組合せ論理回路

組合せ論理回路

$$y_i = f_i(x_1, x_2, \ldots, x_n),\ i = 1, 2, \ldots, m$$

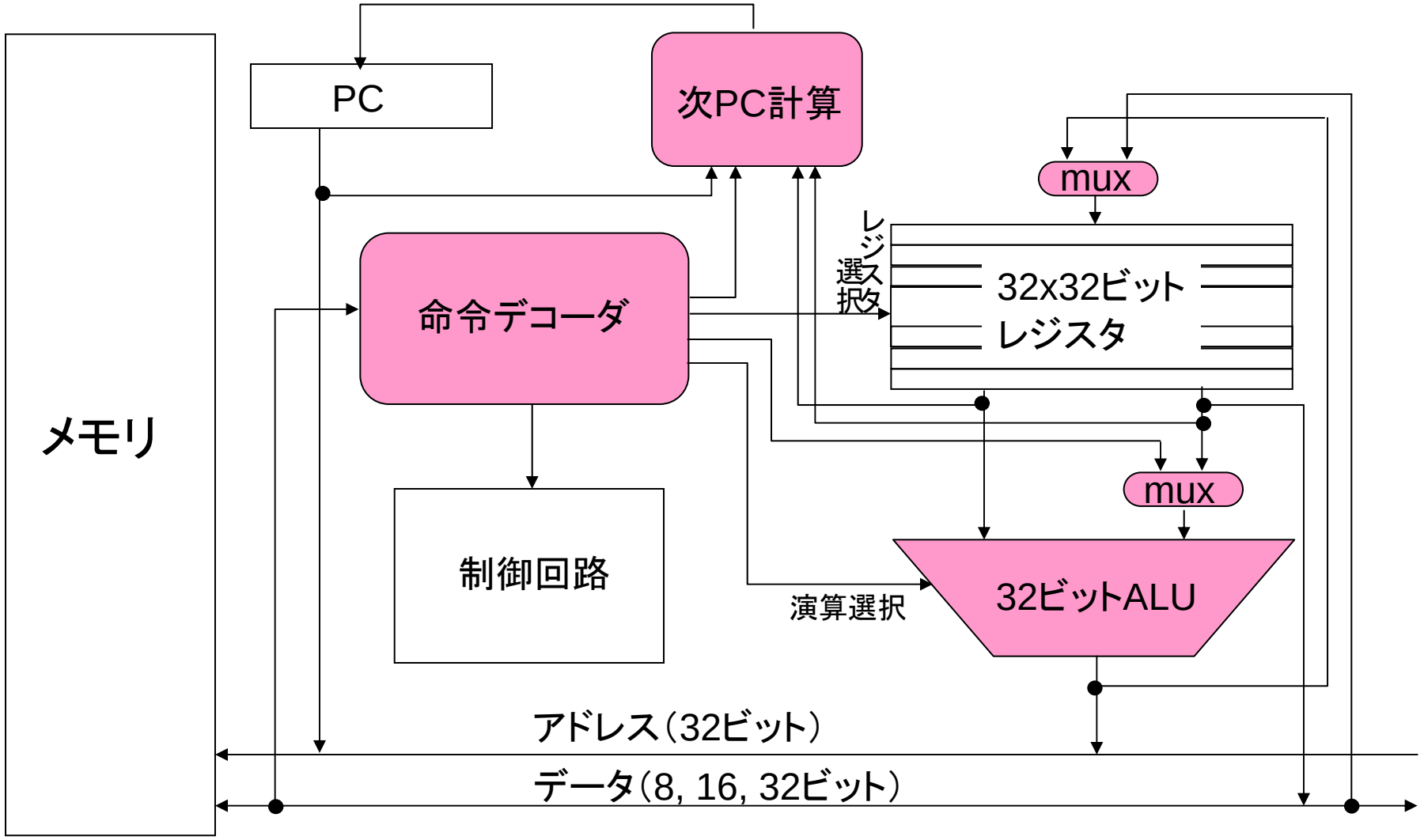
- ある時点での出力が，その時点の入力のみで決まる（記憶を持たない）回路
  - フィードバックが存在しない（入力→出力の方向にだけゲートが接続されている）
- 原理的には，n 入力の論理関数が m 個並んでいるものだと考えればよい

組合せ論理回路の構成方法

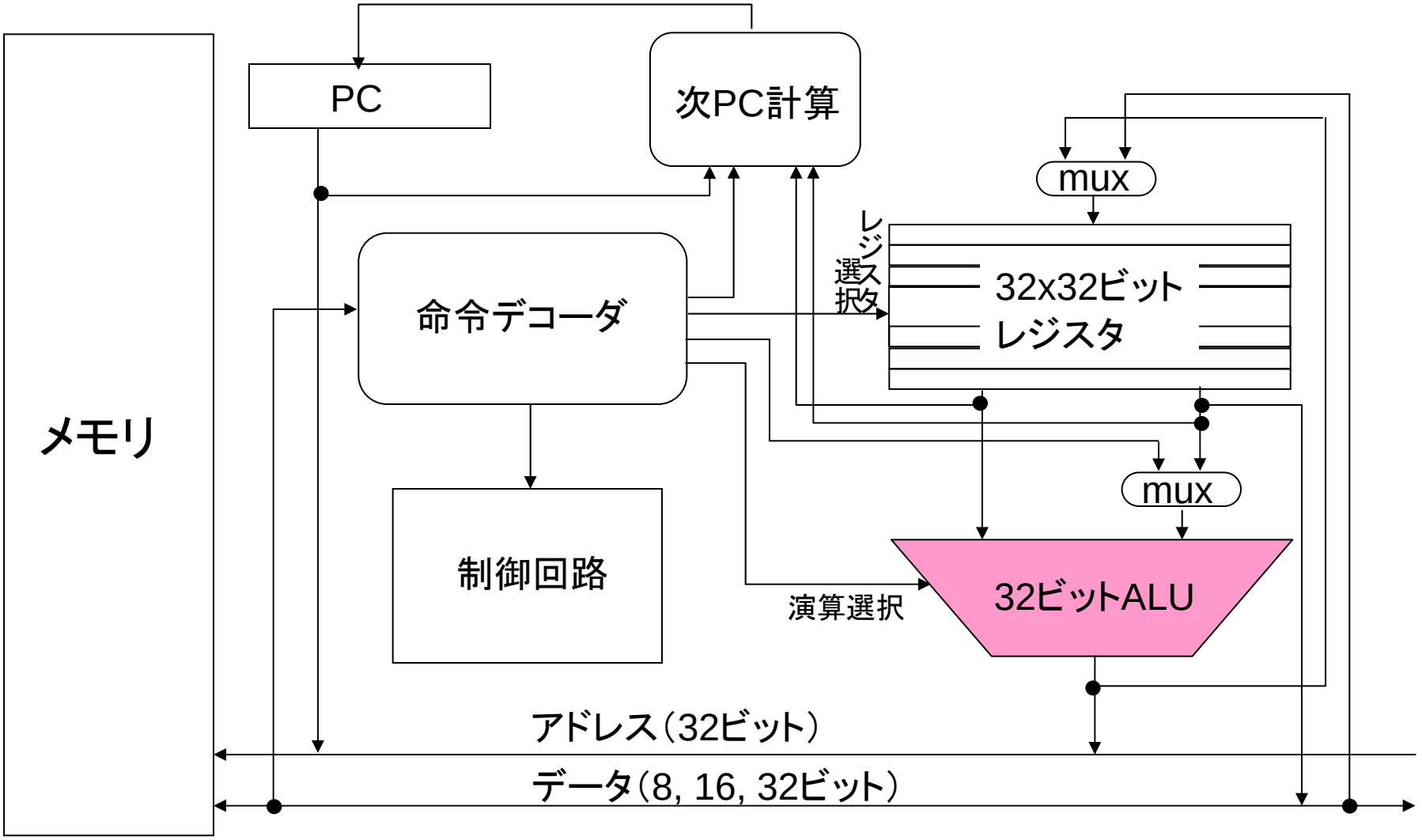
- 原理上は，必ず積和形回路で表すことができる，しかし
  - nが大きい場合，簡単化の計算に膨大なコストがかかる
  - それが最適とは限らない
- 算術論理演算のように入出力関係の規則性が高い場合は，その規則性に注目して回路を組み立てる方がよい

- 演算回路
  - 算術演算，論理演算
- 複数の回路を接続するための部品
  - 2進デコーダ，マルチプレクサ
- 複数回路の接続例
  - ALU

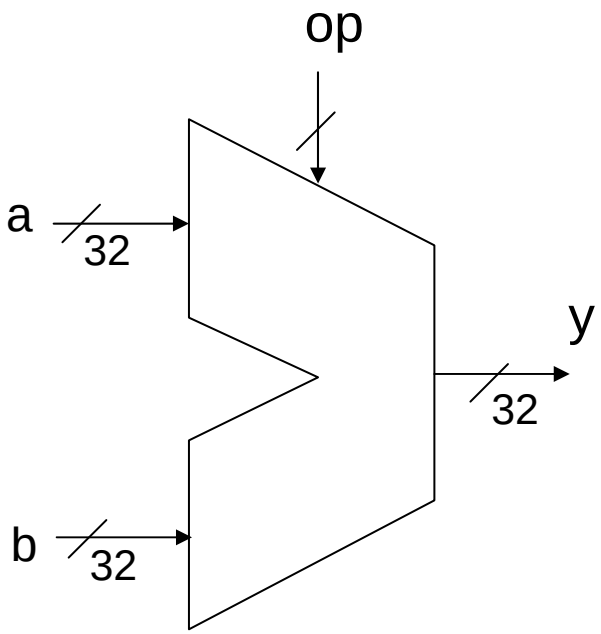
復習: MIPSの構造

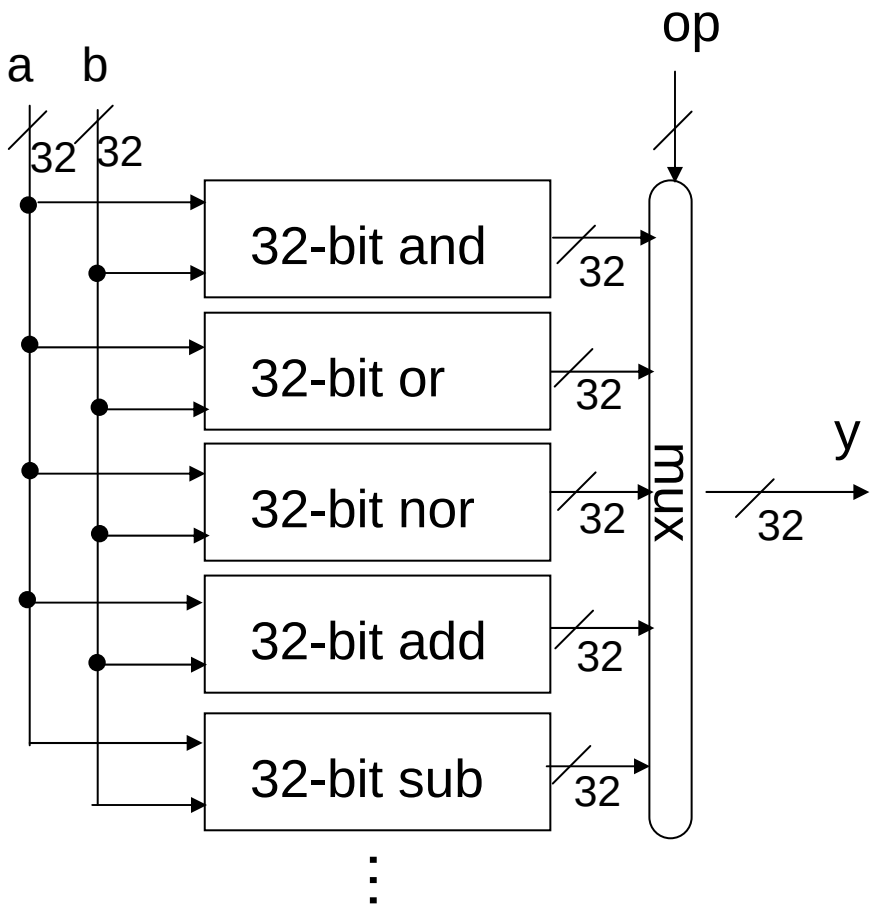

メモリ
PC
次PC計算
命令デコーダ
制御回路
mux
レジスタ選択
32x32ビット
レジスタ
mux
演算選択
32ビットALU
アドレス(32ビット)
データ(8, 16, 32ビット)

復習: MIPSの構造


メモリ
PC
次PC計算
mux
レジスタ選択
32x32ビット
レジスタ
命令デコーダ
制御回路
mux
演算選択
32ビットALU
アドレス（32ビット）
データ（8, 16, 32ビット）

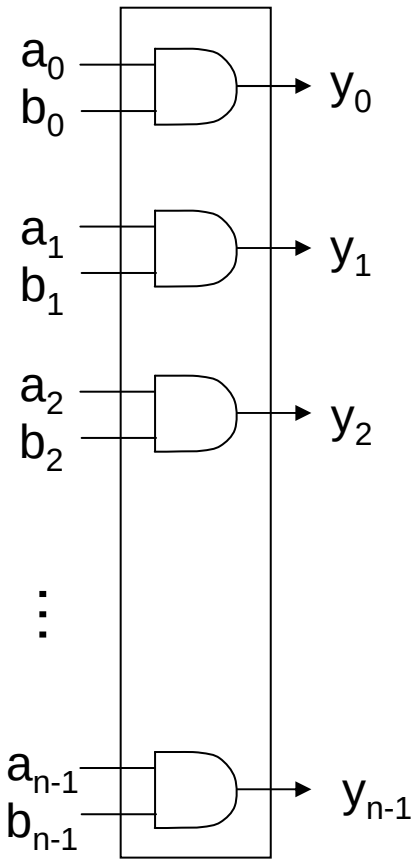
ALU



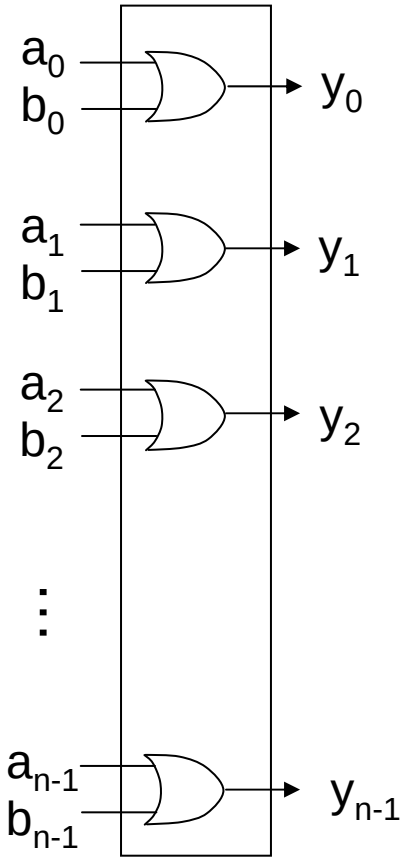


- 短い斜線と数字は，複数ビットをまとめたことを表示している（自明な場合，興味のない場合は適宜省略）

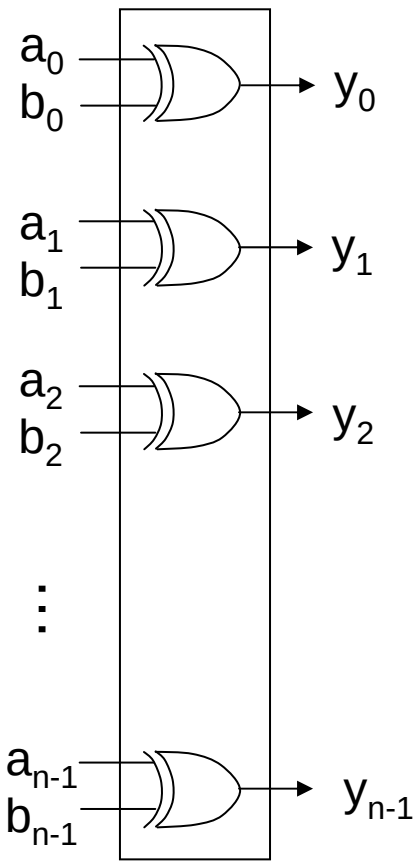
ビットごと論理演算器



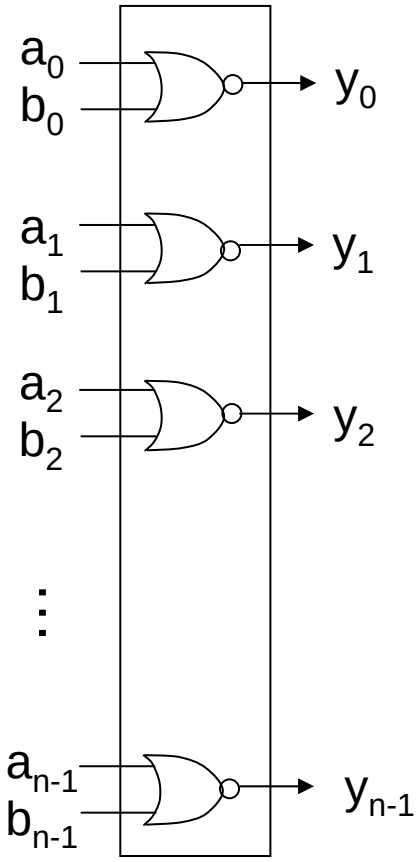
y = a & b



y = a | b

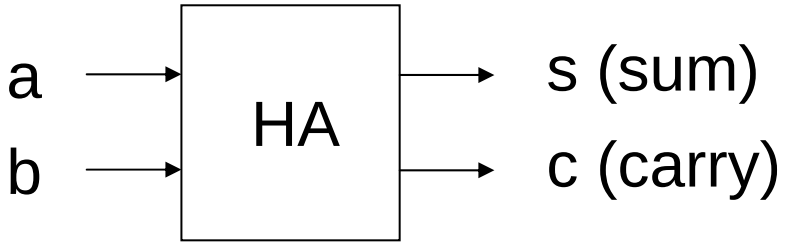


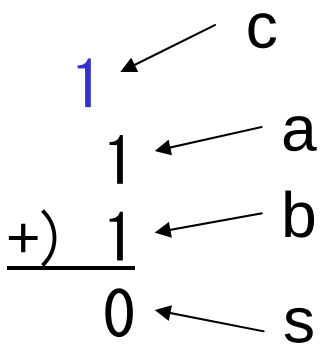
y = a ^ b



y = ~(a | b)

半加算器 (half adder)

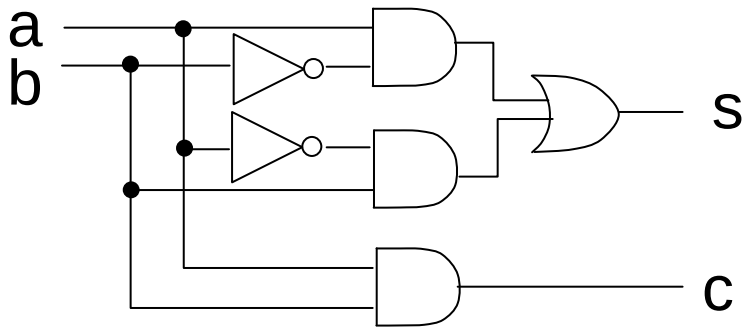




a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s = \bar{a}b + a\bar{b} \ (= a \oplus b)$$

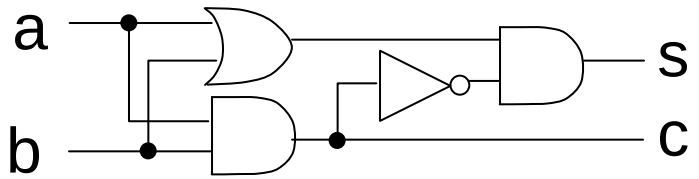
$$c = ab$$



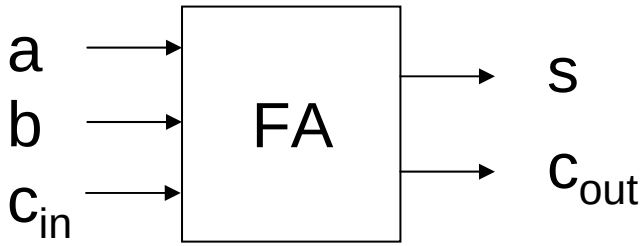
（参考）

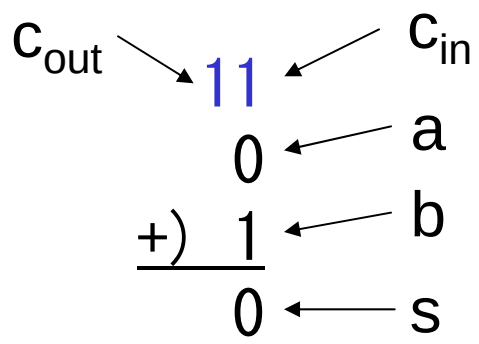
あるいは，s の計算に c の計算結果を利用することを考えると

$$
\begin{aligned}
s &= \overline{a}b + a\overline{b} \\
&= (\overline{a} + a\overline{b})(b + a\overline{b}) && \text{分配則} \\
&= (\overline{a} + a)(\overline{a} + \overline{b})(b + a)(b + \overline{b}) && \text{分配則} \\
&= (\overline{a} + \overline{b})(b + a) && \text{相補則} \\
&= \overline{ab}(a + b) && \text{ド・モルガン則} \\
&= \overline{c}(a + b)
\end{aligned}
$$

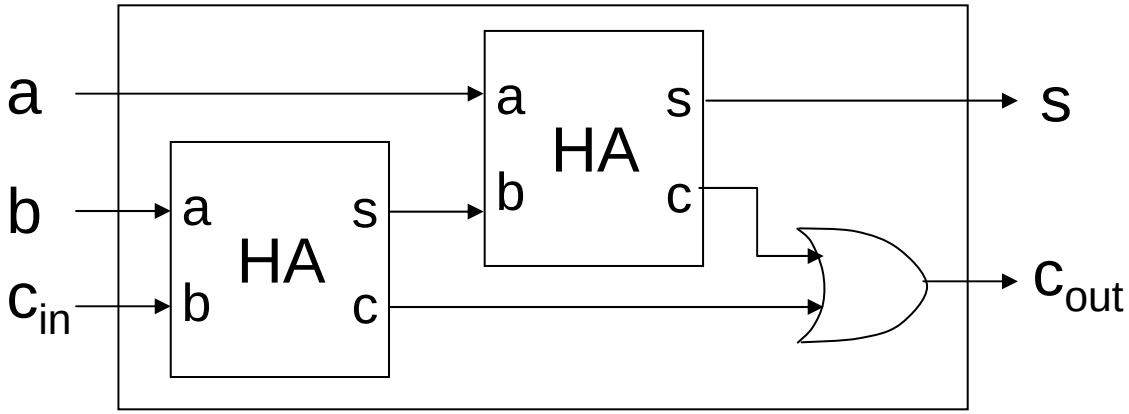


全加算器 (full adder)





前の位からの繰り上がりを考慮する．半加算器が2つ必要

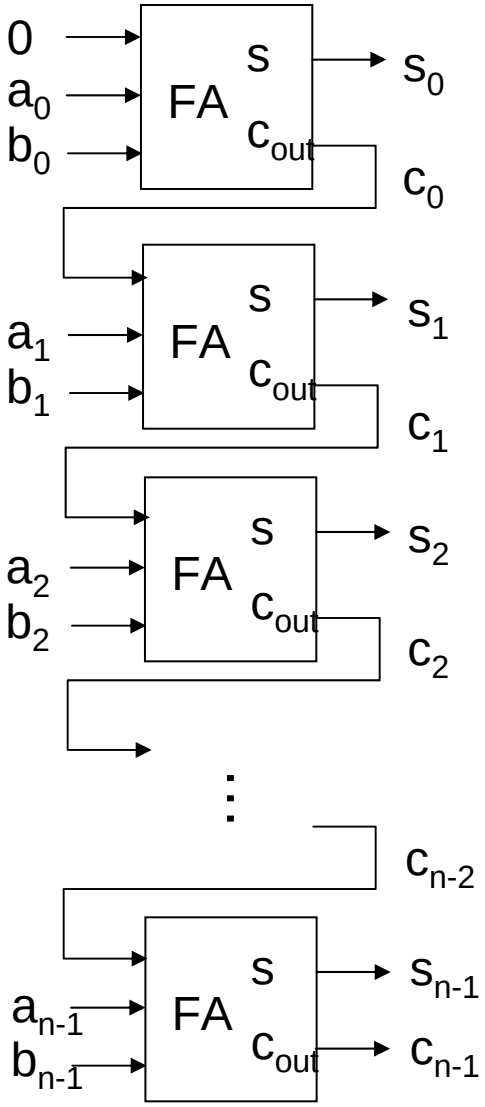


a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$
\begin{aligned}
s &= a \oplus b \oplus c_{\mathsf{in}} \\
&= \overline{a}\,\overline{b}\,c_{\mathsf{in}} + \overline{a}\,b\,\overline{c_{\mathsf{in}}} + a\,\overline{b}\,\overline{c_{\mathsf{in}}} + abc_{\mathsf{in}} \\
c_{\mathsf{out}} &= bc_{\mathsf{in}} + (b \oplus c_{\mathsf{in}})a \\
&= (\overline{a} + a)bc_{\mathsf{in}} + (b\,\overline{c_{\mathsf{in}}} + \overline{b}c_{\mathsf{in}})a \\
&= \overline{a}bc_{\mathsf{in}} + ab\,\overline{c_{\mathsf{in}}} + a\,\overline{b}c_{\mathsf{in}} + abc_{\mathsf{in}} \\
&= ab + bc_{\mathsf{in}} + c_{\mathsf{in}}a
\end{aligned}
$$

- 3つの入力が対称な点に注意
- c_{out} は3入力多数決関数

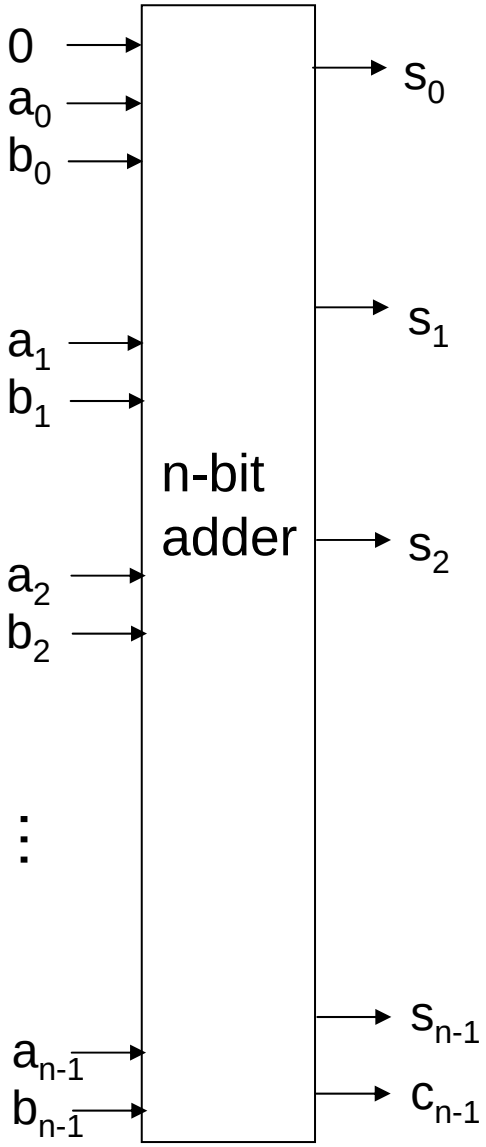
n-ビット加算器



リプルキャリー型加算器と呼ばれる

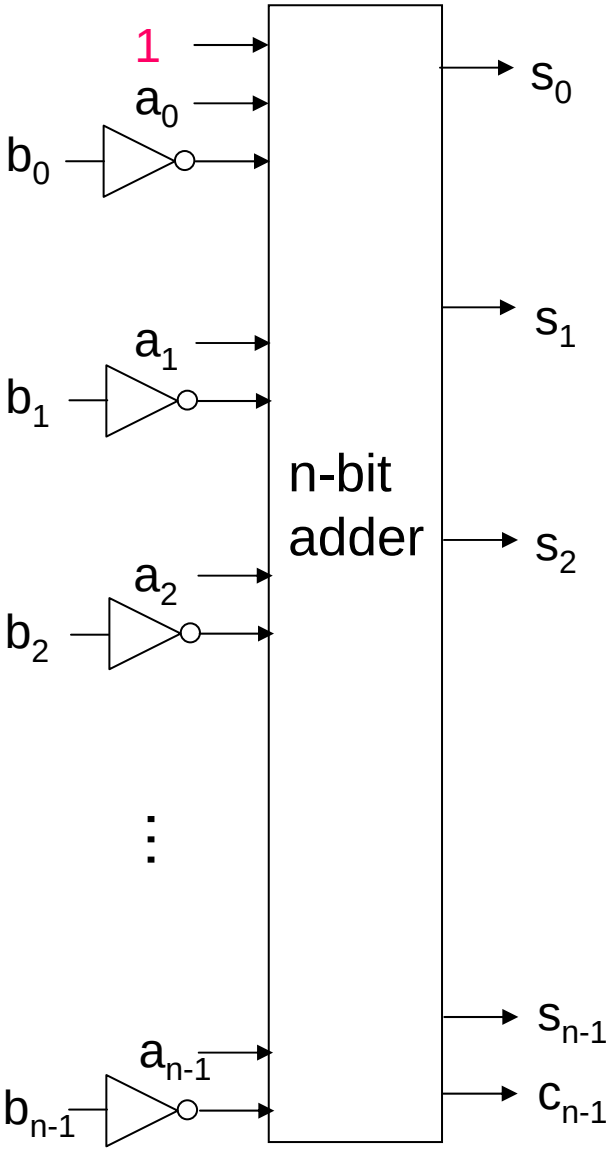
- nに比例して遅延が蓄積するため，決して速い回路ではない
- より高速な（しかし回路規模の大きい）加算回路も広く用いられている（e.g. キャリー先読み型加算器）

n-ビット減算器

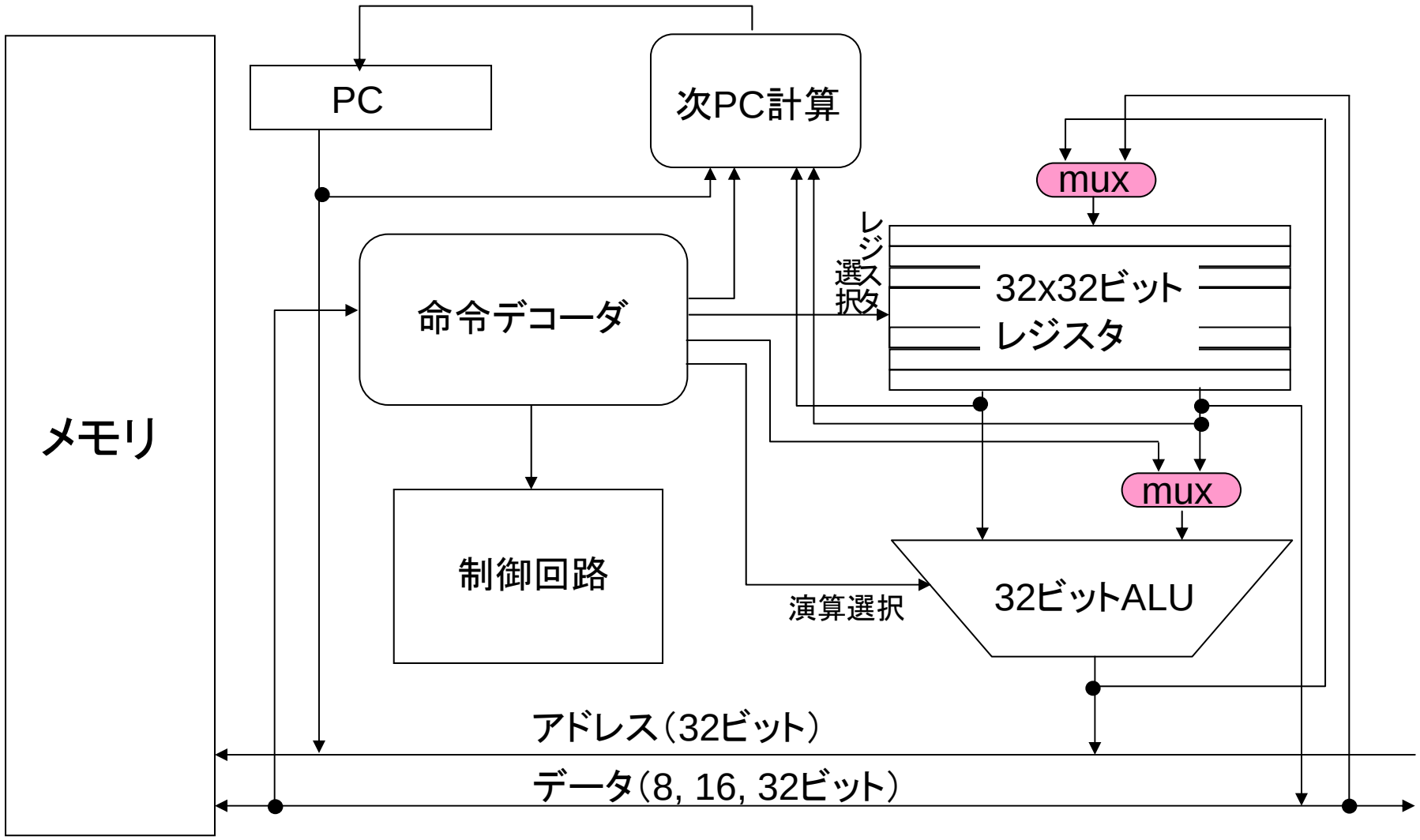


$$a - b = a + (-b)$$

入力を変えるだけで
減算器になる



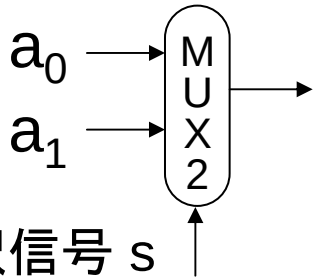
復習: MIPSの構造


メモリ
PC
次PC計算
mux
レジスタ選択
32x32ビット
レジスタ
命令デコーダ
制御回路
mux
演算選択
32ビットALU
アドレス(32ビット)
データ(8, 16, 32ビット)

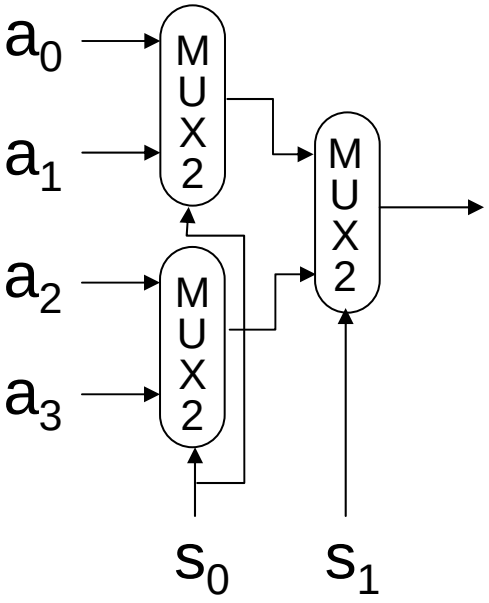
マルチプレクサ (セレクタ)

s = i なら a_i を選ぶ

2入力

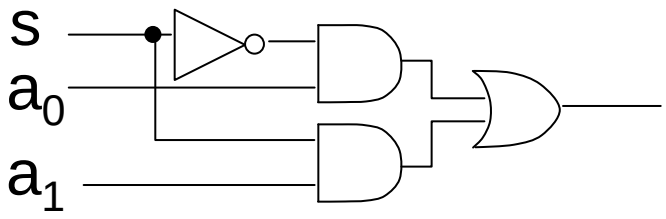


4入力

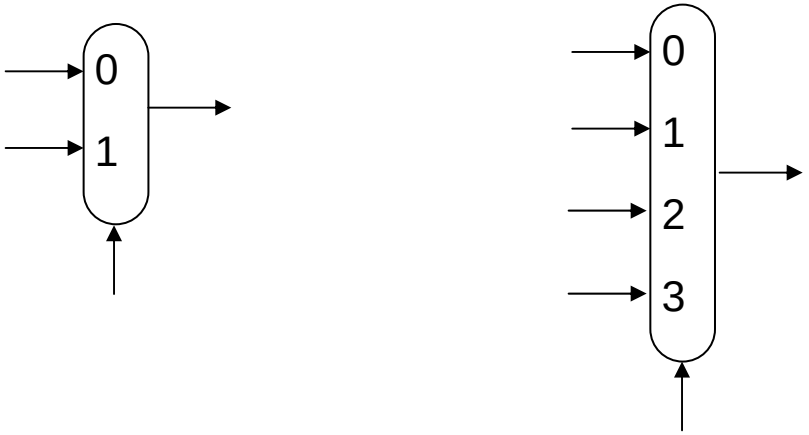


a_0	a_1	s	mux2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

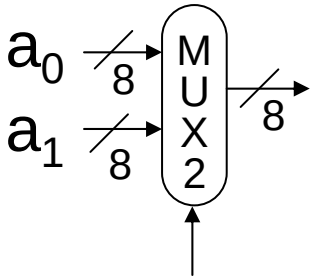
$m = a_0 \overline{s} + a_1 s$



マルチプレクサのバリエーション

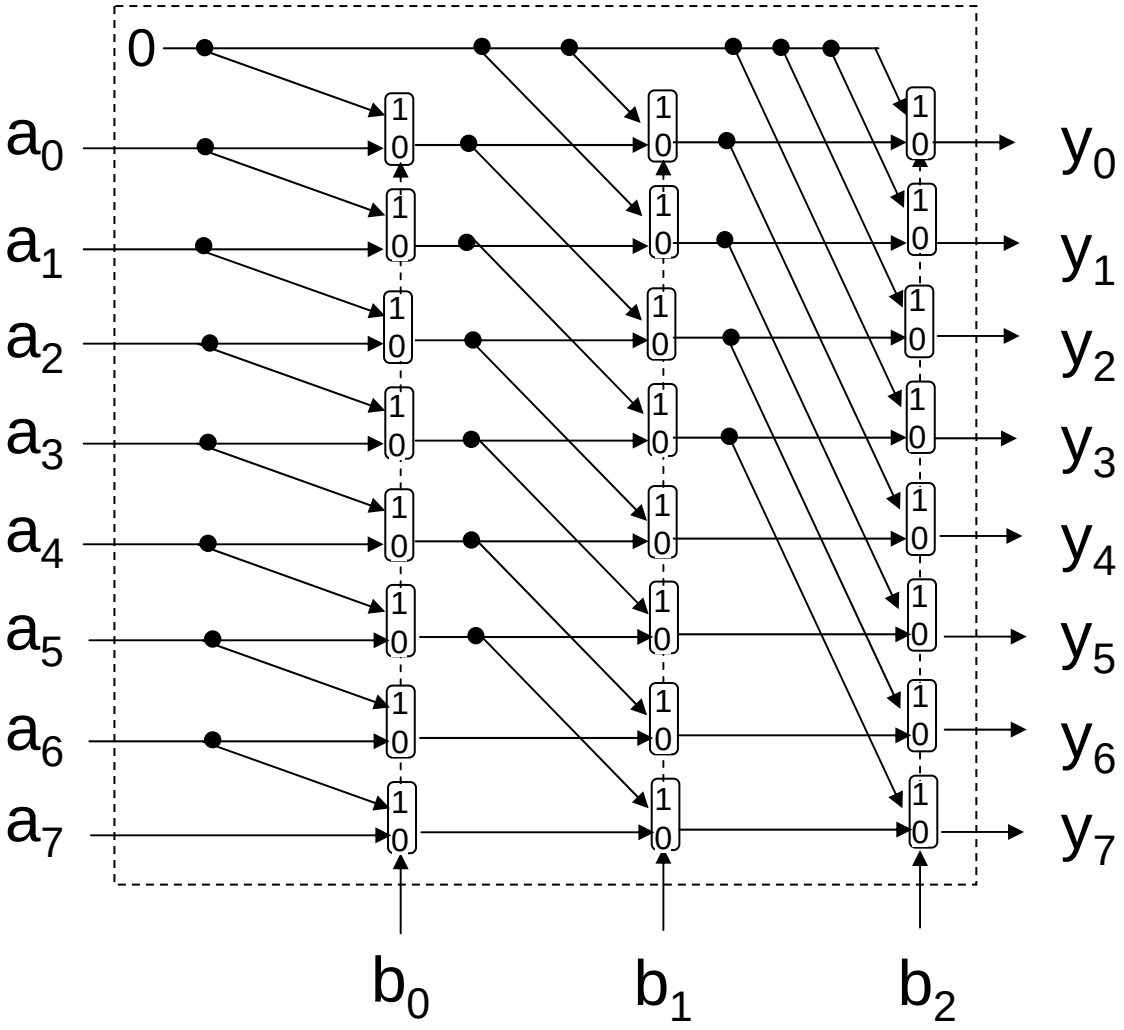


どの制御信号によってどの入力が選択されるかを明示したい場合



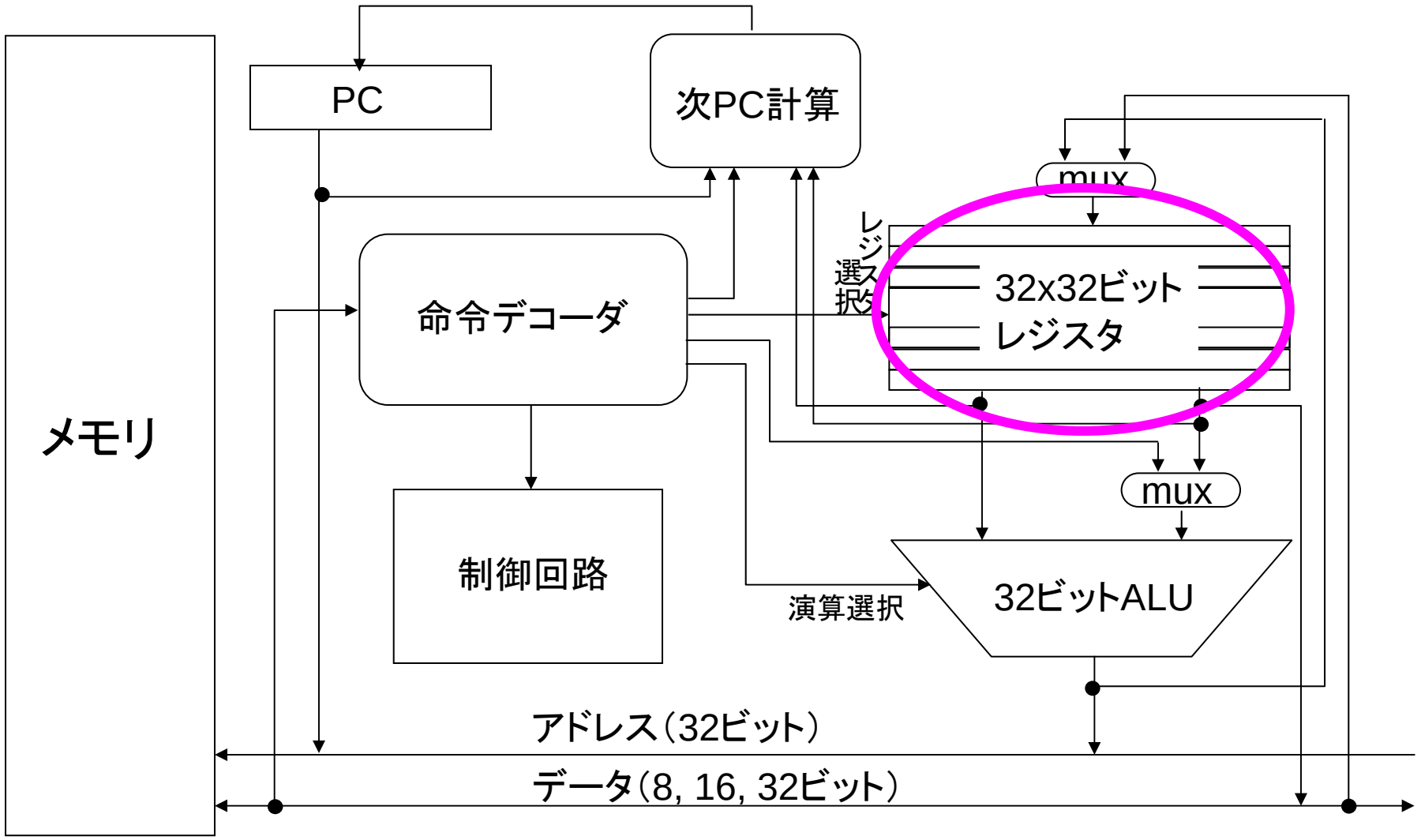
多ビットをまとめて選択したい場合

バレルシフタ

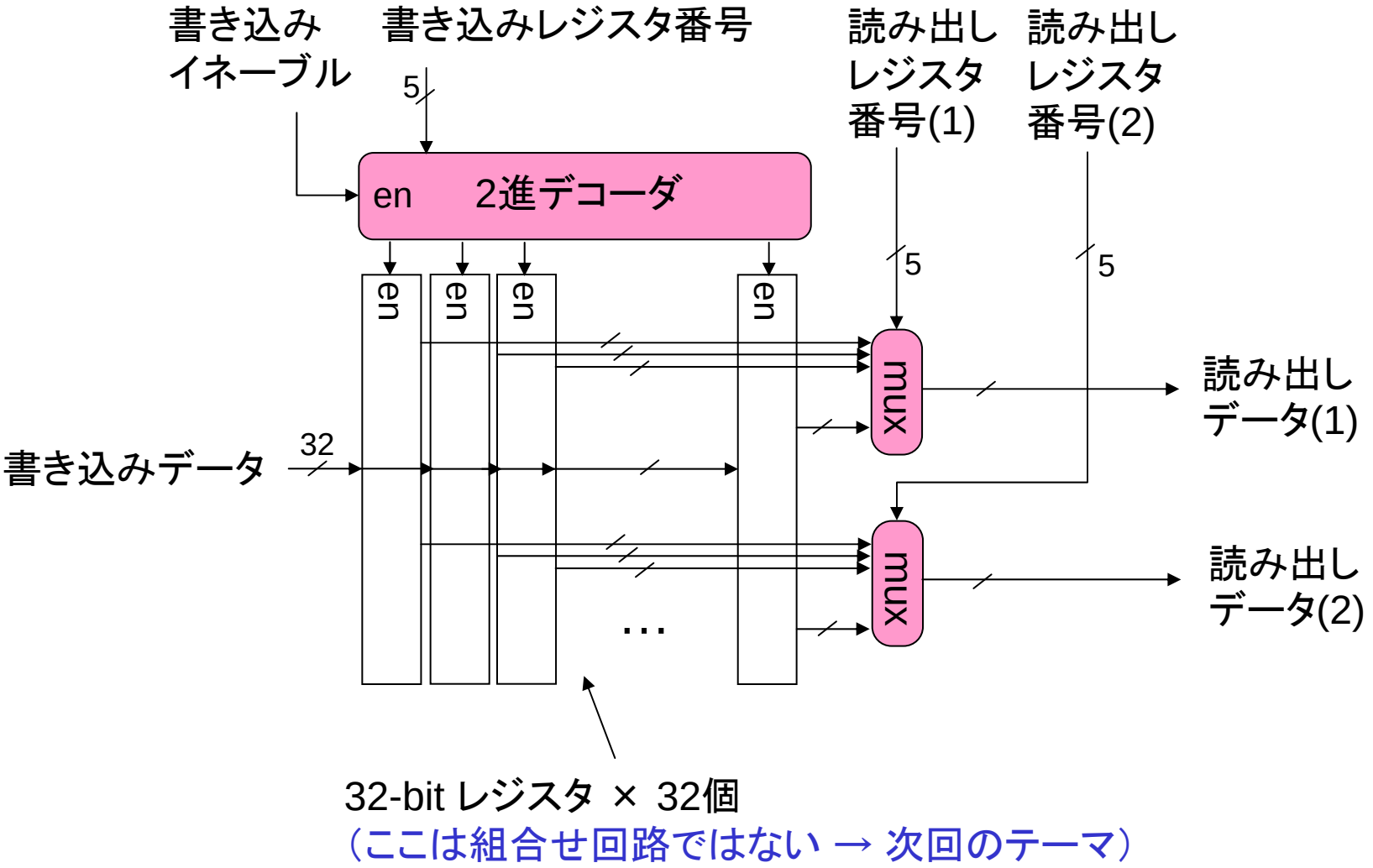


8ビットの左シフタ
y = a << b

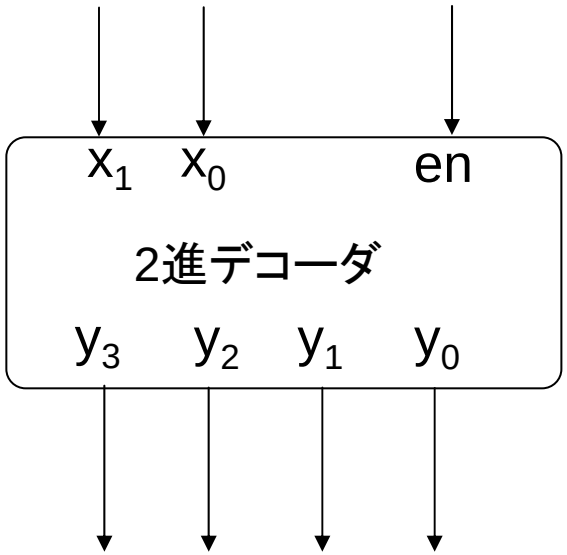
復習: MIPSの構造


メモリ
PC
次PC計算
命令デコーダ
制御回路
mux
レジスタ選択
32x32ビット
レジスタ
mux
演算選択
32ビットALU
アドレス(32ビット)
データ(8, 16, 32ビット)

32×32ビットレジスタ (1入力2出力)



2進デコーダ



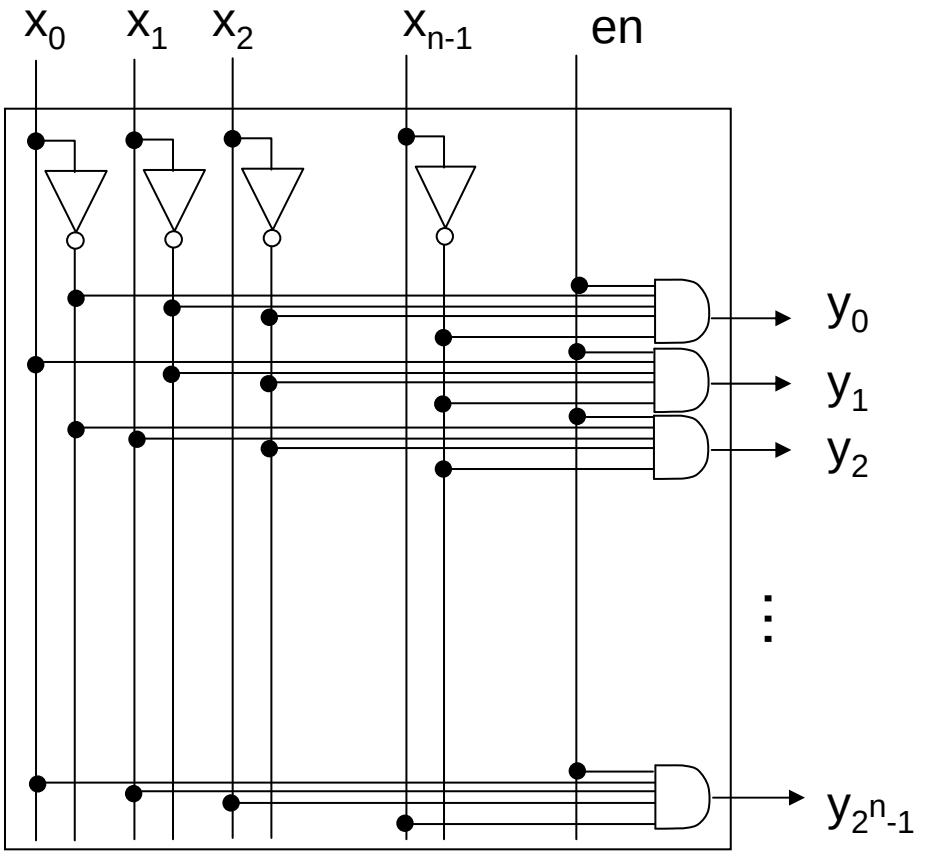
- en = 0 のときは全出力が0
- en = 1 のときは，入力を2進数 k と見なして，出力 y_k を1，他を0とする

  例: $x_1 = 1, x_0 = 0$ のとき，入力は2進数で「2」を表すので，y_2 のみが1となる

(en は enable の略で，活性化信号などと呼ばれる)

各 y_i について真理値表を書くと，1行だけ出力が1になるような表となる

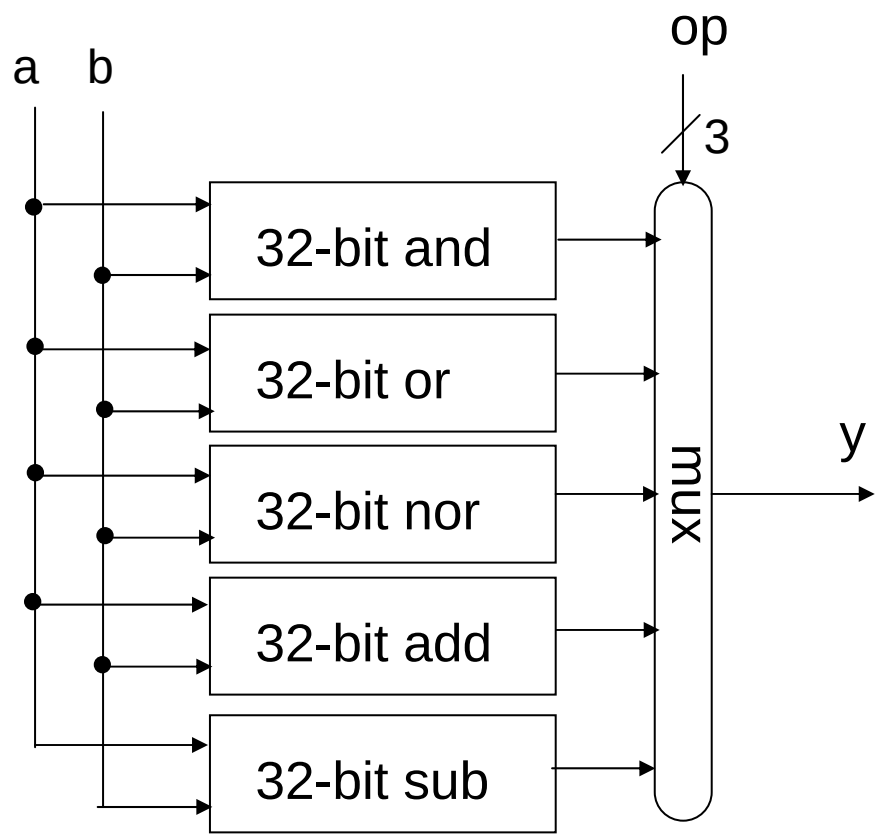
x_2	x_1	x_0	$y_7y_6\cdots y_1y_0$
0	0	0	0000 0001
0	0	1	0000 0010
0	1	0	0000 0100
0	1	1	0000 1000
1	0	0	0001 0000
1	0	1	0010 0000
1	1	0	0100 0000
1	1	1	1000 0000



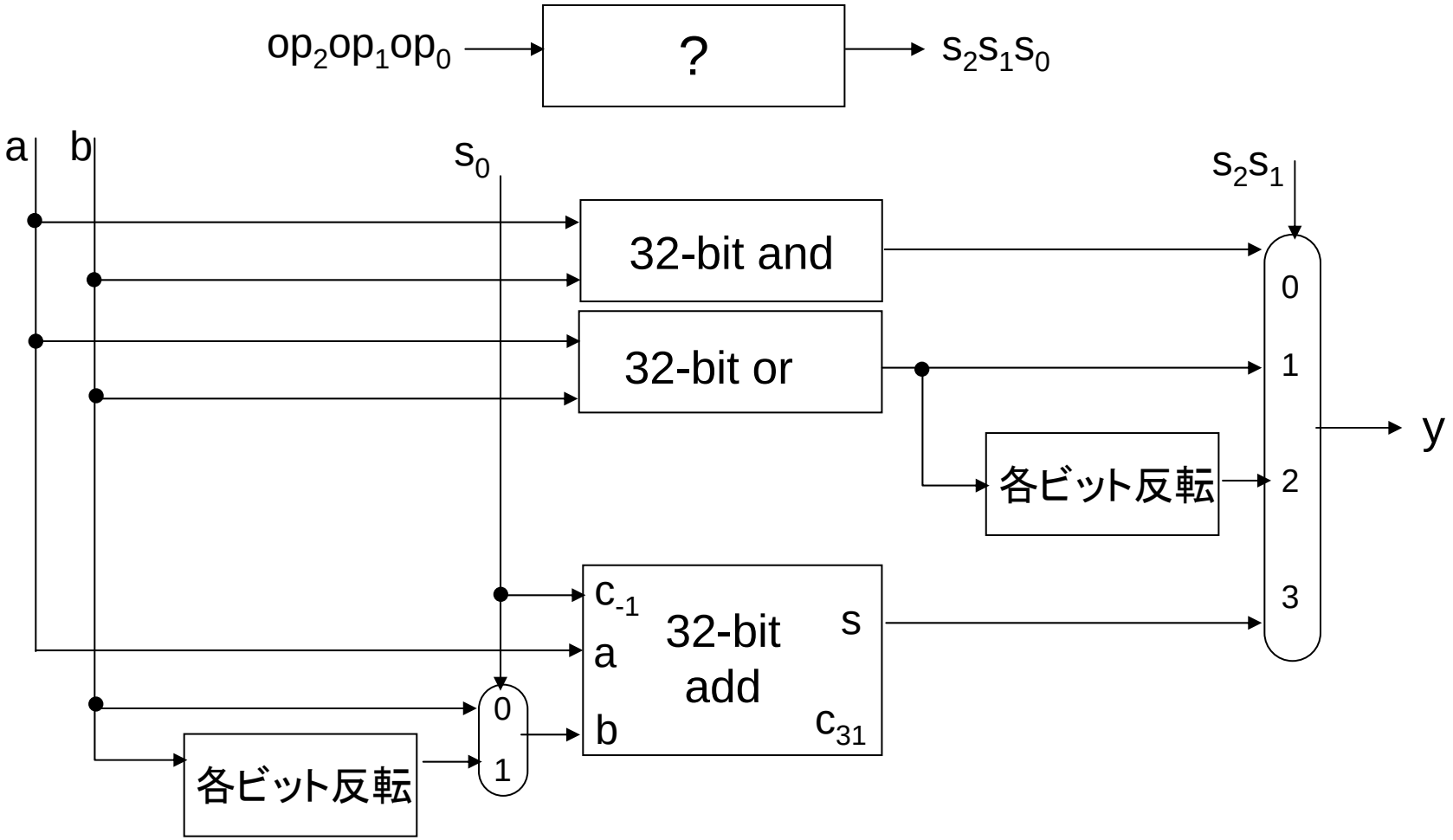
ALU の設計例

3ビット入力 op_2 op_1 op_0 によって以下の各演算を洗濯して実行する ALU を設計する

001: addu
011: subu
100: and
101: or
111: nor



ALU回路の簡単化



- 加算と減算の回路を共有できることは既に学んだとおり
- nor は or の出力を反転すればよい

練習問題

op_2, op_1, op_0 の3 ビットの入力から，信号 s_2, s_1, s_0 をそれぞれ生成する 3 つの組合せ回路を設計したい．これらのそれぞれをカルノー図で表し，最も簡単な積和型論理回路で実現せよ．ただし，s_0, s_1, s_0 のそれぞれを独立に簡単化してよい．

ヒント:

- s_0 は，op が addu と subu のとき以外は don’t care
- s_2 s_1 は組にして，2進数とみなして 0, 1, 2, 3 になる場合の op をそれぞれ考える．使われない op の場合は don’t care

解答例

- $op_2\ op_1\ op_0$ が001 のとき(addu) は s_0 を0 とし，011 のとき (subu) はs_0 を1 とする．その他の場合は don’t care

- $op_2\ op_1\ op_0$ が
  - 100 のとき: $s_2s_1 = 00$
  - 101 のとき: $s_2s_1 = 01$
  - 111 のとき: $s_2s_1 = 10$
  - 001 or 011 のとき: $s_2s_1 = 11$
  - それら以外のとき: don’t care

S_0

op_2 \ $op_1\ op_0$	00	01	11	10
0	*	0	1	*
1	*	*	*	*

S_1

op_2 \ $op_1\ op_0$	00	01	11	10
0	*	1	1	*
1	0	1	0	*

S_2

op_2 \ $op_1\ op_0$	00	01	11	10
0	*	1	1	*
1	0	0	1	*

解答例(つづき)

$$s_0 = \mathrm{op}_1$$
$$s_1 = \overline{\mathrm{op}_2} + \overline{\mathrm{op}_1} \cdot \mathrm{op}_0$$
$$s_2 = \overline{\mathrm{op}_2} + \mathrm{op}_1$$