論理回路入門（９）

マルチプレクサと簡単な ALU の設計

李　亜民

2022 年 11 月 22 日 (火)

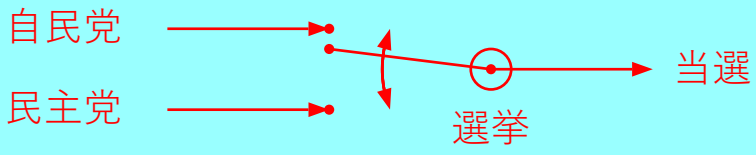
マルチプレクサ

ポイント

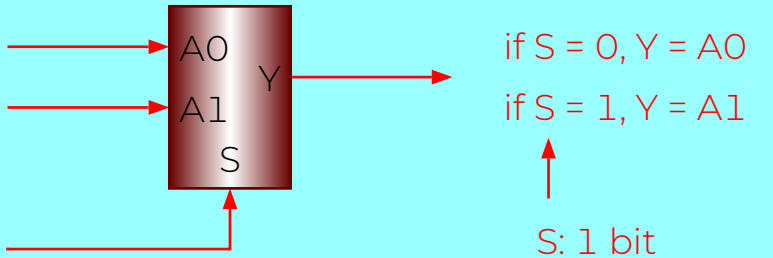
- マルチプレクサ
- 組み合わせ回路設計の手順
- 1 ビット 2-to-1 のマルチプレクサ
- 4 ビット 2-to-1 のマルチプレクサ
- 1 ビット 4-to-1 のマルチプレクサ
- 4 ビット 4-to-1 のマルチプレクサ
- バレルシフタ (Barrel shifter) 回路
- 7 セグメント LED
- ALU: Arithmetic Logic Unit (算術論理演算回路)

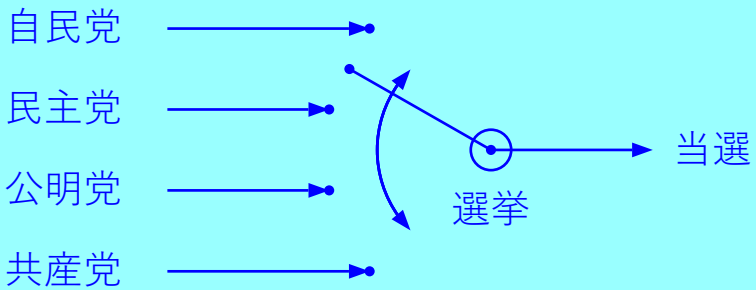
マルチプレクサ

マルチプレクサ (Multiplexer) は、複数の入力から１つを選択し出力する。 (マルチプレクサ = データ選択器)

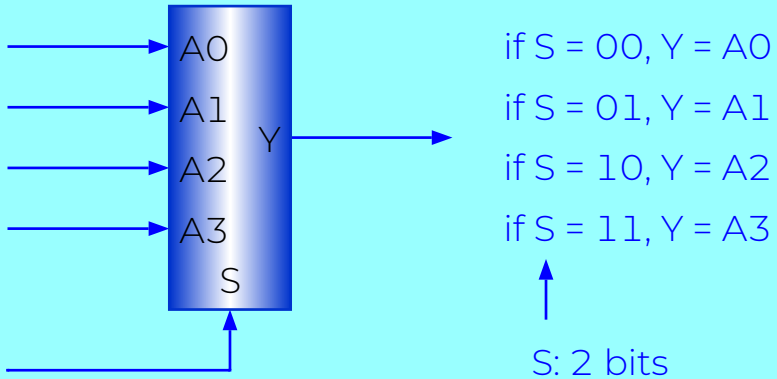


2-to-1 マルチプレクサ

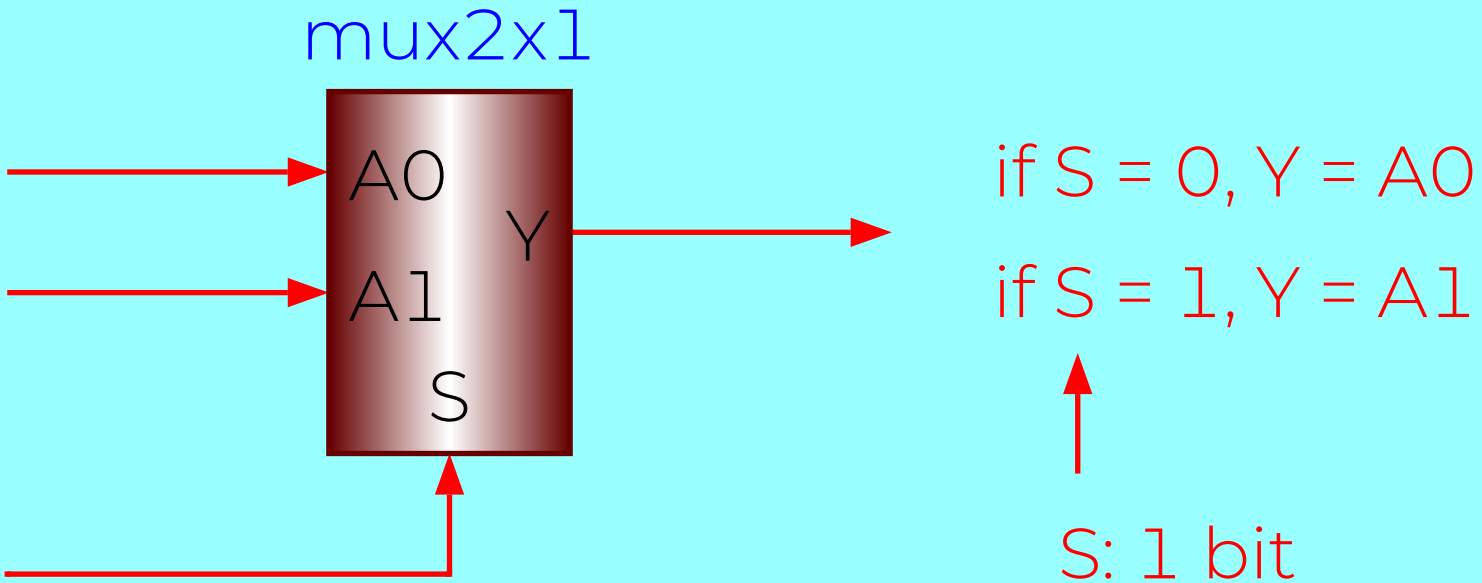




4-to-1 マルチプレクサ



2-to-1 のマルチプレクサを設計



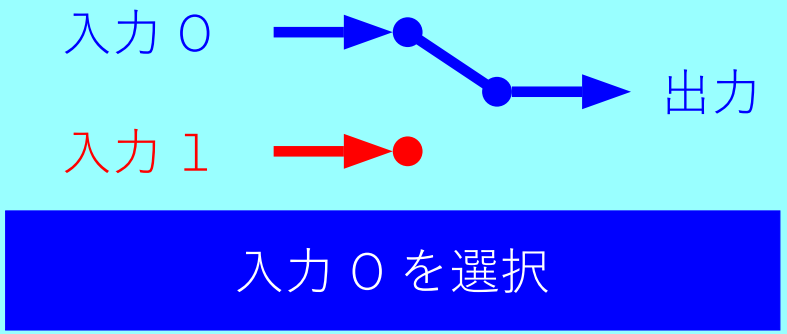
1 ビット 2-to-1 のマルチプレクサ回路を設計しましょう。

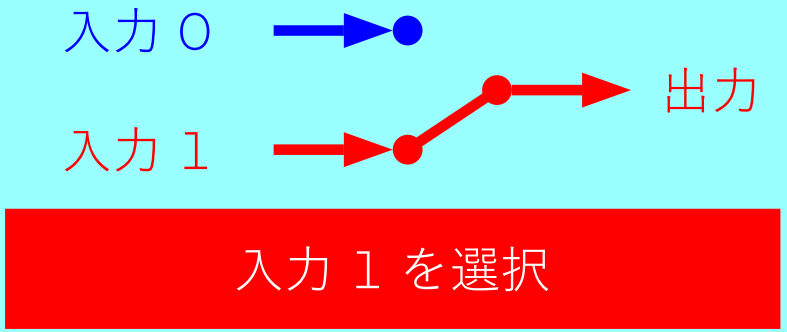
組み合わせ回路設計の手順

1. 問題を理解する。
2. 入力と出力信号の名前を任意に決める。
3. 入力のあらゆる組み合わせを列挙し，真理値表をつくる。
4. 真理値表から論理式をつくる。
   (どの組み合わせで出力が 1 になるか。)
5. カルノー図を用いて論理式を簡単化する。
6. 論理式から回路をつくる。
7. テストベンチをつくる (シミュレーションするため)。
8. 回路をシミュレーションする (回路設計の正当性検証)。
9. 与えられた回路の動作を理解する (波形の説明)。

マルチプレクサ

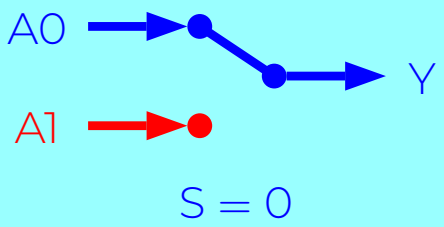
1. 問題: 二つの入力 (それぞれの入力は 1 ビット) から一つを選択して出力するマルチプレクサを設計せよ。

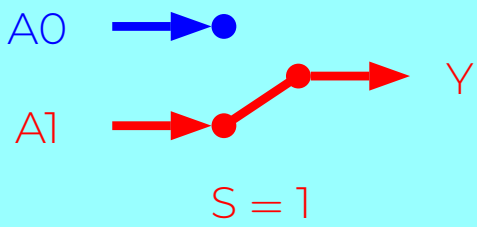




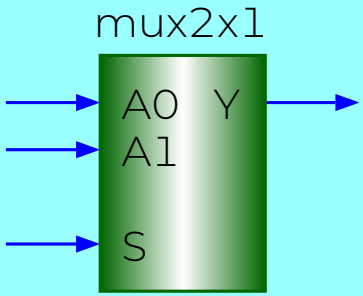
マルチプレクサ — 信号

2. 入力と出力信号の名前を任意に決める。例えば
   - 二つの入力信号: A0 と A1
   - 一つの出力信号: Y
   - どっちを選択するかを告げる信号 (入力): S
     - ★ if (S == 0) Y = A0; ・・・・・・・・S は 0 のとき、A0 を選択する
     - ★ else Y = A1; ・・・・・・・・・・・・・・・S は 1 のとき、A1 を選択する





   - 右側の図はマルチプレクサのシンボルですが、どのような回路で Y は出力されているでしょうか？



マルチプレクサ — 真理値表

3. 真理値表をつくる。

入力は 3 つ、全ての組み合わせは 8 つ ($2^3 = 8$)

入力			出力	
S	A1	A0	Y	
0	0	0	0	
0	0	1	**1**	S は 0 のとき、
0	1	0	0	A0 を選択する
0	1	1	**1**	
1	0	0	0	
1	0	1	0	S は 1 のとき、
1	1	0	**1**	A1 を選択する
1	1	1	**1**	

マルチプレクサ — 論理式

4. 論理式をつくる (どの組み合わせで出力が 1 になるか)。

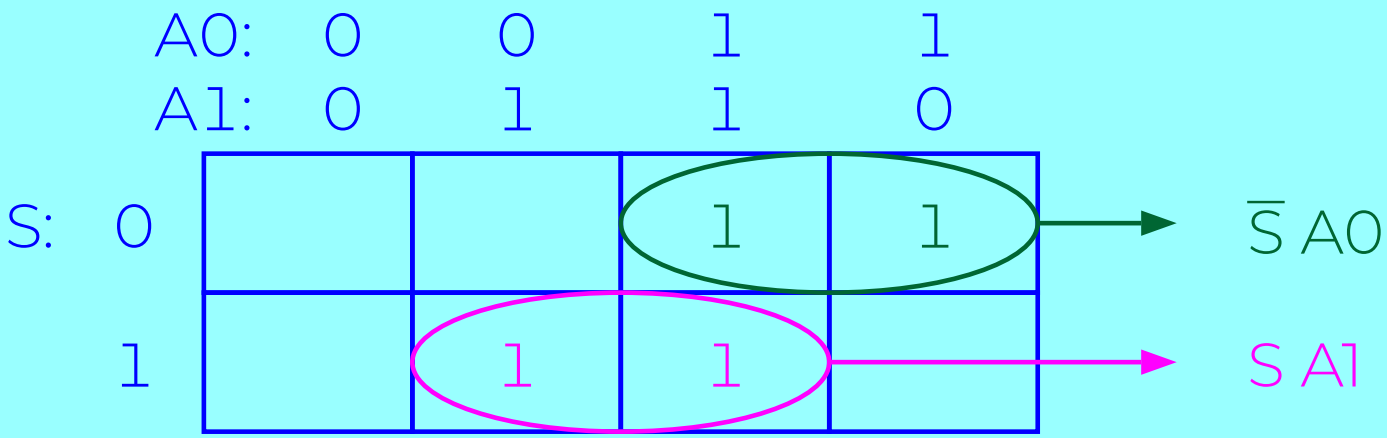
入力			出力	
S	A1	A0	Y	
0	0	0	0	
0	0	1	**1**	$\leftarrow \overline{S}\,\overline{A1}\,A0$
0	1	0	0	
0	1	1	**1**	$\leftarrow \overline{S}\,A1\,A0$
1	0	0	0	
1	0	1	0	
1	1	0	**1**	$\leftarrow S\,A1\,\overline{A0}$
1	1	1	**1**	$\leftarrow S\,A1\,A0$

$Y = 1$:

$$Y = \overline{S}\,\overline{A1}\,A0 + \overline{S}\,A1\,A0 + S\,A1\,\overline{A0} + S\,A1\,A0$$

マルチプレクサ — 論理式の簡単化

5. カルノー図を用いて論理式を簡単化する:



$$Y = \overline{S}\,A0 + S\,A1$$

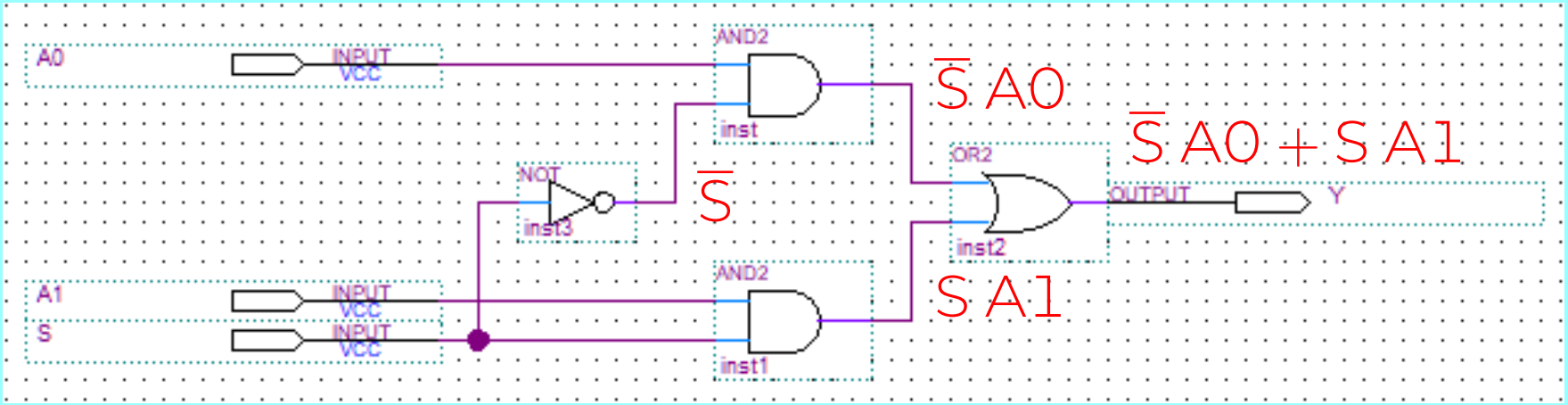
あるいはブール代数の定理により論理式を簡単化する:

$$\begin{aligned} Y &= \overline{S}\,\overline{A1}\,A0 + \overline{S}\,A1\,A0 + S\,A1\,\overline{A0} + S\,A1\,A0 \\ &= \overline{S}\,A0\,(\overline{A1} + A1) + S\,A1\,(\overline{A0} + A0) \\ &= \overline{S}\,A0 + S\,A1 \end{aligned}$$

マルチプレクサ — 回路

6. 論理式から回路をつくる。

$$Y = \overline{S}\,A0 + S\,A1$$



マルチプレクサ — テストベンチ

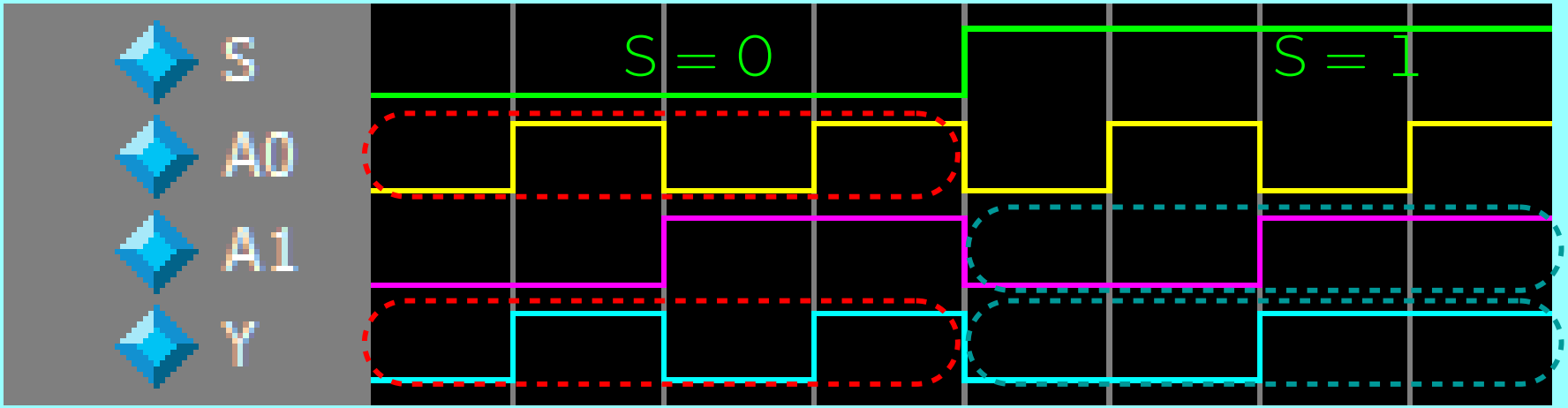
7. テストベンチをつくる (シミュレーションするため)。

```
‘timescale 1ns/1ps // unit = 1 ns; accuracy = 1 ps
module mux2x1_tb;  // test bench, no input / output pins
    reg  S,A0,A1;  // reg type for inputs of mux2x1
    wire Y;        // wire type for outputs of mux2x1
    mux2x1 i1 (.A0(A0), .A1(A1), .S(S), .Y(Y)); // invoke mux2x1
    initial begin  // input signal patterns          S A1 A0
        #0 S = 0; A1 = 0; A0 = 0; //         0 ns   0  0  0
        #1 S = 0; A1 = 0; A0 = 1; // 0 + 1 = 1 ns   0  0  1
        #1 S = 0; A1 = 1; A0 = 0; // 1 + 1 = 2 ns   0  1  0
        #1 S = 0; A1 = 1; A0 = 1; // 2 + 1 = 3 ns   0  1  1
        #1 S = 1; A1 = 0; A0 = 0; // 3 + 1 = 4 ns   1  0  0
        #1 S = 1; A1 = 0; A0 = 1; // 4 + 1 = 5 ns   1  0  1
        #1 S = 1; A1 = 1; A0 = 0; // 5 + 1 = 6 ns   1  1  0
        #1 S = 1; A1 = 1; A0 = 1; // 6 + 1 = 7 ns   1  1  1
        #1 $stop;   // stop simulation after 8 ns
    end
endmodule
```

マルチプレクサ — 波形

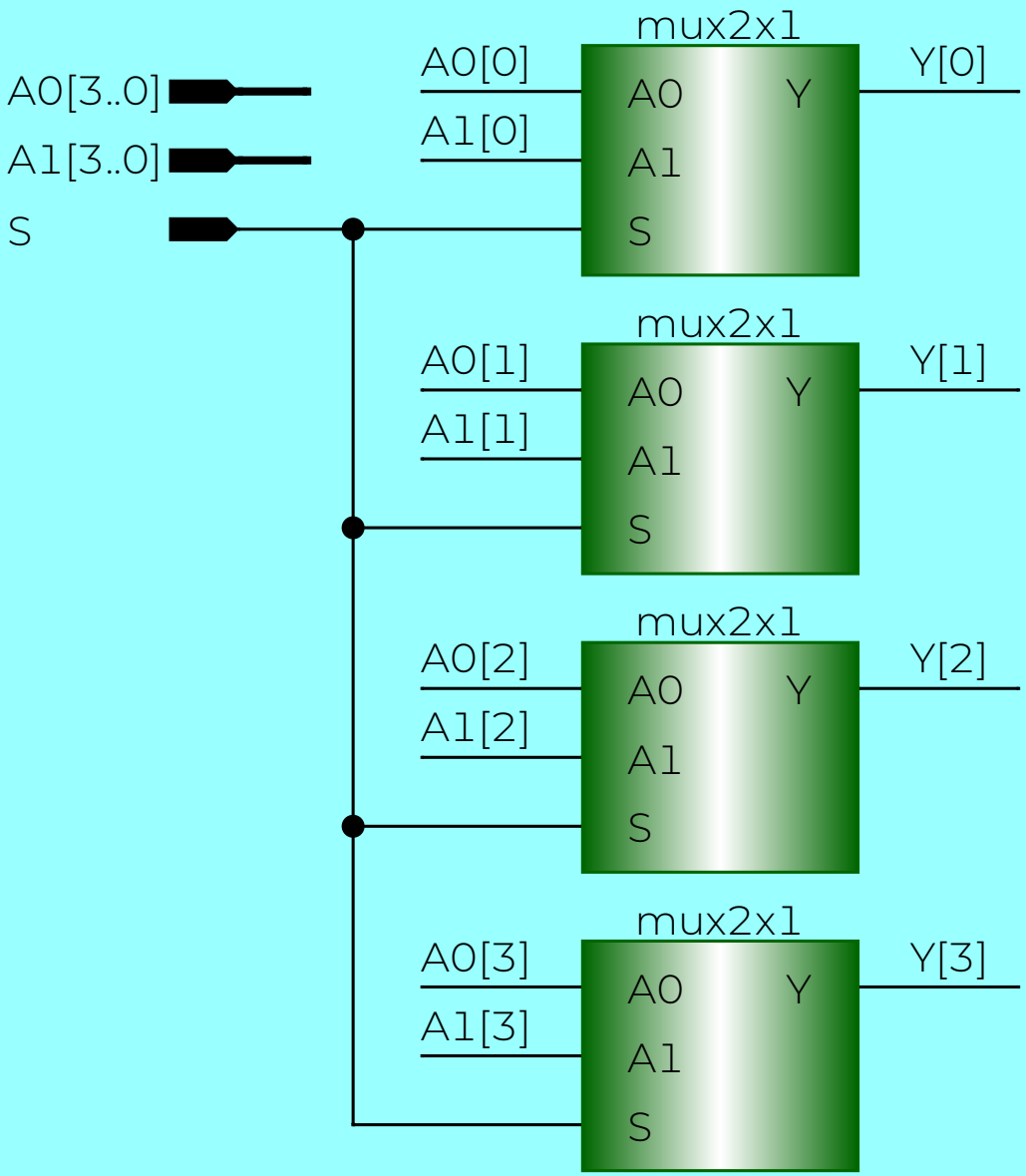
8. 回路をシミュレーションする。

$$Y = \overline{S}\,A0 + S\,A1$$



9. 理解: 入力 S = 0 のとき出力 Y = A0; 入力 S = 1 のとき出力 Y = A1

4 ビット 2-to-1 のマルチプレクサ



A0[3..0]: 4 ビット幅である

A0[3], A0[2], A0[1], A0[0]

のまとめ表示であること

A1[3..0]: 4 ビット幅である

A1[3], A1[2], A1[1], A1[0]

のまとめ表示であること

Y[3..0]: 4 ビット幅である

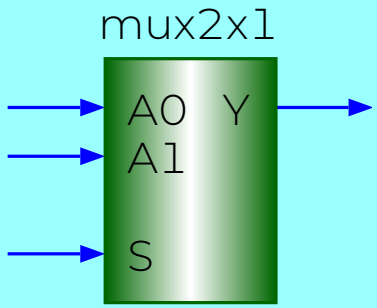
Y[3], Y[2], Y[1], Y[0]

のまとめ表示であること

4-to-1 のマルチプレクサ（１ビット）

- 2-to-1 のマルチプレクサ

$$Y = \overline{S} \cdot A0 + S \cdot A1$$

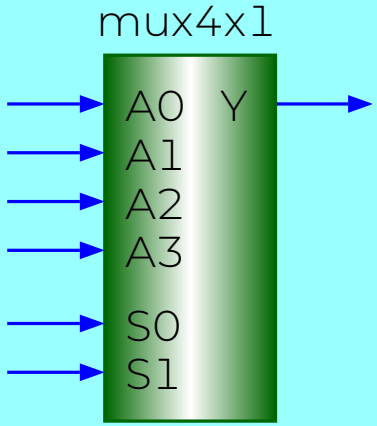


真理値表

S	Y
0	A0
1	A1

- 4-to-1 のマルチプレクサ

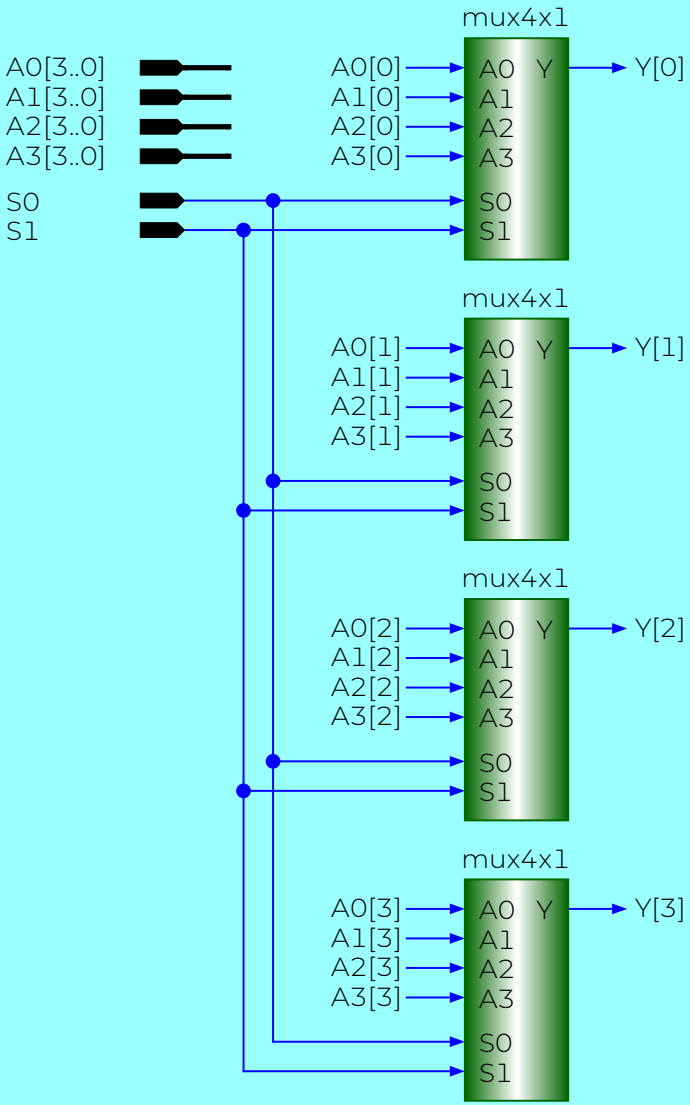
$$Y = \overline{S1} \cdot \overline{S0} \cdot A0 + \overline{S1} \cdot S0 \cdot A1 + S1 \cdot \overline{S0} \cdot A2 + S1 \cdot S0 \cdot A3$$

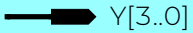


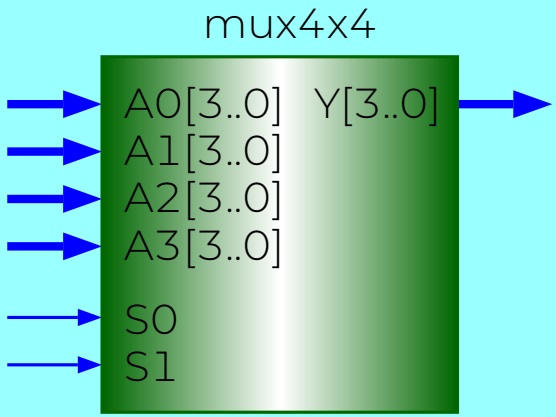
真理値表

S1	S0	Y
0	0	A0
0	1	A1
1	0	A2
1	1	A3

4-to-1 のマルチプレクサ（4 ビット）







シンボル

A0[3..0]: 4 ビット: A0[3], A0[2], A0[1], A0[0]

A1[3..0]: 4 ビット: A1[3], A1[2], A1[1], A1[0]

A2[3..0]: 4 ビット: A2[3], A2[2], A2[1], A2[0]

A3[3..0]: 4 ビット: A3[3], A3[2], A3[1], A3[0]

Y[3..0]: 4 ビット: Y[3], Y[2], Y[1], Y[0]

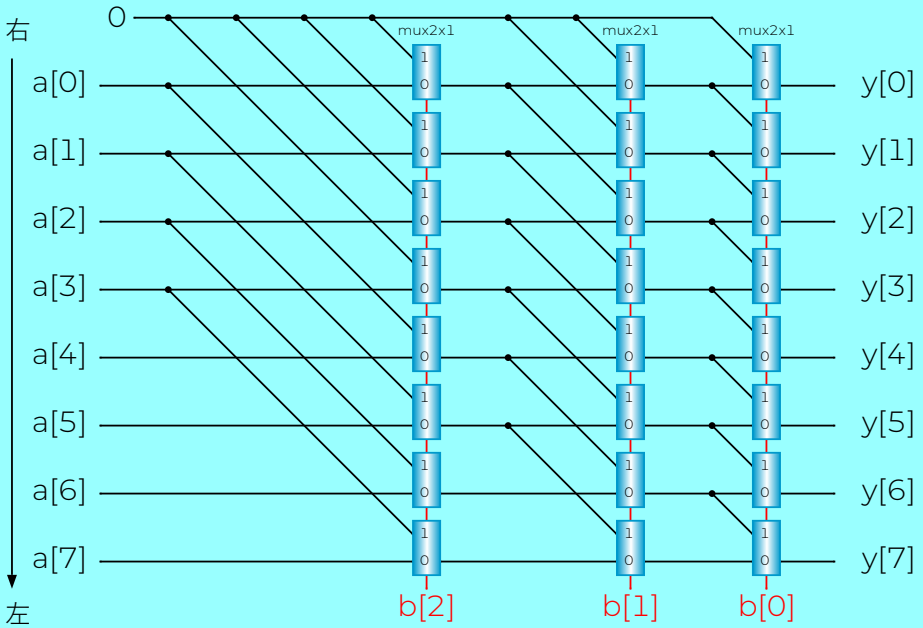
マルチプレクサの応用 — バレルシフタ

- SLL — Shift Left Logical (論理左シフト)
  - 0010 0011 4-bit shift left:
    0011 0000
  - 1010 0011 4-bit shift left:
    0011 0000
- SRL — Shift Right Logical (論理右シフト)
  - 0010 0011 4-bit shift right logical:
    0000 0010
  - 1010 0011 4-bit shift right logical:
    0000 1010
- SRA — Shift Right Arithmetic (算術右シフト)
  - 0010 0011 4-bit shift right arithmetic:
    0000 0010 (符号拡張)
  - 1010 0011 4-bit shift right arithmetic:
    1111 1010 (符号拡張)

マルチプレクサの応用 — バレルシフタ

論理左シフト (SLL)

01111110 → (0-bit shift) → 01111110 ·····(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 11111100 ·····(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 11111000 ·····(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 11110000 ·····(3=011: shift 0, 2, 1 bit)
01111110 → (4-bit shift) → 11100000 ·····(4=100: shift 4, 0, 0 bit)
01111110 → (5-bit shift) → 11000000 ·····(5=101: shift 4, 0, 1 bit)
01111110 → (6-bit shift) → 10000000 ·····(6=110: shift 4, 2, 0 bit)
01111110 → (7-bit shift) → 00000000 ·····(7=111: shift 4, 2, 1 bit)

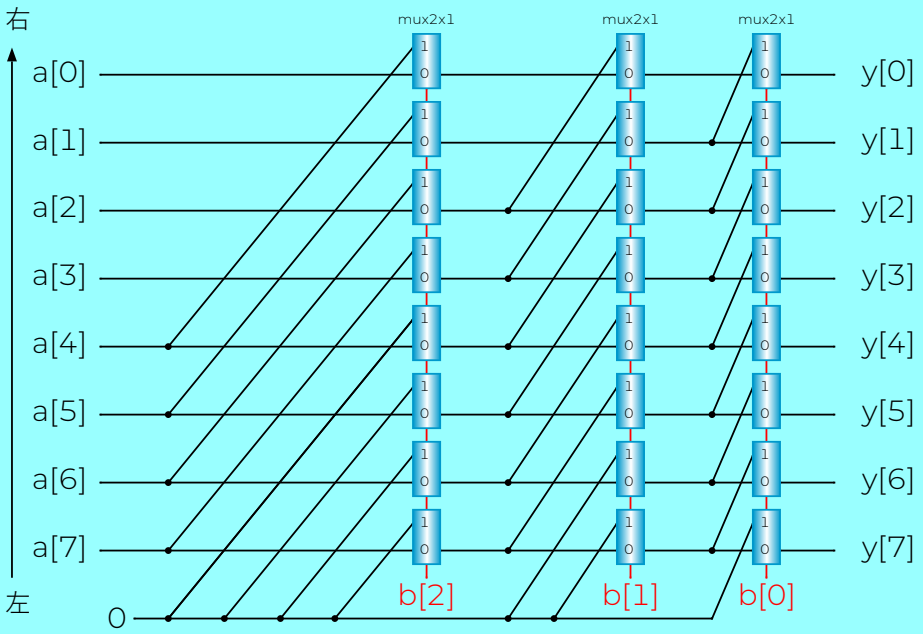


If b[2] = 1, shift 2^2 = 4 bits
If b[1] = 1, shift 2^1 = 2 bits
If b[0] = 1, shift 2^0 = 1 bit

マルチプレクサの応用 — バレルシフタ

論理右シフト (SRL)

01111110 → (0-bit shift) → 01111110 ·····(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 00111111 ·····(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 00011111 ·····(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 00001111 ·····(3=011: shift 0, 2, 1 bit)
10000000 → (4-bit shift) → 00001000 ·····(4=100: shift 4, 0, 0 bit)
10000000 → (5-bit shift) → 00000100 ·····(5=101: shift 4, 0, 1 bit)
10000000 → (6-bit shift) → 00000010 ·····(6=110: shift 4, 2, 0 bit)
10000000 → (7-bit shift) → 00000001 ·····(7=111: shift 4, 2, 1 bit)

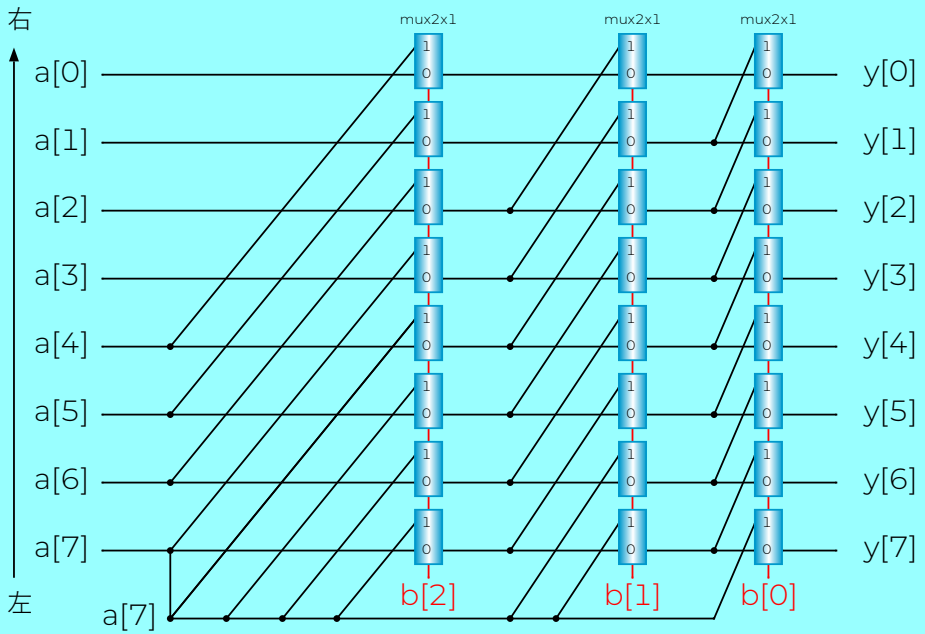


If b[2] = 1, shift 2^2 = 4 bits
If b[1] = 1, shift 2^1 = 2 bits
If b[0] = 1, shift 2^0 = 1 bit

マルチプレクサの応用 — バレルシフタ

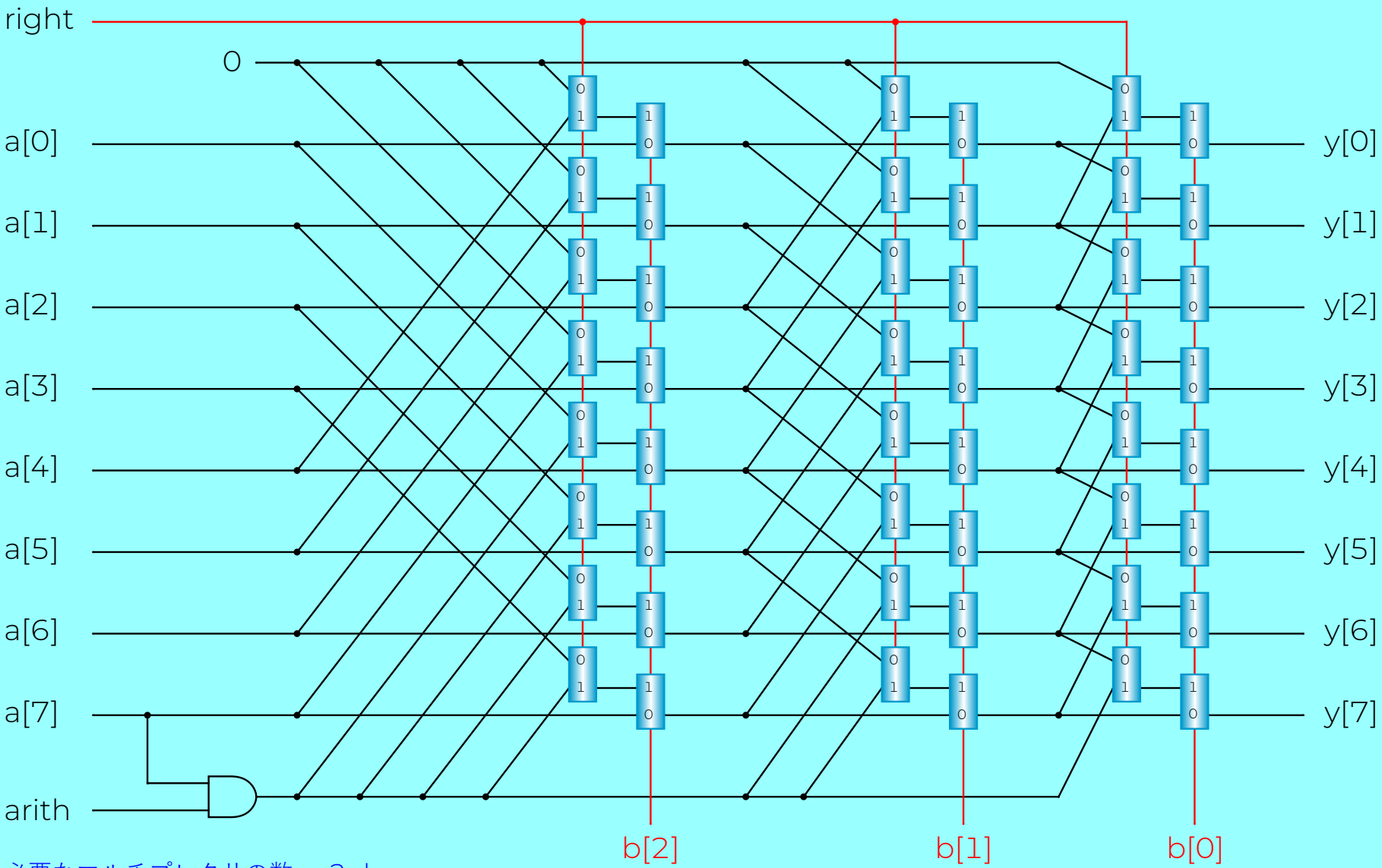
算術右シフト (SRA)

01111110 → (0-bit shift) → 01111110 ·····(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 00111111 ·····(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 00011111 ·····(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 00001111 ·····(3=011: shift 0, 2, 1 bit)
10000000 → (4-bit shift) → 11111000 ·····(4=100: shift 4, 0, 0 bit)
10000000 → (5-bit shift) → 11111100 ·····(5=101: shift 4, 0, 1 bit)
10000000 → (6-bit shift) → 11111110 ·····(6=110: shift 4, 2, 0 bit)
10000000 → (7-bit shift) → 11111111 ·····(7=111: shift 4, 2, 1 bit)



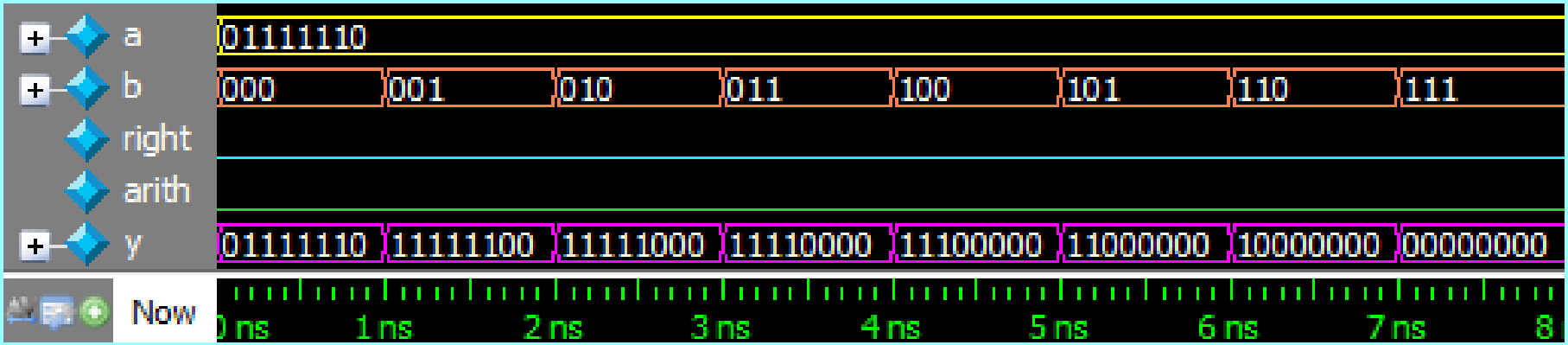
If $b[2] = 1$, shift $2^2 = 4$ bits
If $b[1] = 1$, shift $2^1 = 2$ bits
If $b[0] = 1$, shift $2^0 = 1$ bit

バレルシフタの回路



必要なマルチプレクサの数 $= 2n\log_2 n$

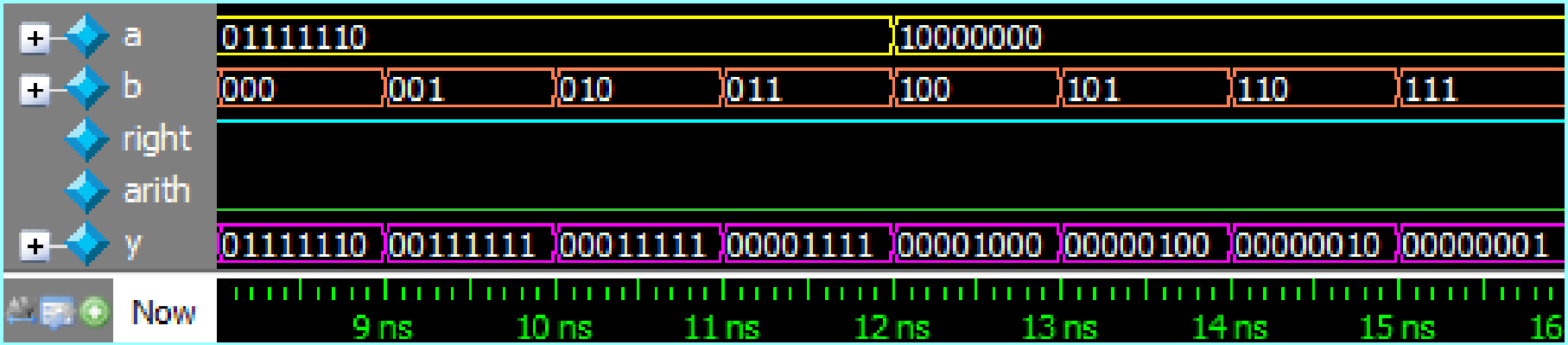
バレルシフタの波形



論理左シフト (arith = 0; right = 0) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・(SLL)

01111110 → (0-bit shift) → 01111110 ・・・・(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 11111100 ・・・・(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 11111000 ・・・・(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 11110000 ・・・・(3=011: shift 0, 2, 1 bit)
01111110 → (4-bit shift) → 11100000 ・・・・(4=100: shift 4, 0, 0 bit)
01111110 → (5-bit shift) → 11000000 ・・・・(5=101: shift 4, 0, 1 bit)
01111110 → (6-bit shift) → 10000000 ・・・・(6=110: shift 4, 2, 0 bit)
01111110 → (7-bit shift) → 00000000 ・・・・(7=111: shift 4, 2, 1 bit)

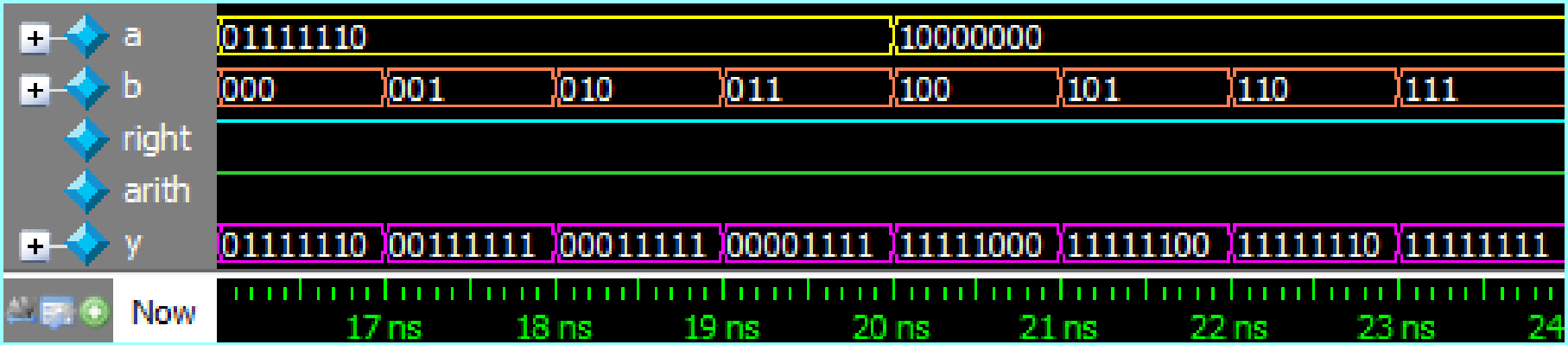
バレルシフタの波形



論理右シフト (arith = 0; right = 1) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・(SRL)

01111110 → (0-bit shift) → 01111110 ・・・・(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 00111111 ・・・・(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 00011111 ・・・・(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 00001111 ・・・・(3=011: shift 0, 2, 1 bit)
10000000 → (4-bit shift) → 00001000 ・・・・(4=100: shift 4, 0, 0 bit)
10000000 → (5-bit shift) → 00000100 ・・・・(5=101: shift 4, 0, 1 bit)
10000000 → (6-bit shift) → 00000010 ・・・・(6=110: shift 4, 2, 0 bit)
10000000 → (7-bit shift) → 00000001 ・・・・(7=111: shift 4, 2, 1 bit)

バレルシフタの波形



算術右シフト (arith = 1; right = 1) ・・・・・・・・・・・・・・・・・・・・・・・・・・・・(SRA)

01111110 → (0-bit shift) → 01111110 ・・・・(0=000: shift 0, 0, 0 bit)
01111110 → (1-bit shift) → 00111111 ・・・・(1=001: shift 0, 0, 1 bit)
01111110 → (2-bit shift) → 00011111 ・・・・(2=010: shift 0, 2, 0 bit)
01111110 → (3-bit shift) → 00001111 ・・・・(3=011: shift 0, 2, 1 bit)
10000000 → (4-bit shift) → 11111000 ・・・・(4=100: shift 4, 0, 0 bit)
10000000 → (5-bit shift) → 11111100 ・・・・(5=101: shift 4, 0, 1 bit)
10000000 → (6-bit shift) → 11111110 ・・・・(6=110: shift 4, 2, 0 bit)
10000000 → (7-bit shift) → 11111111 ・・・・(7=111: shift 4, 2, 1 bit)

マルチプレクサの応用 — RegFile

コンピューターシステム

1. コンピューター

(1) メモリ

(2) 入出力インターフェース

(3) CPU (プロセッサー)

ALU

· · ·

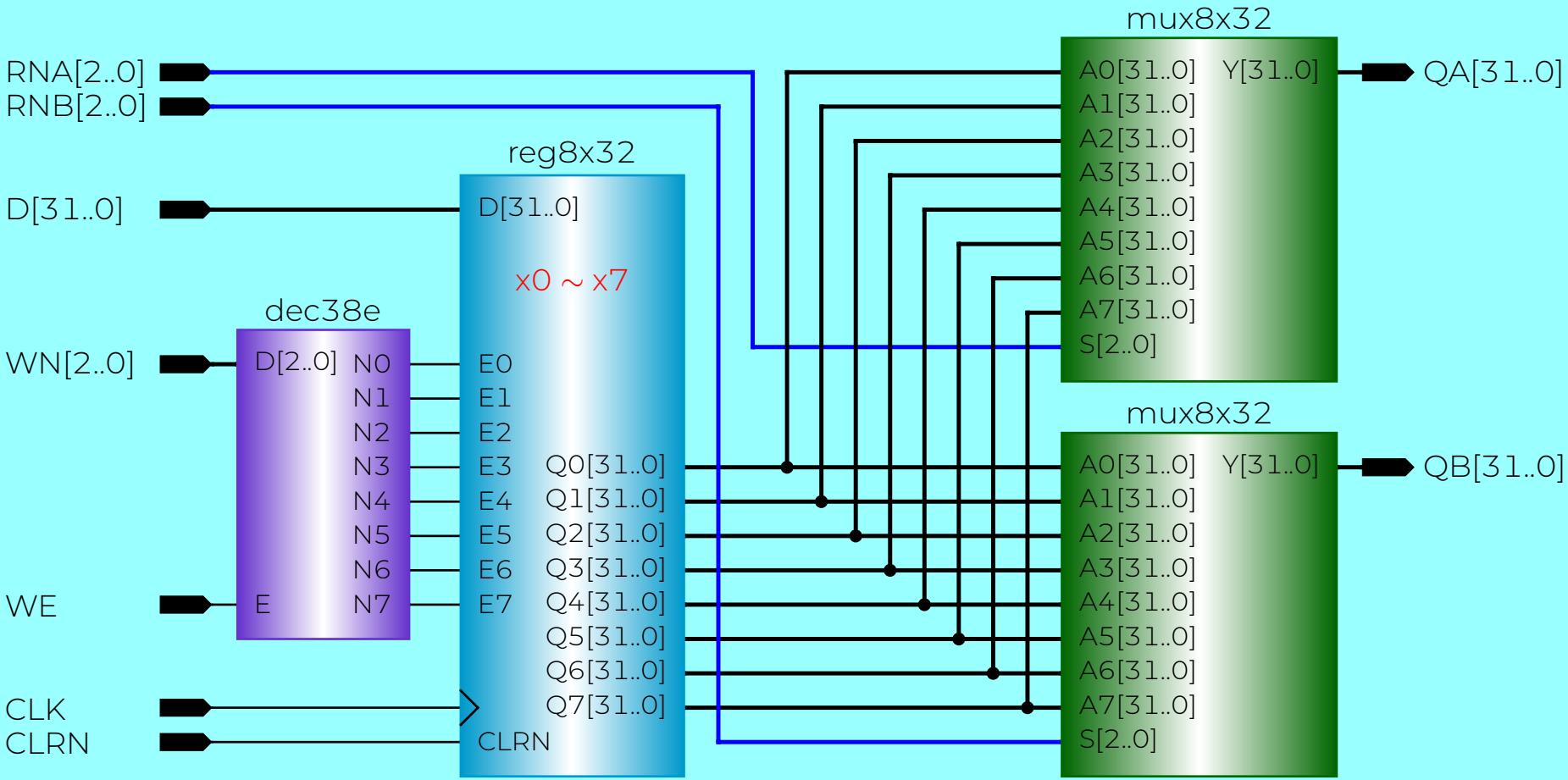
レジスタファイル

2. ソフトウェア
(OS やコンパイラなど)

3. 入出力デバイス
(キーボードやディスプレイなど)

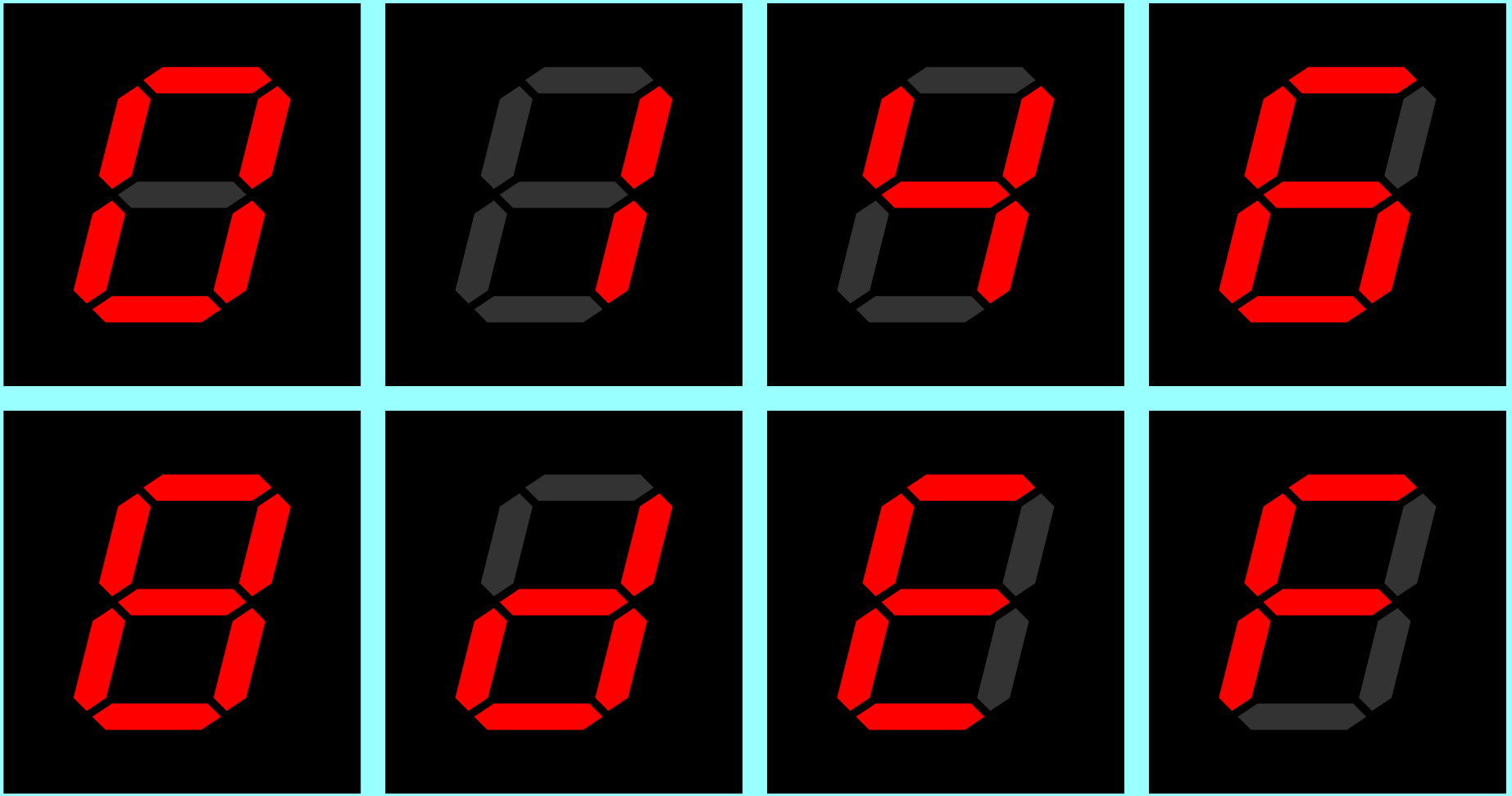
マルチプレクサの応用 — RegFile

8 × 32 ビットレジスタ・ファイル

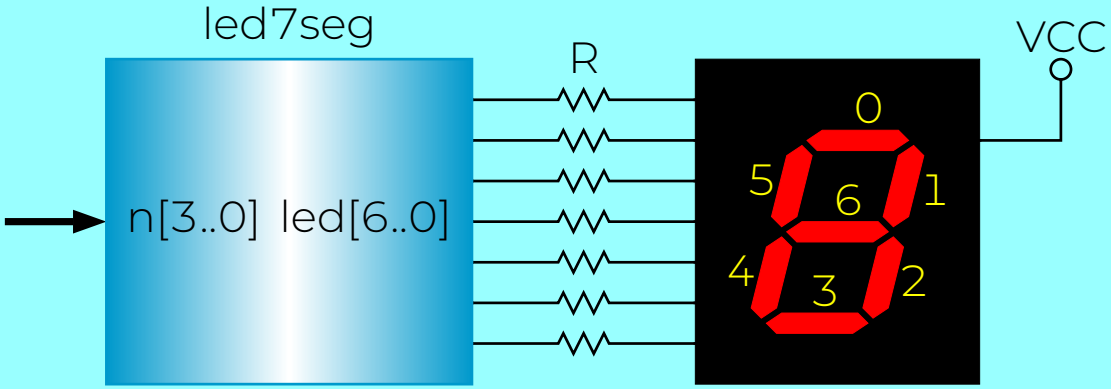


16 進数 7 セグメント LED 点灯回路

16 進数 7 セグメント LED 点灯の例



16 進数 7 セグメント LED 点灯回路



led[i] = 0: 点灯

led[i] = 1: 消灯

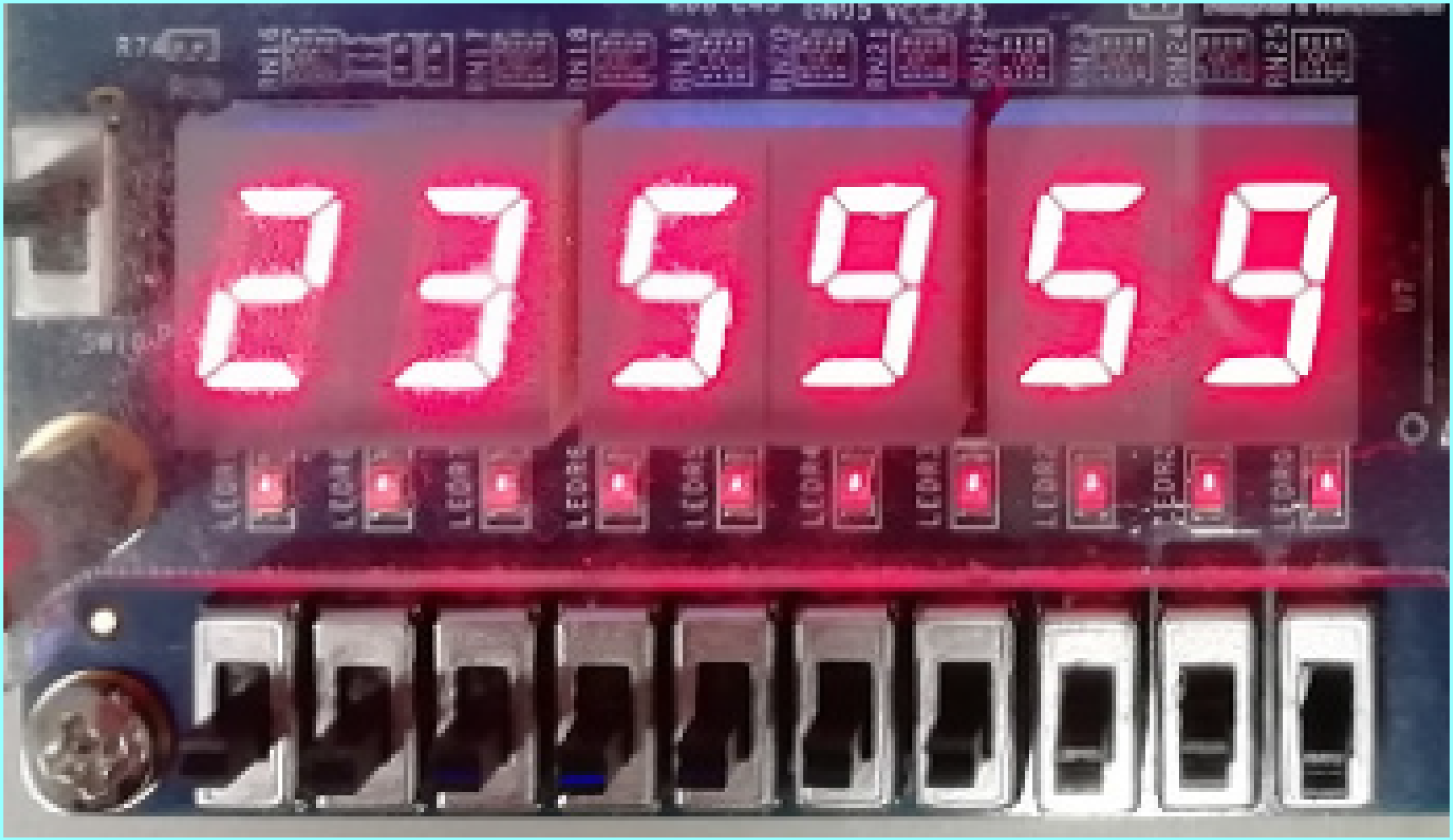
真理値表

入力 (スイッチ)				出力 (LED)						
n[3]	n[2]	n[1]	n[0]	led[6]	led[5]	led[4]	led[3]	led[2]	led[1]	led[0]
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	1
…	…	…	…	…	…	…	…	…	…	…
1	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	0	0	0	1	1	1	0

カルノー図を使用して、各 led 出力信号を簡単化する。

16 進数 7 セグメント LED 点灯回路

6 つの 7 セグメント LED と 10 個のスイッチ



コンピューターシステムとALUの関係

コンピューターシステム

1. コンピューター

(1) メモリ

(2) 入出力インターフェース

(3) CPU (プロセッサー)

ALU

· · ·

レジスタファイル

2. ソフトウェア
(OS やコンパイラなど)

3. 入出力デバイス
(キーボードやディスプレイなど)

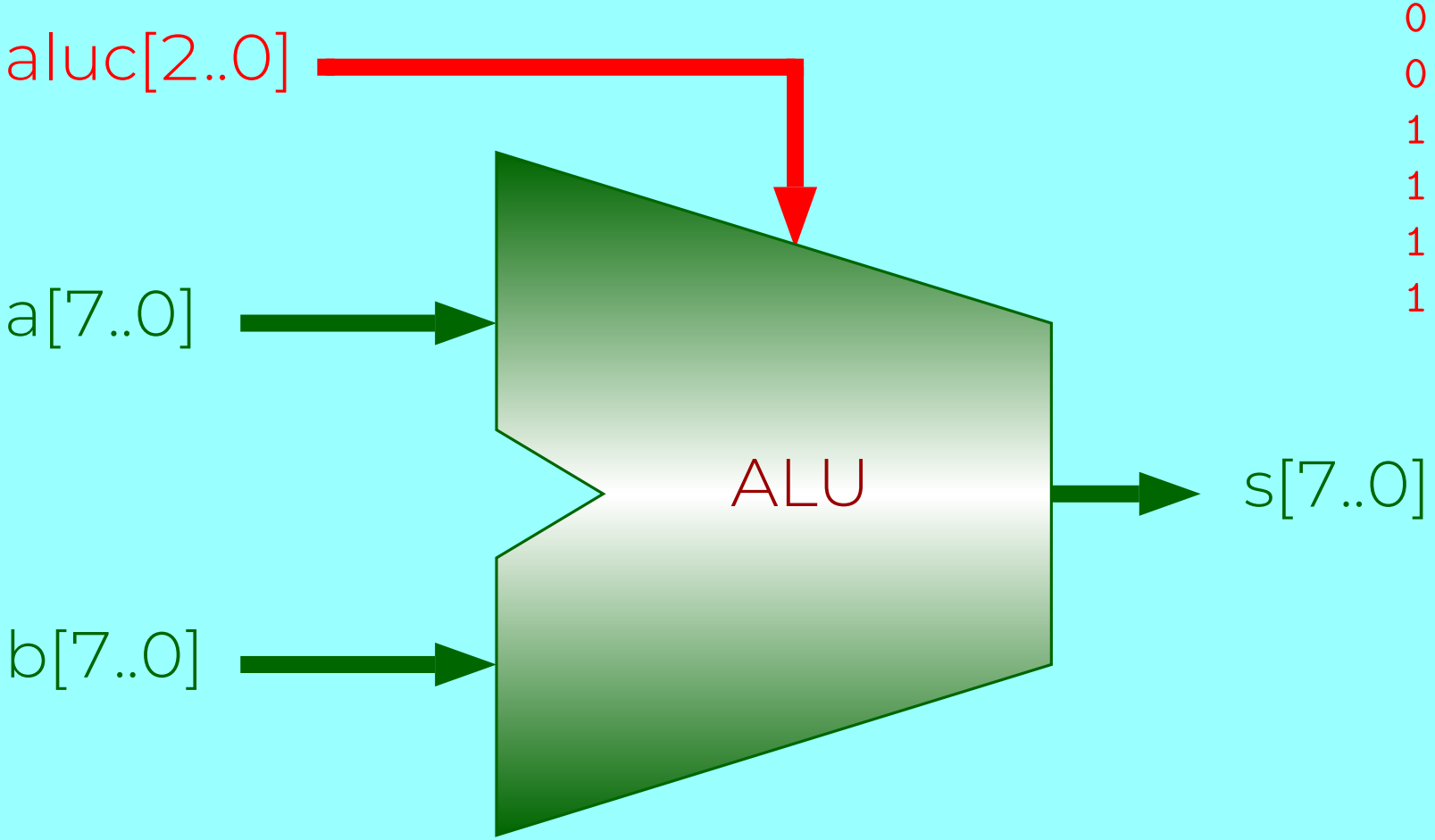
簡単な ALU の設計 (シンボル)

ALU: Arithmetic Logic Unit

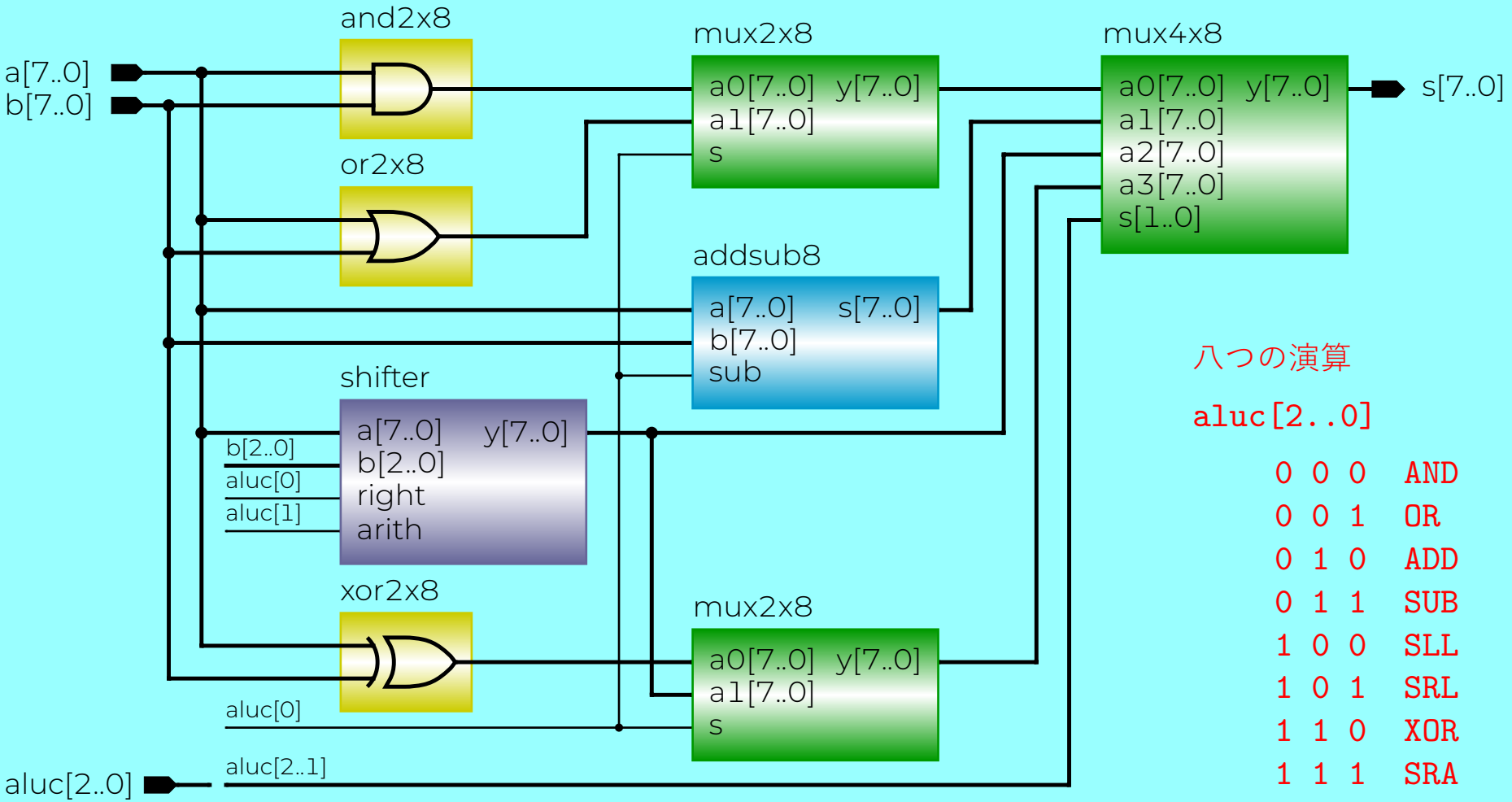
算術論理演算回路

八つの演算

aluc[2..0]	
0 0 0	AND
0 0 1	OR
0 1 0	ADD
0 1 1	SUB
1 0 0	SLL
1 0 1	SRL
1 1 0	XOR
1 1 1	SRA



簡単な ALU の設計 (回路)



八つの演算

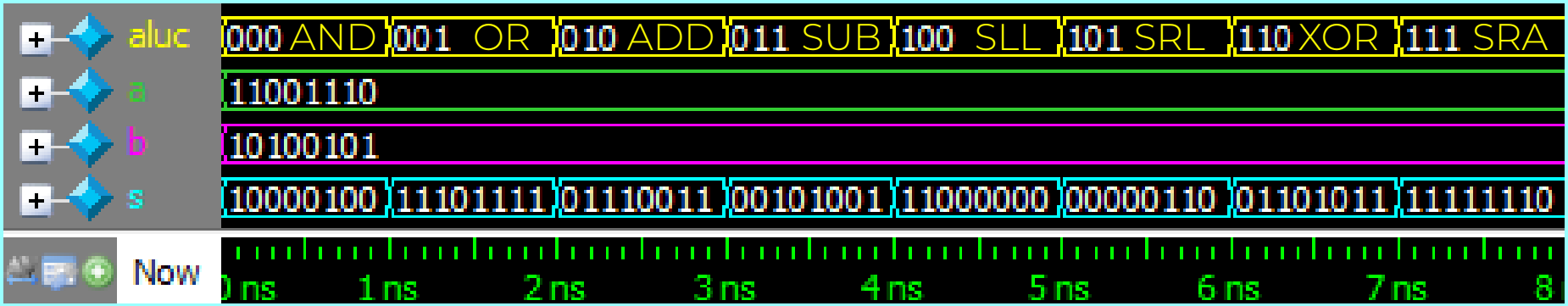
`aluc[2..0]`

0 0 0	AND
0 0 1	OR
0 1 0	ADD
0 1 1	SUB
1 0 0	SLL
1 0 1	SRL
1 1 0	XOR
1 1 1	SRA

すべての演算を用意して、マルチプレクサで選択する。

簡単な ALU の設計 (波形)

ALU のシミュレーション波形



０００(論理積)	００１(論理和)	０１０(足し算)	０１１(引き算)	← aluc
１１００１１１０	１１００１１１０	１１００１１１０	１１００１１１０	← a
& １０１００１０１	| １０１００１０１	+ １０１００１０１	− １０１００１０１	← b
１００００１００	１１１０１１１１	０１１１００１１	００１０１００１	← s

１００(SLL)	１０１(SRL)	１１０(XOR)	１１１(SRA)	← aluc
１１００１１１０	１１００１１１０	１１００１１１０	１１００１１１０	← a
← １０１００１０１	→ １０１００１０１	⊕ １０１００１０１	→ １０１００１０１	← b
１１０００００	００００１１０	０１１０１０１１	１１１１１１１０	← s

マルチプレクサ

まとめ

- マルチプレクサ
- 組み合わせ回路設計の手順
- 1 ビット 2-to-1 のマルチプレクサ
- 4 ビット 2-to-1 のマルチプレクサ
- 1 ビット 4-to-1 のマルチプレクサ
- 4 ビット 4-to-1 のマルチプレクサ
- バレルシフタ (Barrel shifter) 回路
- 7 セグメント LED
- ALU: Arithmetic Logic Unit (算術論理演算回路)

課題 IX (200 点 + 100 点)

問題１： (100 点): 4 ビット 4-to-1 のマルチプレクサ回路を設計し動作検証シミュレーションして下さい (15 ～ 16 ページを参照)。
入力信号: A0[3..0]、A1[3..0]、A2[3..0]、A3[3..0]
入力信号: S[1..0] . . . . . 注意：まとめの表示
出力信号: Y[3..0]
プロジェクト名は mux4x4 にすること。
テストベンチ mux4x4_tb.v を使って下さい。

問題２： (100 点): 16 進数 7 セグメント LED 点灯回路を設計し動作検証シミュレーションして下さい (27 ～ 28 ページを参照)。
プロジェクト名は led7seg にすること。
テストベンチ led7seg_tb.v を使って下さい。

オプション (+100 点): 簡単な ALU を設計し動作検証シミュレーションして下さい (30 ～ 33 ページを参照)。
プロジェクト名は alu8 にすること。
テストベンチ alu8_tb.v を使って下さい。

発展：自由練習

Verilog HDL による課題の実装

1 ビット 4-to-1 マルチプレクサの設計（P15 を参照）

```
module mux4x1b (A0, A1, A2, A3, S, Y);
    input  [1:0] S;
    input        A0, A1, A2, A3;
    output       Y;
    assign Y = ~S[1] & ~S[0] & A0 | // S = 00: Y = A0
                                  | // S = 01: Y = A1
                                  | // S = 10: Y = A2
                                  ;  // S = 11: Y = A3
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

Verilog HDL による課題の実装

4 ビット 4-to-1 マルチプレクサの設計（P16 を参照）

```
module mux4x4b (A0, A1, A2, A3, S, Y);
    input  [1:0] S;
    input  [3:0] A0, A1, A2, A3;
    output [3:0] Y;
    mux4x1b i0 (A0[0], A1[0], A2[0], A3[0], S, Y[0]); // bit 0
    mux4x1b i1 (                                    ); // bit 1
    mux4x1b i2 (                                    ); // bit 2
    mux4x1b i3 (                                    ); // bit 3
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

Verilog HDL による課題の実装

16 進数 7 セグメント LED 点灯回路（P27、P28 を参照）

```
module seven_seg_led (n, led);
    input  [3:0] n;   // 4 bits, 2^4=16: 0-F in hexadecimal format
    output [6:0] led; // 7-segment LEDs: 0: light on; 1: light off
    assign led[6] =                                          ; // segment 6
    assign led[5] =                                          ; // segment 5
    assign led[4] =                                          ; // segment 4
    assign led[3] =                                          ; // segment 3
    assign led[2] =                                          ; // segment 2
    assign led[1] =                                          ; // segment 1
    assign led[0] =                                          ; // segment 0
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

Verilog HDL による課題の実装

簡単な ALU の回路（P30 ~ P33 を参照）

```
module alu8 (a, b, aluc, s);
    input  [7:0] a, b;
    input  [2:0] aluc;
    output [7:0] s;
    wire   [7:0] d_and = a & b;                          // AND
    wire   [7:0] d_or  = a | b;                          // OR
    wire   [7:0] d_xor = a ^ b;                          // XOR
    wire   [7:0] d_ao, d_as, d_sh, d_xa;                 // internal wires
    mux2x8b mx_a (d_and, d_or, aluc[0], d_ao);           // 0: AND or OR
    addsub8 as   (a,     b,    aluc[0], d_as);           // 1: ADD or SUB
    shift   sh   (a,   b[2:0], aluc[0], aluc[1], d_sh); // 2: 3: barrel shifter
    mux2x8b mx_x (   ,    ,    aluc[0], d_xa);           // 3: XOR or SRA
    mux4x8b mx_s (   ,    , d_sh,     , aluc[2:1], s);  // final result s
endmodule
```

上記コードを完成しシミュレーションして下さい。

発展：自由練習

Verilog HDL による課題の実装

バレルシフタの回路（P17 ~ P24 を参照）

```
module barrel_shifter (d, sa, right, arith, sh);
    input  [31:0] d;                 // input: 32-bit data to be shifted
    input   [4:0] sa;                // input: shift amount, 5 bits
    input         right;             // 1: shift right; 0: shift left
    input         arith;             // 1: arithmetic shift; 0: logical
    output [31:0] sh;                // output: shifted result
    reg    [31:0] sh;                // will be combinational
    always @* begin                  // always block
        if (!right) begin            // if shift left
            sh = d << sa;            //    shift left sa bits
        end else if (!arith) begin   // if shift right logical
            sh = d >> sa;            //    shift right logical sa bits
        end else begin               // if shift right arithmetic
            sh = $signed(d) >>> sa; //    shift right arithmetic sa bits
        end
    end
endmodule
```

上記コードを完成しシミュレーションして下さい。