

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

2023 年 11 月 21 日 (火)

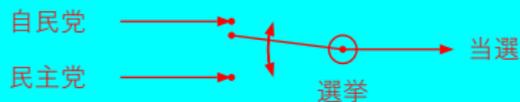
マルチプレクサ

ポイント

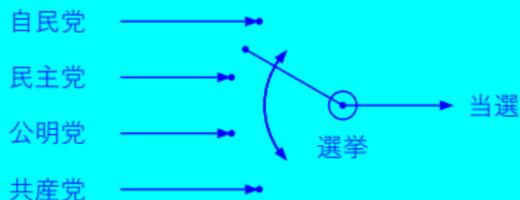
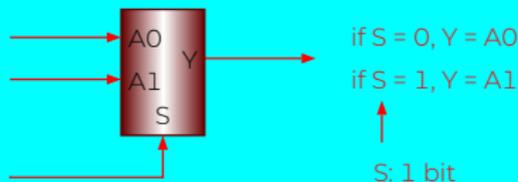
- マルチプレクサ
- 組み合わせ回路設計の手順
- 1 ビット 2-to-1 のマルチプレクサ
- 4 ビット 2-to-1 のマルチプレクサ
- 1 ビット 4-to-1 のマルチプレクサ
- 4 ビット 4-to-1 のマルチプレクサ
- バレルシフタ (Barrel shifter) 回路
- 7 セグメント LED
- ALU: Arithmetic Logic Unit (算術論理演算回路)

マルチプレクサ

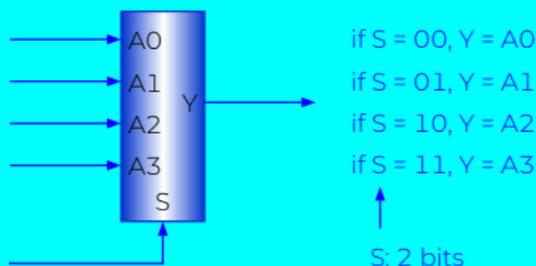
マルチプレクサ (Multiplexer) は、複数の入力から1つを選択し出力する。
(マルチプレクサ = データ選択器)



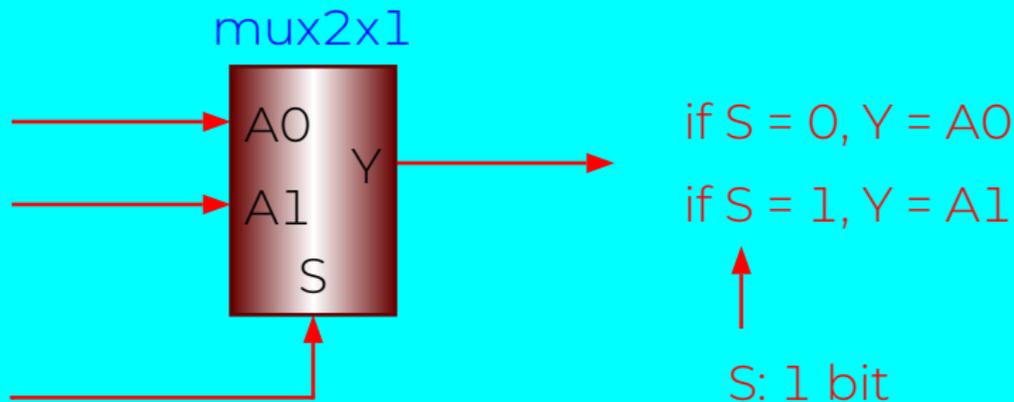
2-to-1 マルチプレクサ



4-to-1 マルチプレクサ



2-to-1 のマルチプレクサを設計



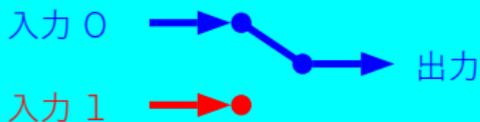
1 ビット 2-to-1 のマルチプレクサ回路を設計しましょう。

組み合わせ回路設計の手順

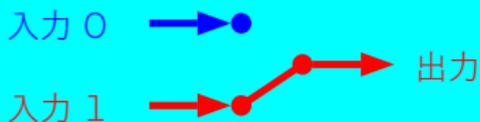
- ① 問題を理解する。
- ② 入力と出力信号の名前を任意に決める。
- ③ 入力のあらゆる組み合わせを列挙し，真理値表をつくる。
- ④ 真理値表から論理式をつくる。
(どの組み合わせで出力が 1 になるか。)
- ⑤ カルノー図を用いて論理式を簡単化する。
- ⑥ 論理式から回路をつくる。
- ⑦ テストベンチをつくる (シミュレーションするため)。
- ⑧ 回路をシミュレーションする (回路設計の正当性検証)。
- ⑨ 与えられた回路の動作を理解する (波形の説明)。

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。



入力 0 を選択

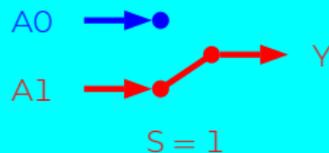
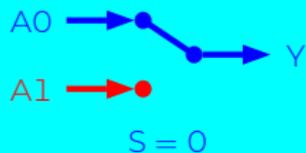


入力 1 を選択

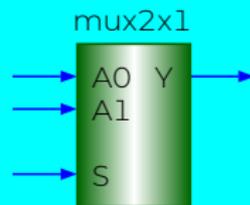
マルチプレクサ — 信号

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
- ▶ どちらを選択するかを告げる信号 (入力): S
 - ★ $\text{if } (S == 0) Y = A0; \dots\dots\dots S$ は 0 のとき、A0 を選択する
 - ★ $\text{else } Y = A1; \dots\dots\dots S$ 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は出力されているのでしょうか？



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
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Sは0のとき、
A0を選択する

Sは1のとき、
A1を選択する

マルチプレクサ — 論理式

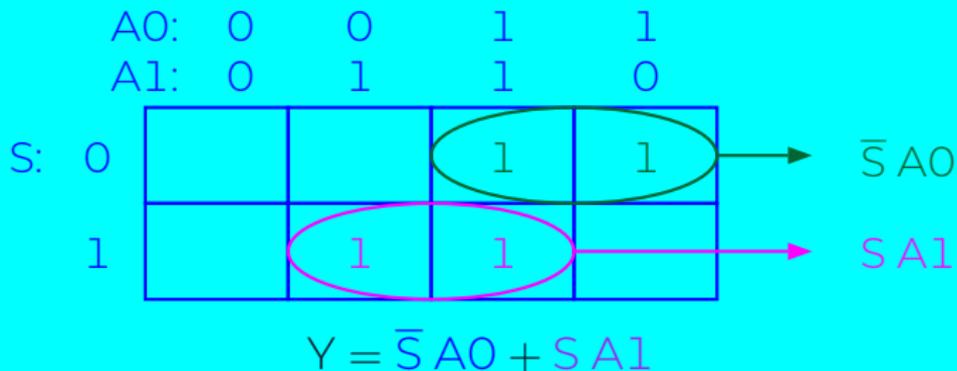
4. 論理式をつくる (どの組み合わせで出力が1になるか)。

S	入力		出力	
	A1	A0	Y	
0	0	0	0	
0	0	1	1	← $\bar{S}\bar{A}1A0$
0	1	0	0	
0	1	1	1	← $\bar{S}A1A0$
1	0	0	0	
1	0	1	0	
1	1	0	1	← $S\bar{A}1\bar{A}0$
1	1	1	1	← $SA1A0$

$$Y = \bar{S}\bar{A}1A0 + \bar{S}A1A0 + S\bar{A}1\bar{A}0 + SA1A0$$

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:



あるいはブール代数の定理により論理式を簡単化する:

$$\begin{aligned} Y &= \bar{S}\bar{A}1A0 + \bar{S}A1A0 + SA1\bar{A}0 + SA1A0 \\ &= \bar{S}A0(\bar{A}1 + A1) + SA1(\bar{A}0 + A0) \\ &= \bar{S}A0 + SA1 \end{aligned}$$

マルチプレクサ — 回路

6. 論理式から回路をつくる。

$$Y = \bar{S}A0 + SA1 \quad \rightarrow \quad Y = \sim S \& A0 \mid S \& A1$$

```
'timescale 1ns/1ns

module mux2x1 (A0, A1, S, Y);
    input  A0, A1, S;
    output Y;

    assign Y = ~S & A0 | S & A1;

endmodule
```

[mux2x1.v](#)

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/1ns
module mux2x1_tb;
    reg A0, A1, S;
    wire Y;
    mux2x1 i0 (A0, A1, S, Y);
    initial begin
        #0 A0 = 0; A1 = 0; S = 0;
        #8 $finish;
    end
    always #1 A0 = ~A0;
    always #2 A1 = ~A1;
    always #4 S = ~S;
    initial begin
        $dumpfile ("mux2x1.vcd");
        $dumpvars;
    end
endmodule
```

[mux2x1_tb.v](#)

マルチプレクサ — 波形

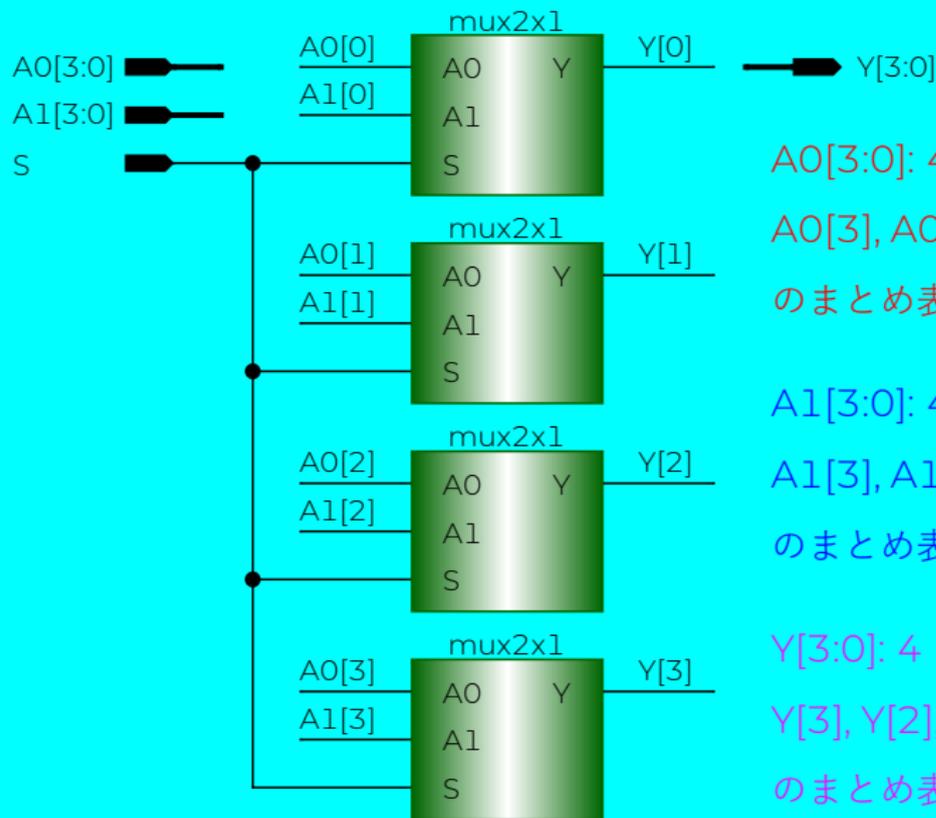
```
% iverilog -Wall -o mux2x1 mux2x1_tb.v mux2x1.v  
% ./mux2x1  
% gtkwave mux2x1.vcd
```



$S = 0$ の時、 $Y = \bar{S} \cdot A0 + S \cdot A1 = 1 \cdot A0 + 0 \cdot A1 = A0 + 0 = A0$

$S = 1$ の時、 $Y = \bar{S} \cdot A0 + S \cdot A1 = 0 \cdot A0 + 1 \cdot A1 = 0 + A1 = A1$

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

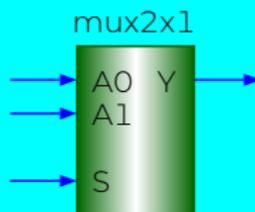
$A1[3:0]$: 4ビット幅である
 $A1[3], A1[2], A1[1], A1[0]$
のまとめ表示であること

$Y[3:0]$: 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

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



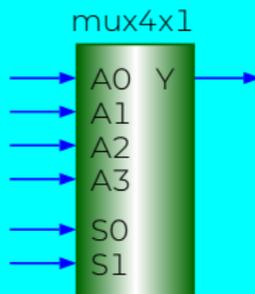
真理値表

S	Y
0	A0
1	A1

```
assign Y = ~S & A0 | S & A1;
```

● 4-to-1 のマルチプレクサ

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



真理値表

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

```
assign Y = ??????????????????;
```

4-to-1 のマルチプレクサ (回路)

$$\begin{aligned} Y &= \overline{S1} \cdot \overline{S0} \cdot A0 \\ &+ \overline{S1} \cdot S0 \cdot A1 \\ &+ S1 \cdot \overline{S0} \cdot A2 \\ &+ S1 \cdot S0 \cdot A3 \end{aligned}$$

```
'timescale 1ns/1ns
module mux4x1 (A0, A1, A2, A3, S0, S1, Y);
    input  A0, A1, A2, A3, S0, S1;
    output Y;
    assign Y = ~S1 & ~S0 & A0 | // S1 S0 = 0 0, Y = A0
               ~S1 &  S0 & A1 | // S1 S0 = 0 1, Y = A1
               S1  & ~S0 & A2 | // S1 S0 = 1 0, Y = A2
               S1  &  S0 & A3; // S1 S0 = 1 1, Y = A3
endmodule
```

[mux4x1.v](#)

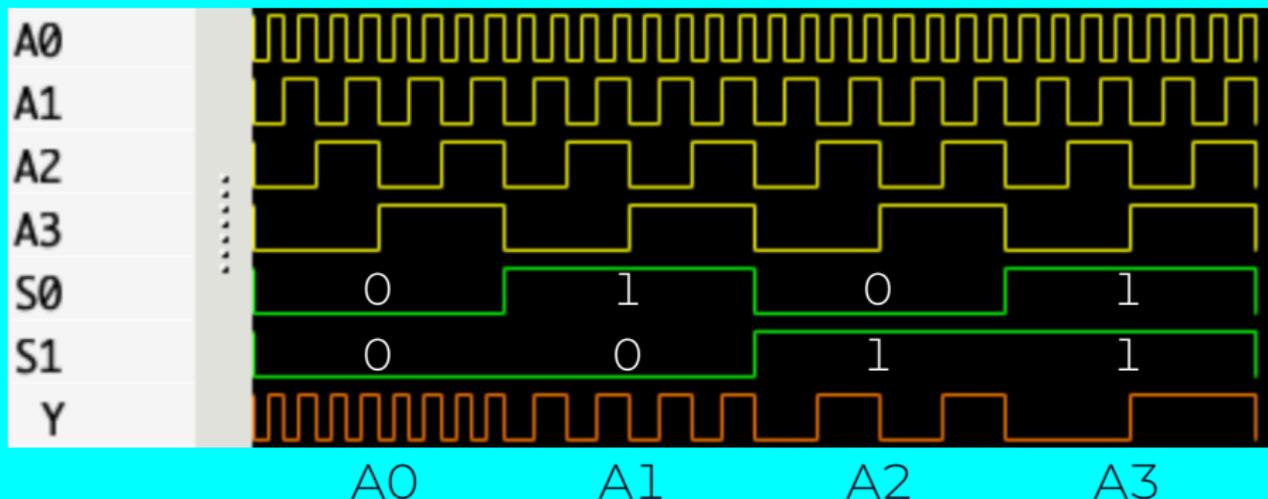
テストベンチ

```
'timescale 1ns/1ns
module mux4x1_tb;
    reg  A0, A1, A2, A3, S0, S1;
    wire Y;
    mux4x1 i0 (A0, A1, A2, A3, S0, S1, Y);
    initial begin
        #0  A0 = 0; A1 = 0; A2 = 0; A3 = 0;
        #0  S0 = 0; S1 = 0;
        #64 $finish;
    end
    always #1  A0 = ~A0;
    always #2  A1 = ~A1;
    always #4  A2 = ~A2;
    always #8  A3 = ~A3;
    always #16 S0 = ~S0;
    always #32 S1 = ~S1;
    initial begin
        $dumpfile ("mux4x1.vcd");
        $dumpvars;
    end
endmodule
```

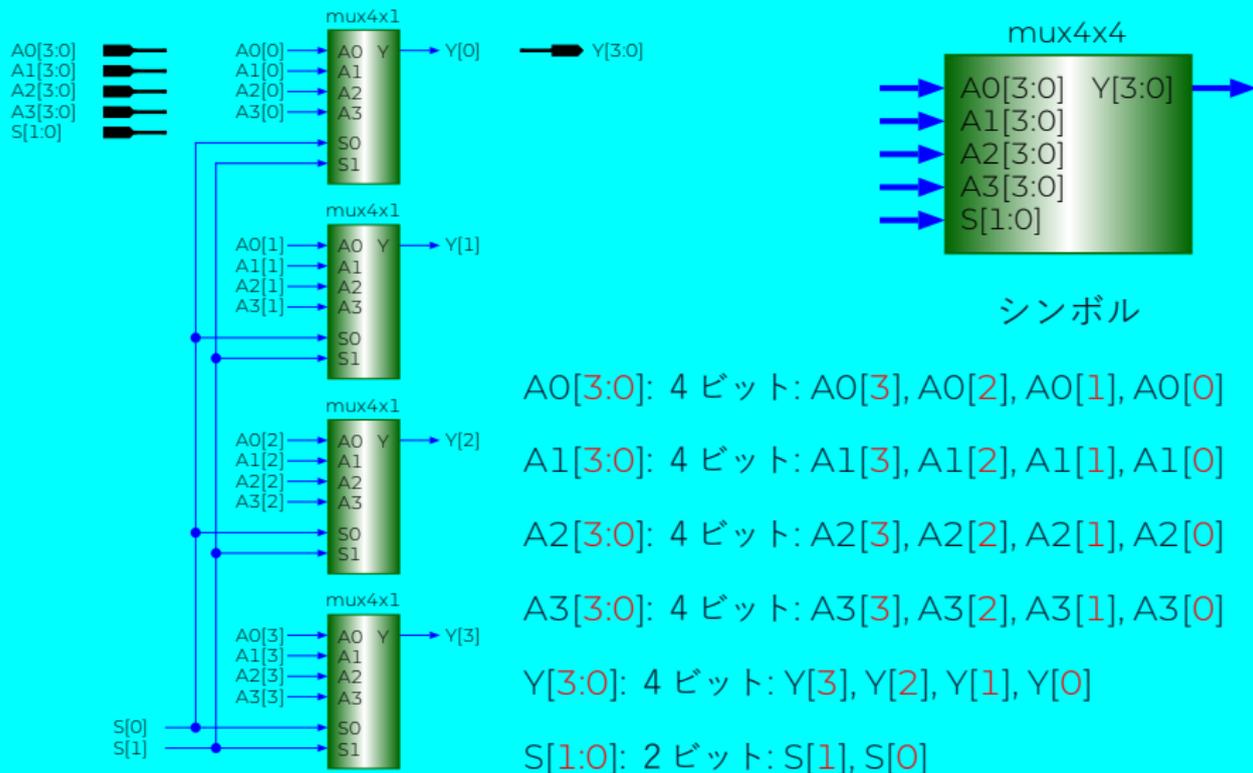
[mux4x1_tb.v](#)

4-to-1 のマルチプレクサ (波形)

```
% iverilog -Wall -o mux4x1 mux4x1_tb.v mux4x1.v  
% ./mux4x1  
% gtkwave mux4x1.vcd
```



4-to-1 のマルチプレクサ (4ビット)



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)

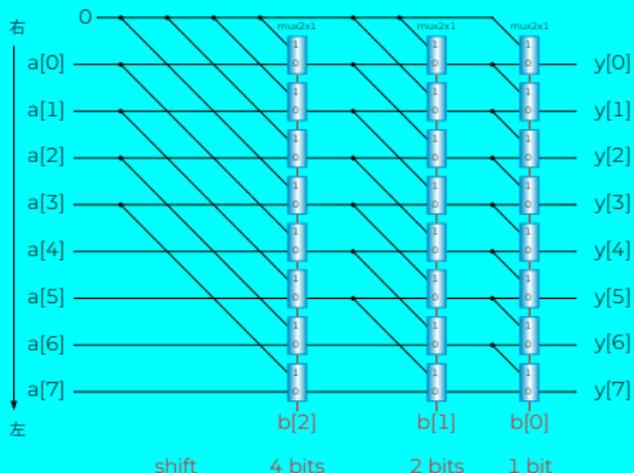
▶ 1010 0011 4-bit shift right arithmetic:

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) → 11110000 (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

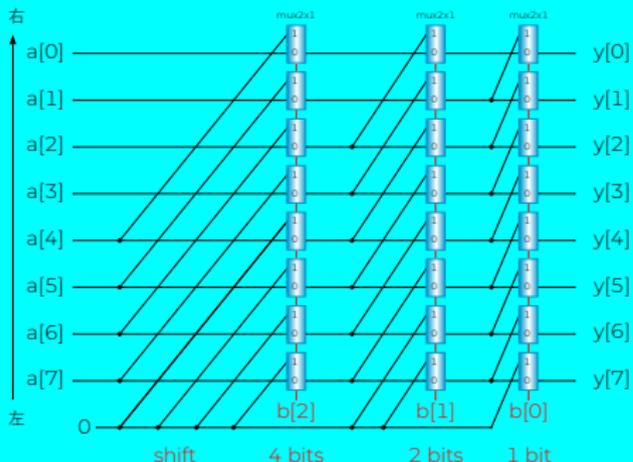
If $b[1] = 1$, shift $2^1 = 2$ bits

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

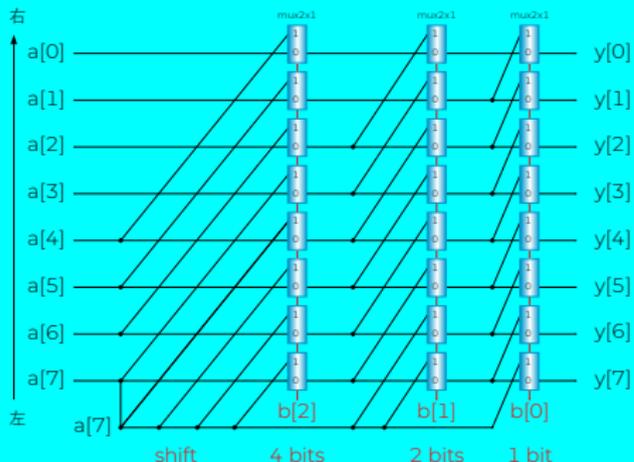
If $b[1] = 1$, shift $2^1 = 2$ bits

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)

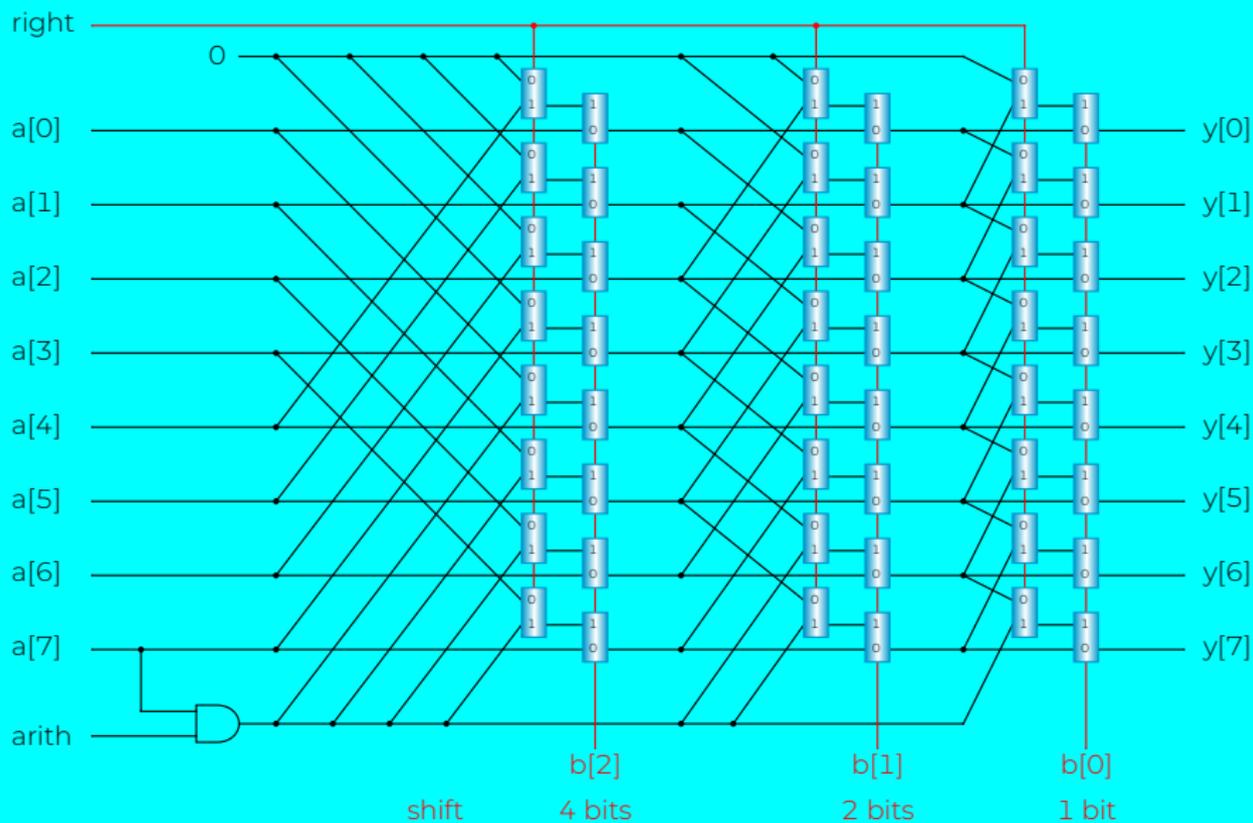


If $b[2] = 1$, shift $2^2 = 4$ bits

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



バレルシフタの波形 (論理左シフト)

a[7:0]	01111110							
b[2:0]	000	001	010	011	100	101	110	111
right							
arith								
y[7:0]	01111110	11111100	11111000	11110000	11100000	11000000	10000000	00000000

[shift_tb.v](#)

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)

a[7:0]	01111110				10000000			
b[2:0]	000	001	010	011	100	101	110	111
right								
arith								
y[7:0]	01111110	00111111	00011111	00001111	00001000	00000100	00000010	00000001

[shift_tb.v](#)

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)

a[7:0]	01111110			10000000				
b[2:0]	000	001	010	011	100	101	110	111
right							
arith								
y[7:0]	01111110	00111111	00011111	00001111	11111000	11111100	11111110	11111111

[shift_tb.v](#)

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) 入出インターフェース

(3) CPU (プロセッサ)

ALU

...

レジスタファイル

2. ソフトウェア

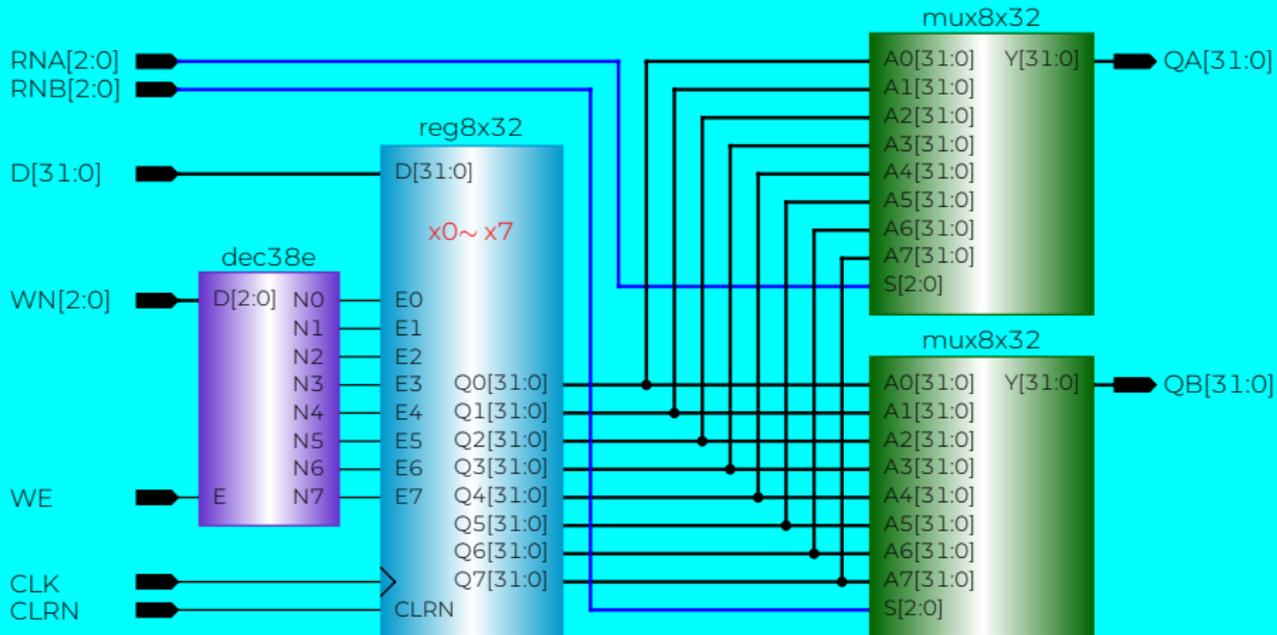
(OS やコンパイラなど)

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



イネーブル付き
3入力8出力デコーダ

レジスタアレイ
(8個×32ビット)

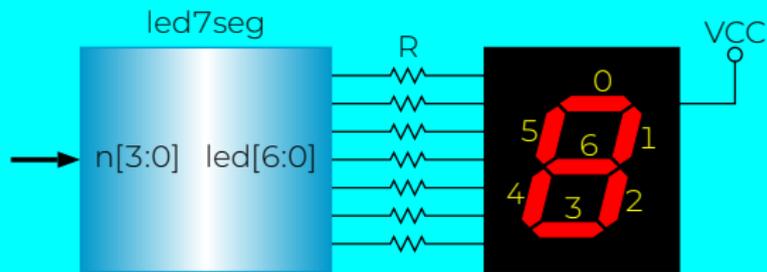
マルチプレクサ
(8入力×32ビット)

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

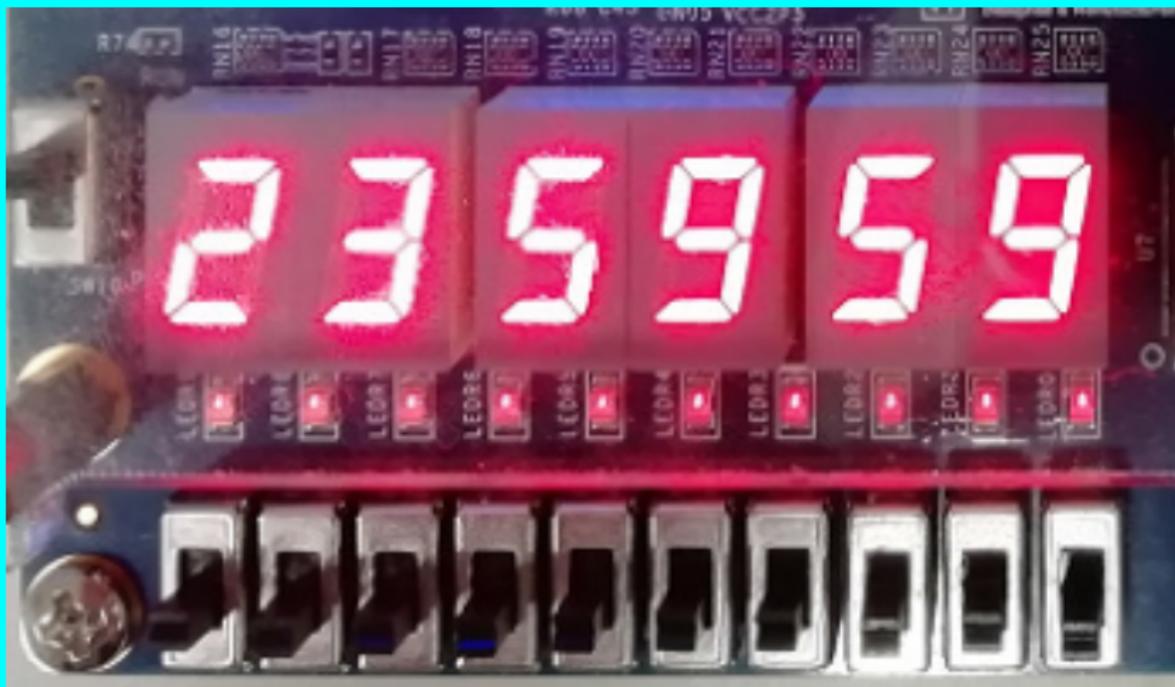
真理値表

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 出力信号を単純化する。

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) 入出カインターフェース

(3) CPU (プロセッサ)

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

八つの演算

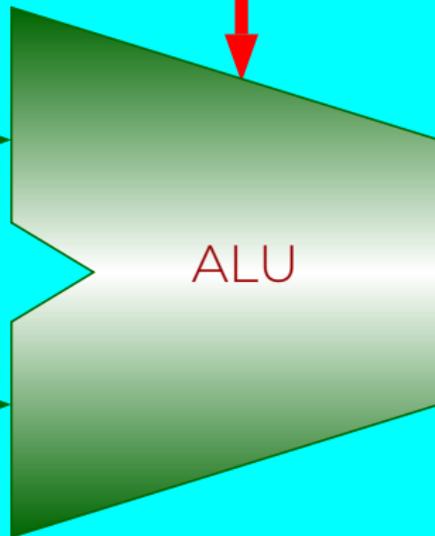
aluc[2:0]

算術論理演算回路

aluc[2:0]

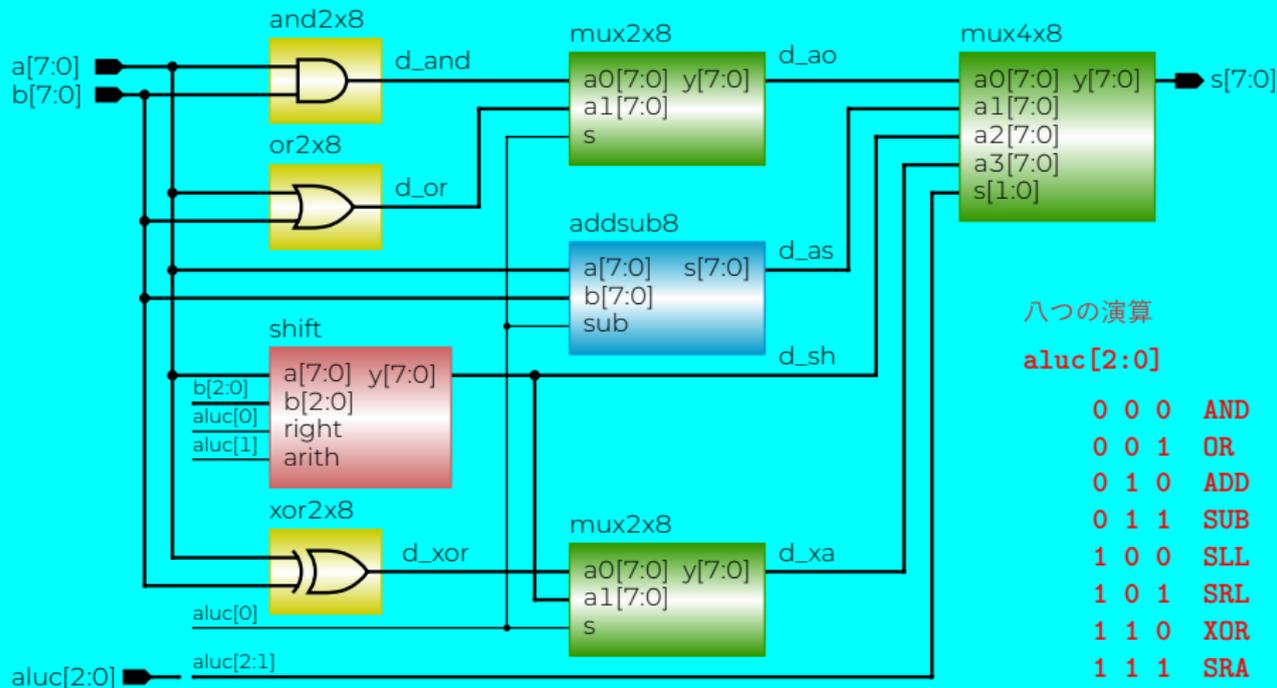
a[7:0]

b[7:0]



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)



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

aluc[2:0]	000 AND	001 OR	010 ADD	011 SUB	100 SLL	101 SRL	110 XOR	111 SRA
a[7:0]	11001110							
b[7:0]	10100101							
s[7:0]	10000100	11101111	01110011	00101001	11000000	00000110	01101011	11111110

[alu8_tb.v](#)

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
11000000	00000110	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 (15-19 ページを参照)。

```
'timescale 1ns/1ns
module mux4x4 (A0, A1, A2, A3, S, Y);
    input  [1:0] S;
    input  [3:0] A0, A1, A2, A3;
    output [3:0] Y;

    // mux4x1 (A0,      A1,      A2,      A3,      S0,      S1,      Y);

    mux4x1 i0 (A0[0], A1[0], A2[0], A3[0], S[0], S[1], Y[0]); // bit 0
    mux4x1 i1 (      ,      ,      ,      ,      ,      ,      ); // bit 1
    mux4x1 i2 (      ,      ,      ,      ,      ,      ,      ); // bit 2
    mux4x1 i3 (      ,      ,      ,      ,      ,      ,      ); // bit 3

endmodule
```

モジュール名は [mux4x4](#) にすること。
テストベンチ [mux4x4_tb.v](#) を使って下さい。

課題 IX (200 点 + 100 点)

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 (30~31 ページを参照)。

```
'timescale 1ns/1ns
module led7seg (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] = n[3] & n[2] & n[1] & n[0]; // segment 6
    assign led[5] = n[3] & n[2] & n[1]; // segment 5
    assign led[4] = n[3] & n[2]; // segment 4
    assign led[3] = n[3] & n[1]; // segment 3
    assign led[2] = n[3] & n[0]; // segment 2
    assign led[1] = n[2] & n[1]; // segment 1
    assign led[0] = n[2] & n[0]; // segment 0
endmodule
```

モジュール名は [led7seg](#) にすること。
テストベンチ [led7seg_tb.v](#) を使って下さい。

課題 IX (200 点 + 100 点)

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 (33~36 ページを参照)。

```
'timescale 1ns/1ns
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
    mux2x8 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: SLL or SRL
    mux2x8 mx_x ( , , , d_xa); // 3: XOR or SRA
    mux4x8 mx_s ( , , , , s); // final result s
endmodule
```

モジュール名は [alu8](#) にすること。

テストベンチ [alu8_tb.v](#) を使って下さい。