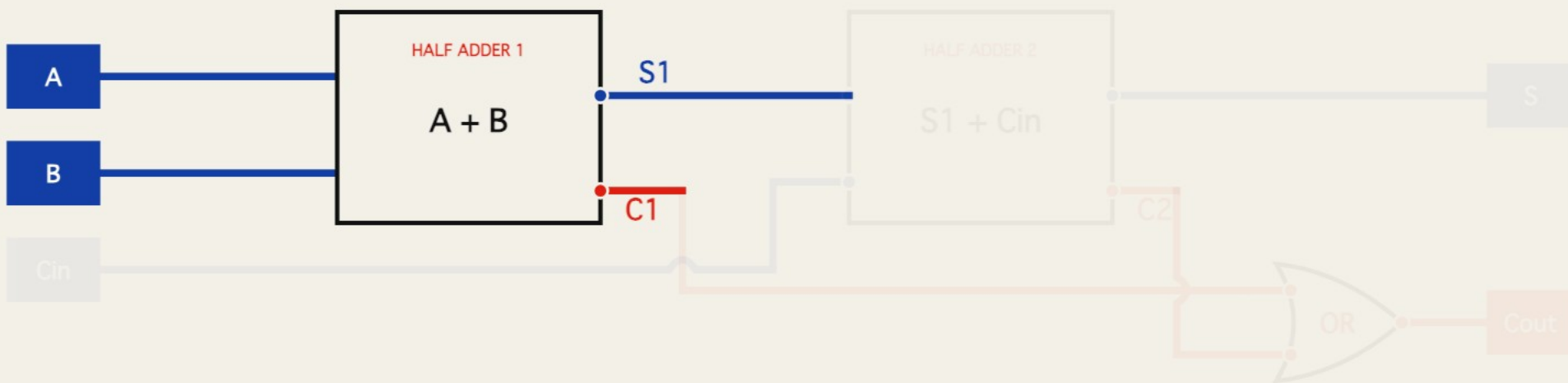
$$2^3 = 8$$

A、B、Cin の
全組合せを並べる

A + B + Cin

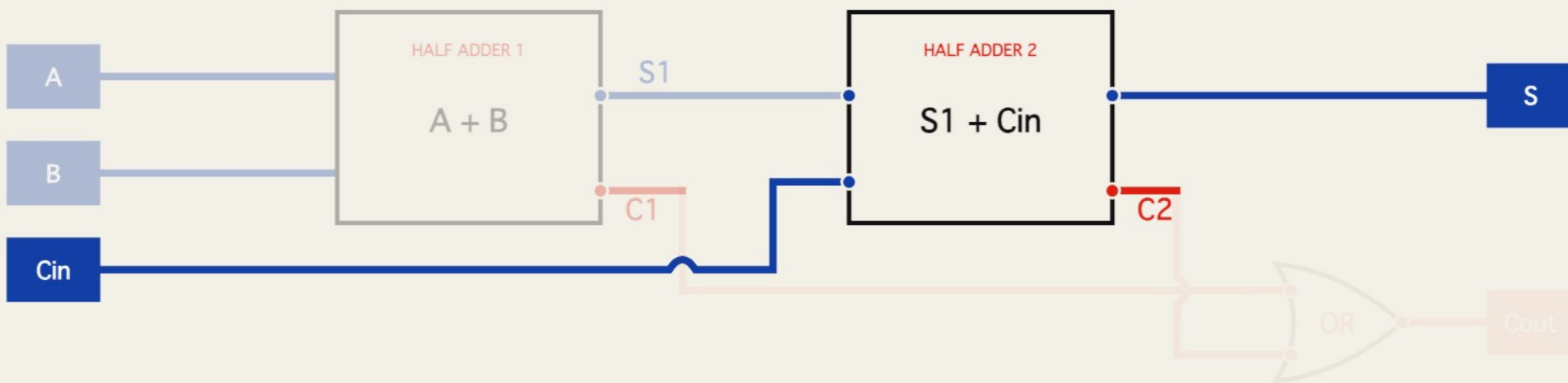
0 / 1 / 2 / 3

一つ目の Half Adder で、 A と B を加算する



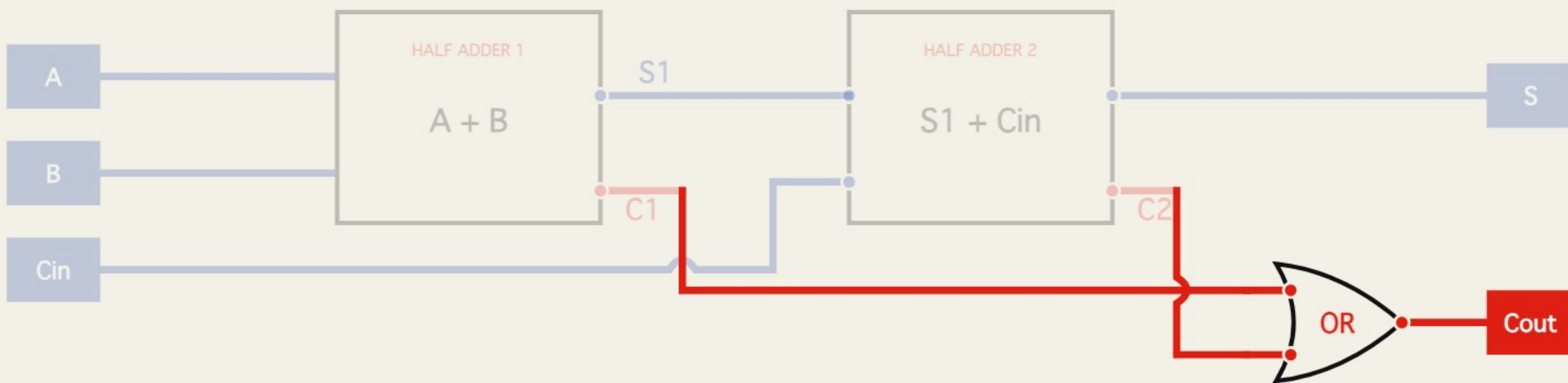
途中結果は **S1**、途中の桁上がりは **C1**。

二つ目の Half Adder で、 S1 と Cin を加算する

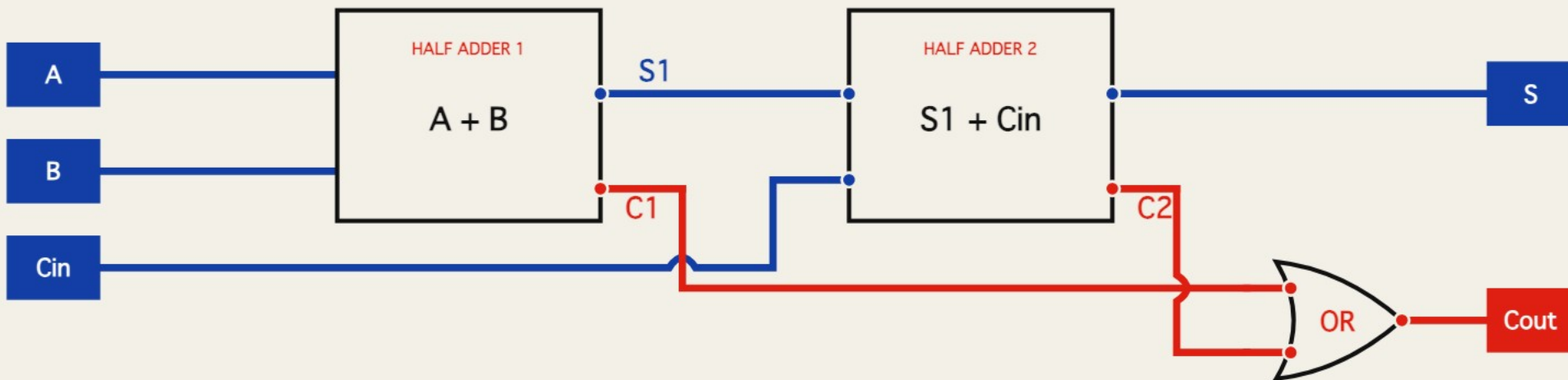


ここで最終的な **S** と、もう一つの桁上がり **C2** が出る。

C1 または C2 が 1 なら、 Cout は 1



二つの Half Adder と OR で、 全加算器 (Full Adder) が完成する



$$\{Cout, S\} = A + B + Cin$$

例 : $A=1, B=1, Cin=0 \rightarrow S1=0, C1=1, C2=0 \rightarrow Cout=1, S=0$

Full Adder を並べ、下位の Cout を上位の Cin へ渡す

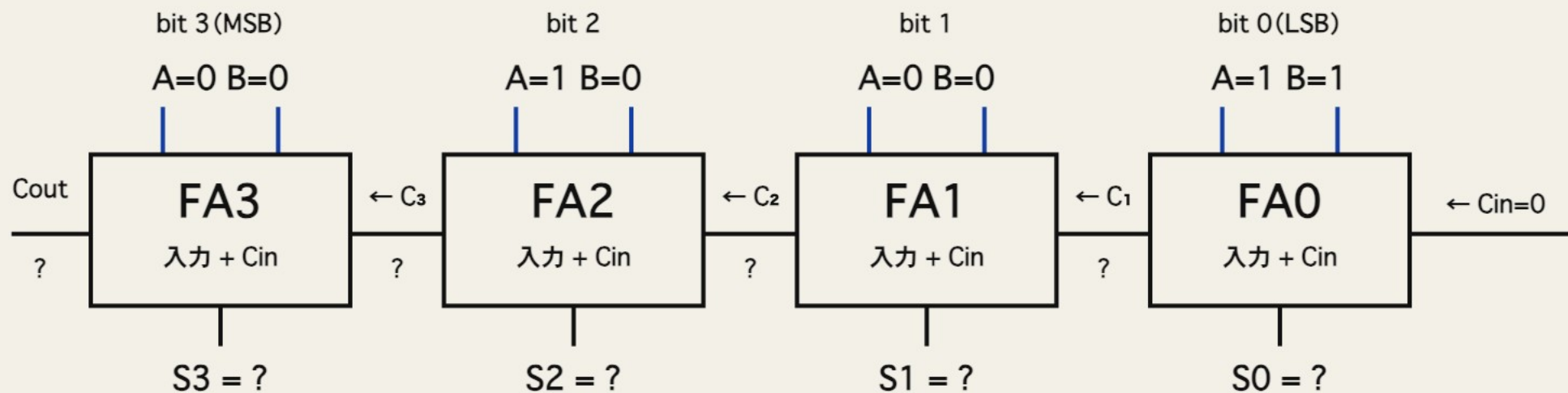
0101 + 0001 ← 下位桁から順に確定する

再生

1 桁進む

最初から

完成形



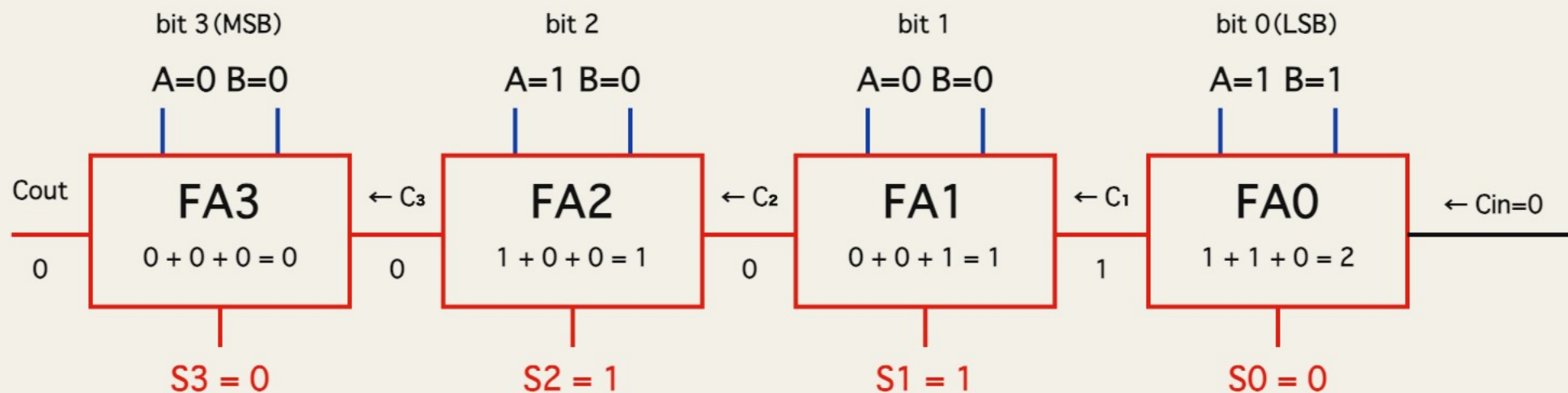
0 / 4 桁が確定

赤は確定した経路。0 / 1 は数字で読む。

右の LSB から左の MSB へ、Carry が波のように進む。

RCA は単純だが、桁数が増えると待ち時間が伸びる

0101 + 0001 ← 下位桁から順に確定する



最悪の場合: Cin → FA0 → FA1 → FA2 → FA3 → Cout

Carry が全段を伝わる入力では、桁数とともに待ち時間が伸びる。

2 の補数を使えば、 減算を加算として扱える

LECTURE 2 RECALL · GENERAL RULE

$$A - B = A + (\text{NOT } B) + 1$$

01 B	02 NOT B	03 + 1	04 ADD	
------	----------	--------	--------	--

EXAMPLE · 4 BIT

$$5 - 1$$

= ?

4 bit 符号付き
-8 ... +7

引く数 B	0001	+1
全 bit を反転	1110	NOT B
さらに +1	1111	-1
0101 + 1111	Cout 1 4 BIT RESULT 0100	答え = +4

前回：2 の補数で「-B」を作った

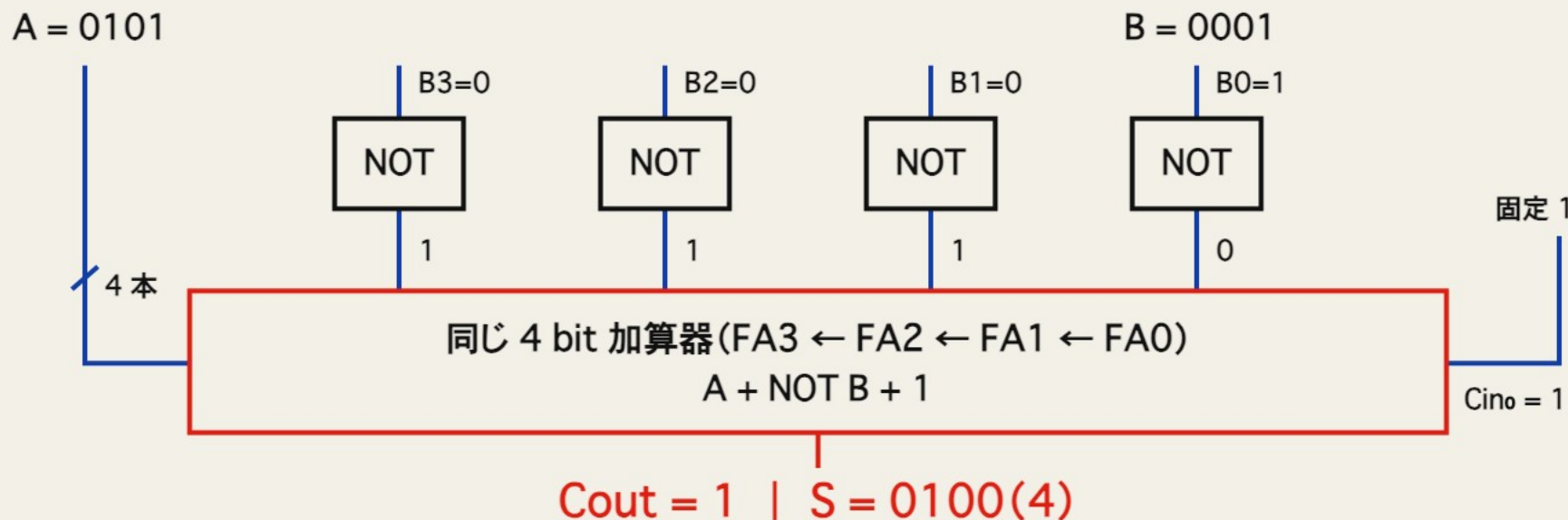


今回：同じ操作を回路にさせる

加算器の入力を変えれば、 減算器として使える

$$A - B = A + (\text{NOT } B) + 1$$

例: $5 - 1 = 4$

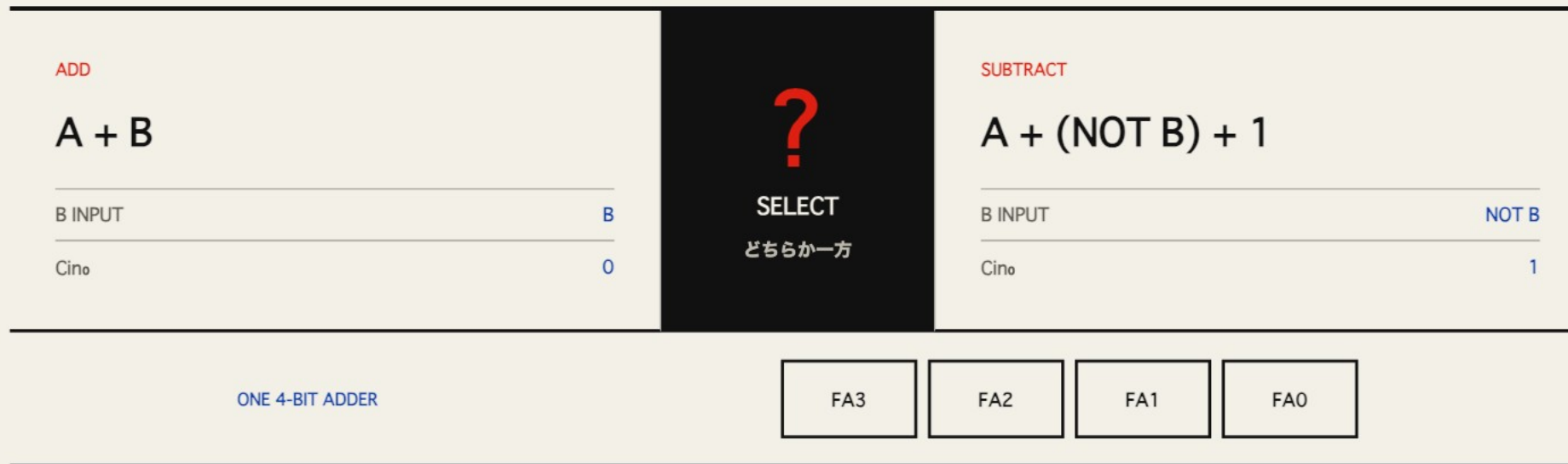


B の 全 bit を NOT に通し、最低位の Cin_0 を 1 に固定する。

一台の加算器で、 加算と減算を切り替えられる？

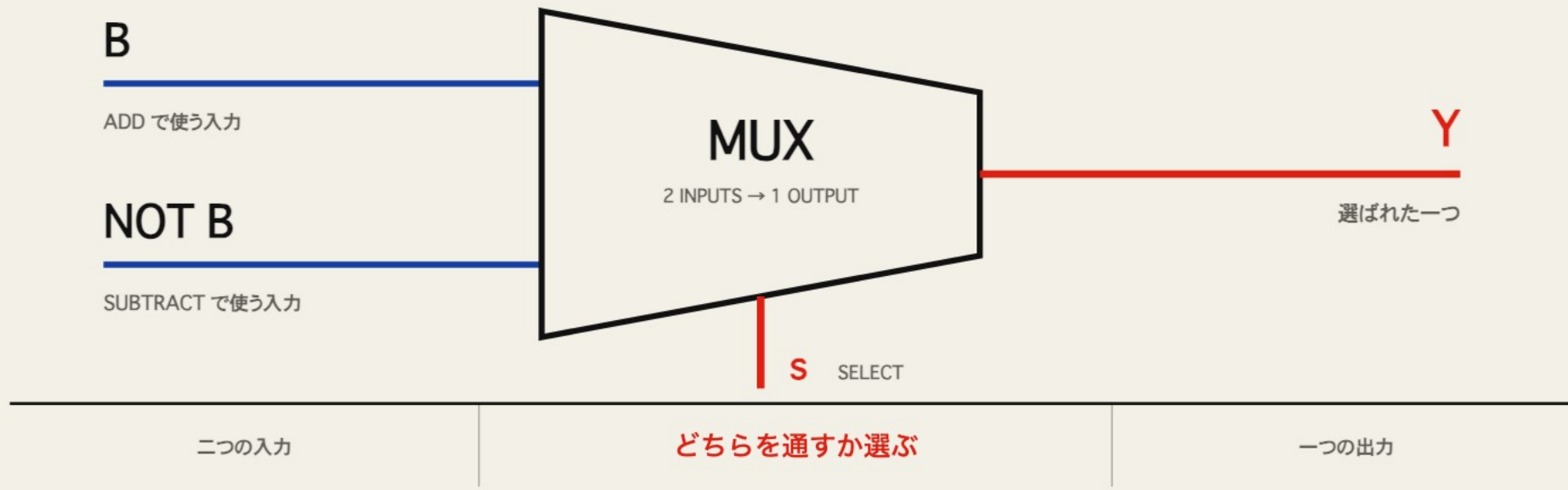
SAME A · SAME B · DIFFERENT INPUT RULE

一台の加算器へ、どちらの B を送る？



必要なのは、候補から一つを選ぶ回路。

B と NOT B から、 一つだけ選ぶ回路が必要になる

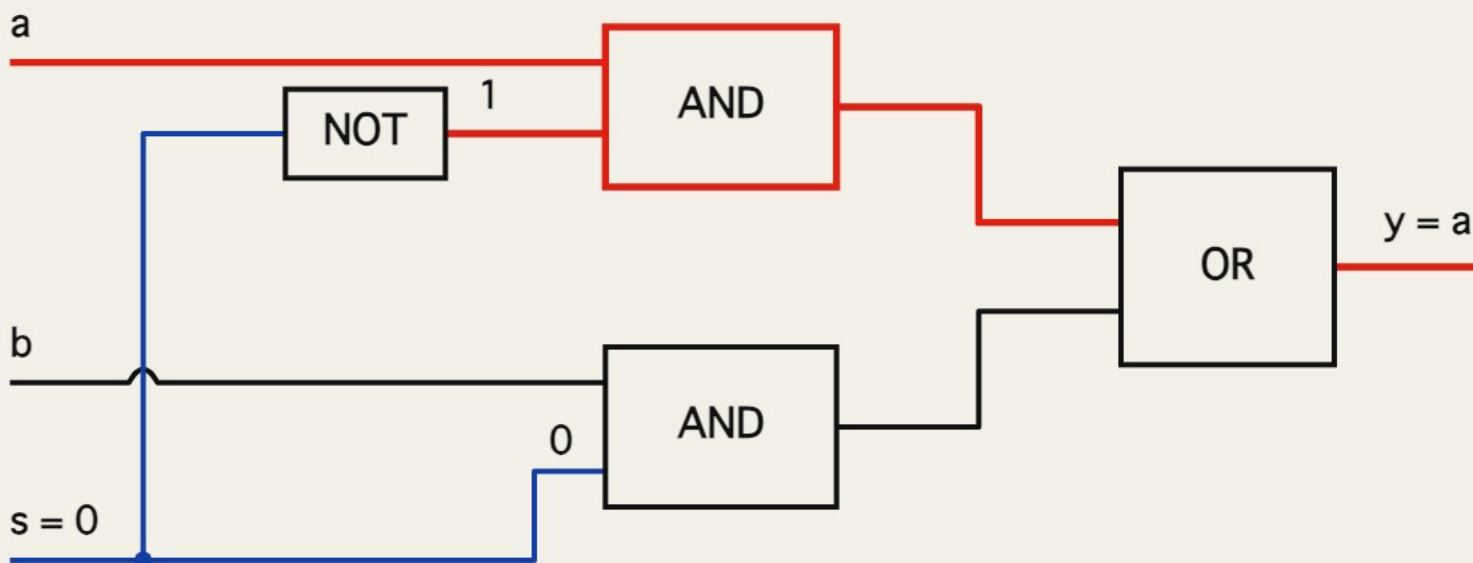


この「選ぶ」回路が MUX (マルチプレクサ)。

$s=0$ なら a 、 $s=1$ なら b を出力する

$$y = (a \text{ AND NOT } s) \text{ OR } (b \text{ AND } s)$$

$s = 0$	$s = 1$
---------	---------



s	y
0	a
1	b

選ばない側は
AND で 0 にする。

赤線：選ばれた経路
青線：選択信号

制御信号 s が、通す入力を決める。

SUB が B 側の入力と C_{in0} を同時に切り替える

ADD / SUB=0

SUBTRACT / SUB=1

$$5 + 1 = 6$$

$B \text{ XOR } 0 = B$

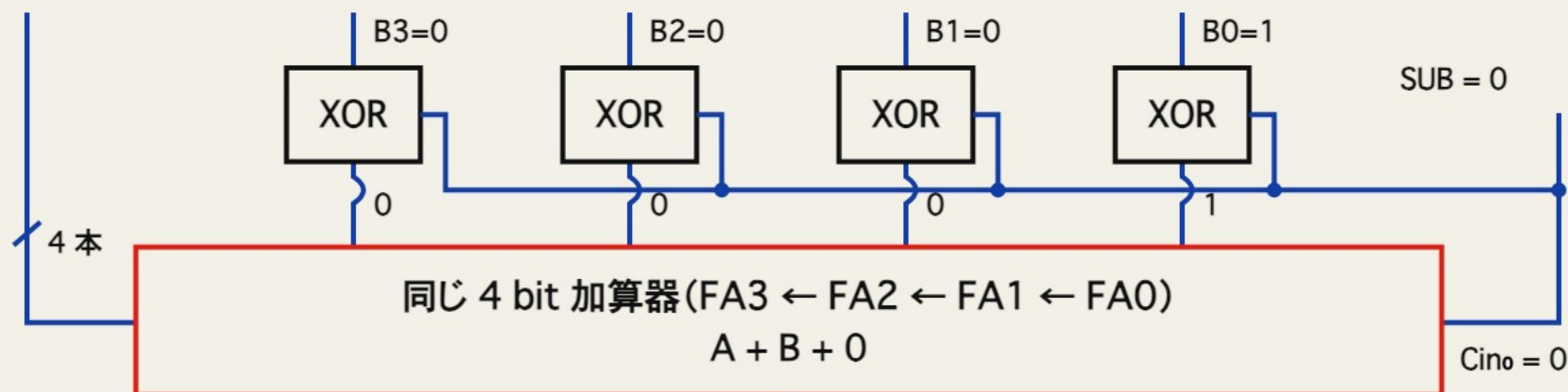
/

$B \text{ XOR } 1 = \text{NOT } B$

各 bit の XOR が B と NOT B を選ぶ。

A = 0101

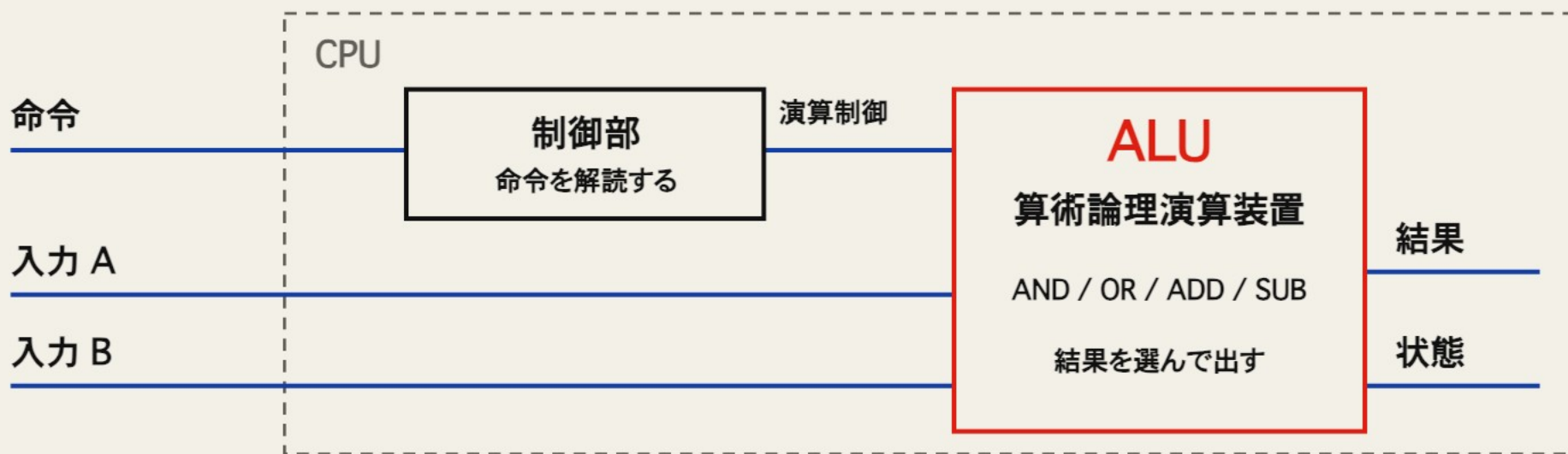
B = 0001



$$\text{Cout} = 0 \mid S = 0110(6)$$

青い SUB 線が、四つの XOR と C_{in0} を同時に制御する。

選べる計算を増やすと、 多機能な演算の中心になる



演算制御は、命令を解読して作る内部信号である。

Arithmetic Logic Unit — 複数の演算と、結果を選ぶ MUX を組み合わせる。

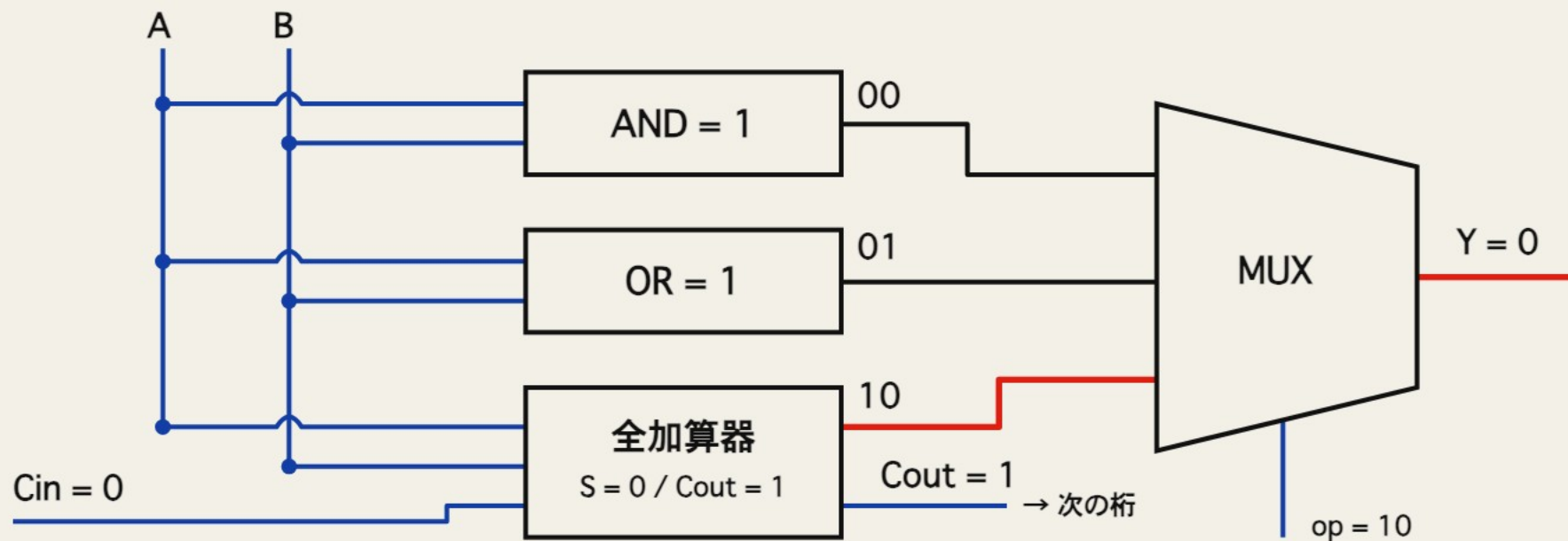
AND、OR、ADD などの候補から、 **op[1:0]** が一つを選ぶ

例: A = 1、B = 1、Cin = 0

00 AND

01 OR

10 ADD

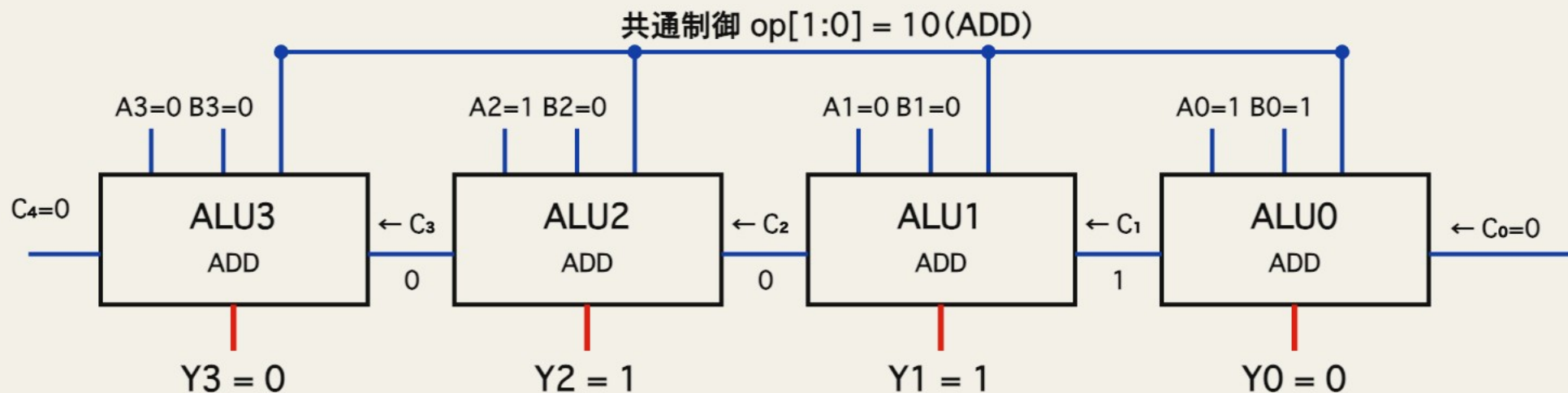


op[1:0] は 2 bit の選択信号 (00 / 01 / 10。11 は未使用)。点だけが分岐接続。

1 ビットの回路を並べ、 結果と Z / N / C / V を得る

A[3:0] = 0101 B[3:0] = 0001

[3:0] は bit 3 ~ 0 の 4 本をまとめた表記



0101 + 0001 = 0110 (6)

Z = 0 N = 0 C = 0 V = 0

Z: すべて 0 か N: 最上位 bit C: 桁上がり V: 符号付きの範囲超過

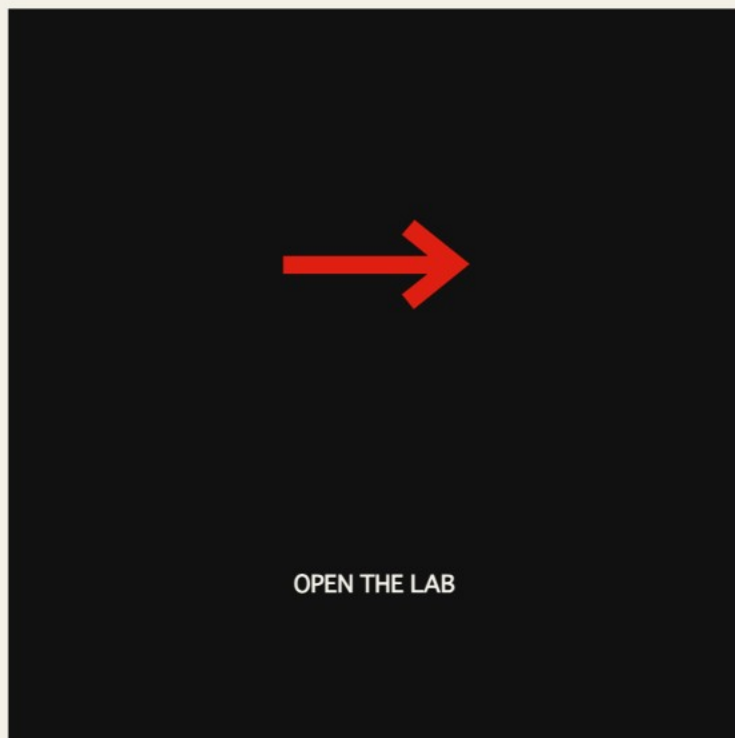
ALU は CPU の中で、**算術演算と論理判断**を実行する。

Carry・Borrow・Overflow は、 それぞれ別の問題を知らせる

符号なし加算 Carry	符号なし減算 Borrow	符号付き加算 Overflow
INTERPRET AS 符号なし加算 4 bit range · 0 ... 15 1111 + 0001 → 1 0000 Carry = 1		WHAT SHOULD WE CHECK? 16 は 4 bit の箱に入る？ 入らない。左の 1 が bit 幅の外へ出た。 箱の外へ出たか

同じ 4 bit でも、符号なし / 符号付き と演算の種類で確認すべき状態が変わる。

論理ゲートを、自分の手で 切り替えてみよう



A と B を切り替え、真理値表と同じ出力になるかを確認めます。まず gate、次に adder へ進みます。

c2lab.org/csf/logic-gate/

gate を確かめ、adder を作り、 4 bit の Carry を追う

01

AND / OR / NOT / XOR

基本・3 分：すべての入力組合せを切り替える

02

Half Adder / Full Adder

基本・4 分：S と Cout を別々に確認する

03

4 bit Ripple Carry Adder

発展・3 分：LSB から MSB へ Carry を追う

発展：c2lab.org/csf/circuit/rca4.html

電圧状態から、論理ゲート、 加算器、ALU まで積み上がった



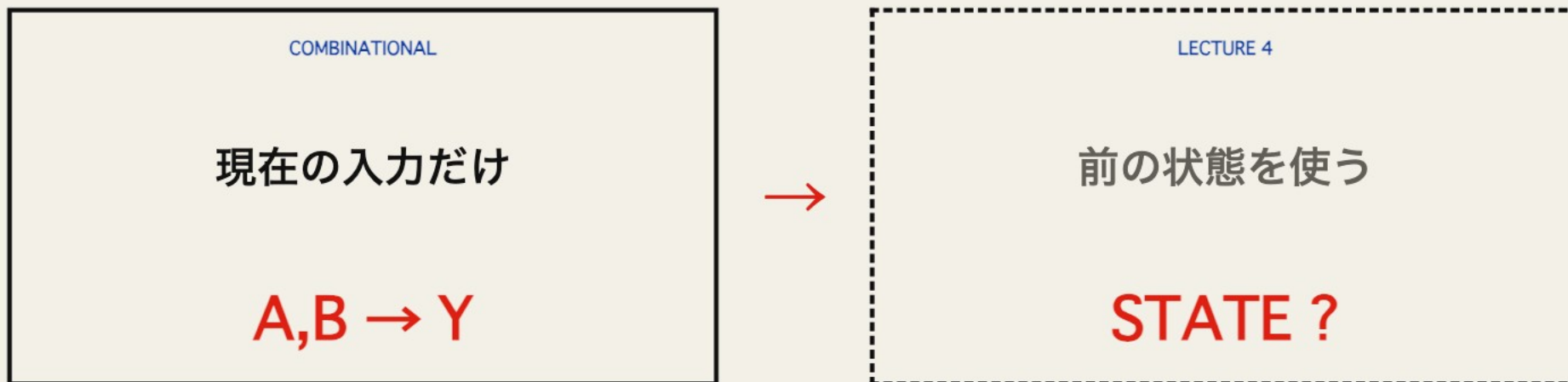
$$0101 + 0001 = 0110$$

bit 0 : $1 + 1 \rightarrow S_0=0, C_1=1$

bit 1 : $0 + 0 + C_1 \rightarrow S_1=1$

bit 2・3 : 上位も確定し、答えは 6

同じ入力なら同じ出力。
では、回路はどうやって「前」を覚える？



次回：順序回路と記憶