

2018年度 計算機システム(演習)
第6回
2019.01.11

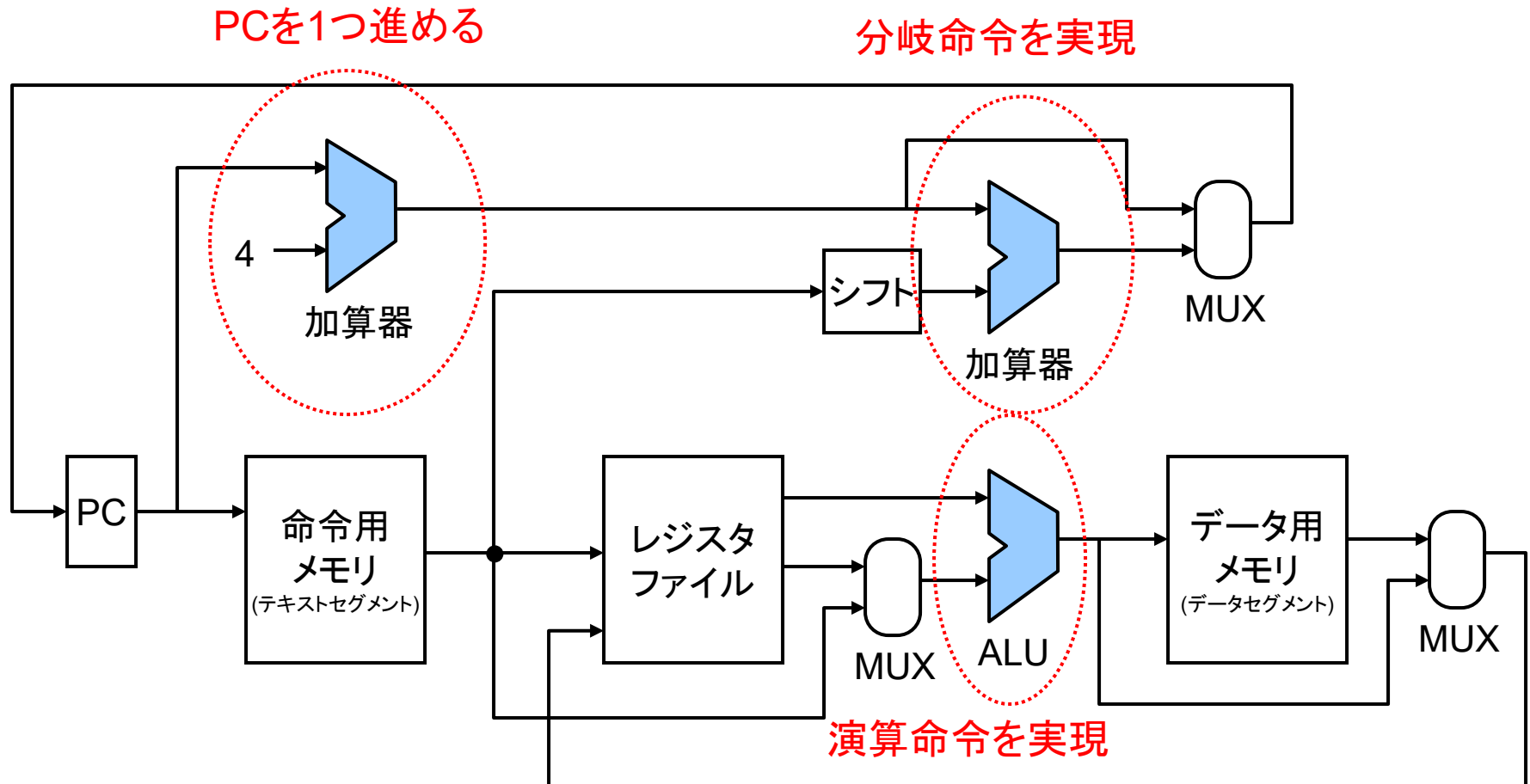
遠藤 敏夫(学術国際情報センター/数理・計算科学系 教授)
野村 哲弘(学術国際情報センター/数理・計算科学系 助教)

本日の内容 (Outline)

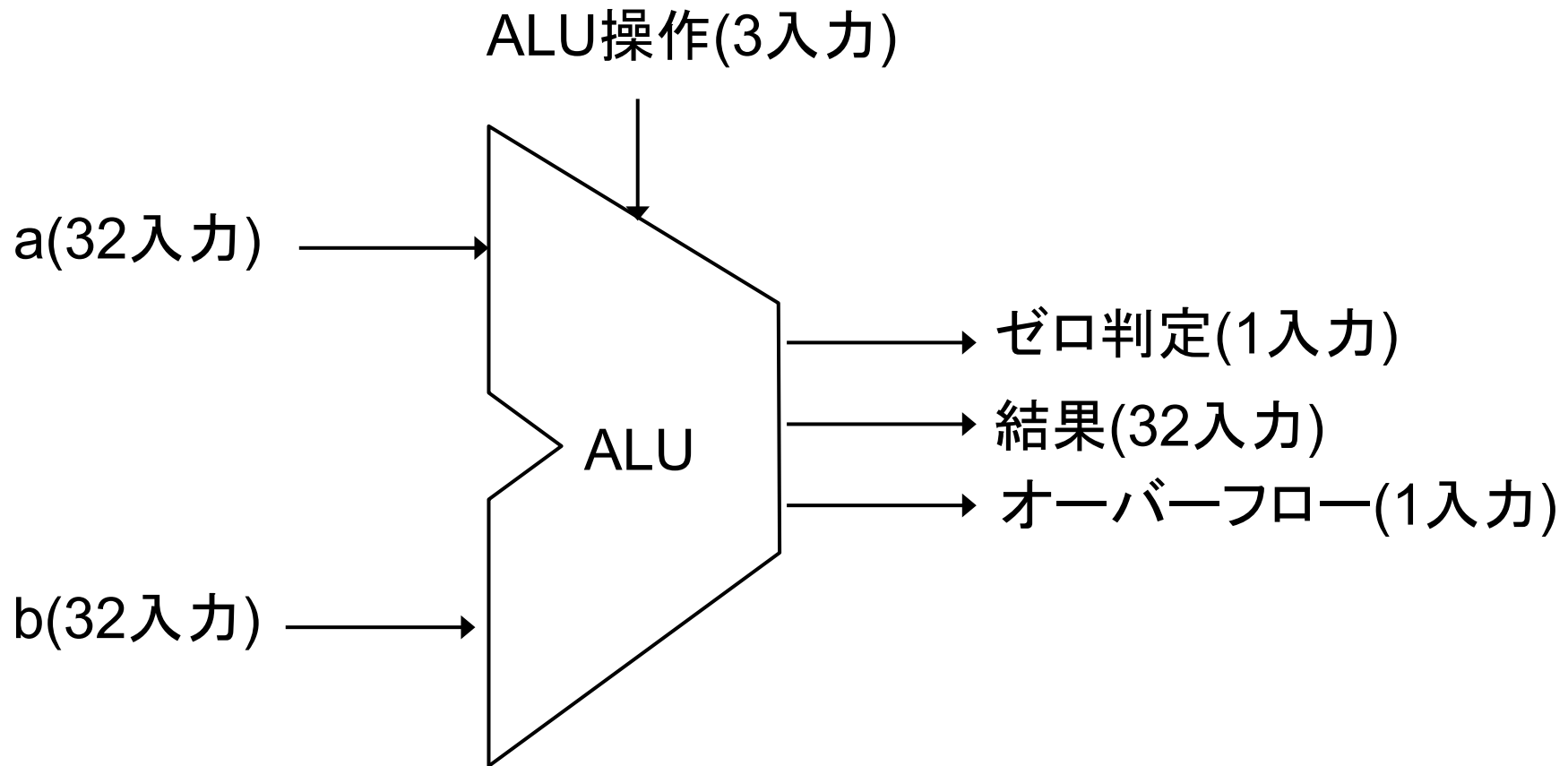
▶ ALUの作成

- ▶ 1ビット加算器の作成
- ▶ 32ビット加算器の作成
- ▶ 1ビットALU
- ▶ 32ビットALU
 - ▶ ver.1: 論理積、論理和、加算
 - ▶ ver.2: ver.1 + 減算
 - ▶ ver.3: ver.2 + 比較演算(slt)
 - ▶ ver.4: ver.3 + 等号演算

MIPSシミュレータの概略図



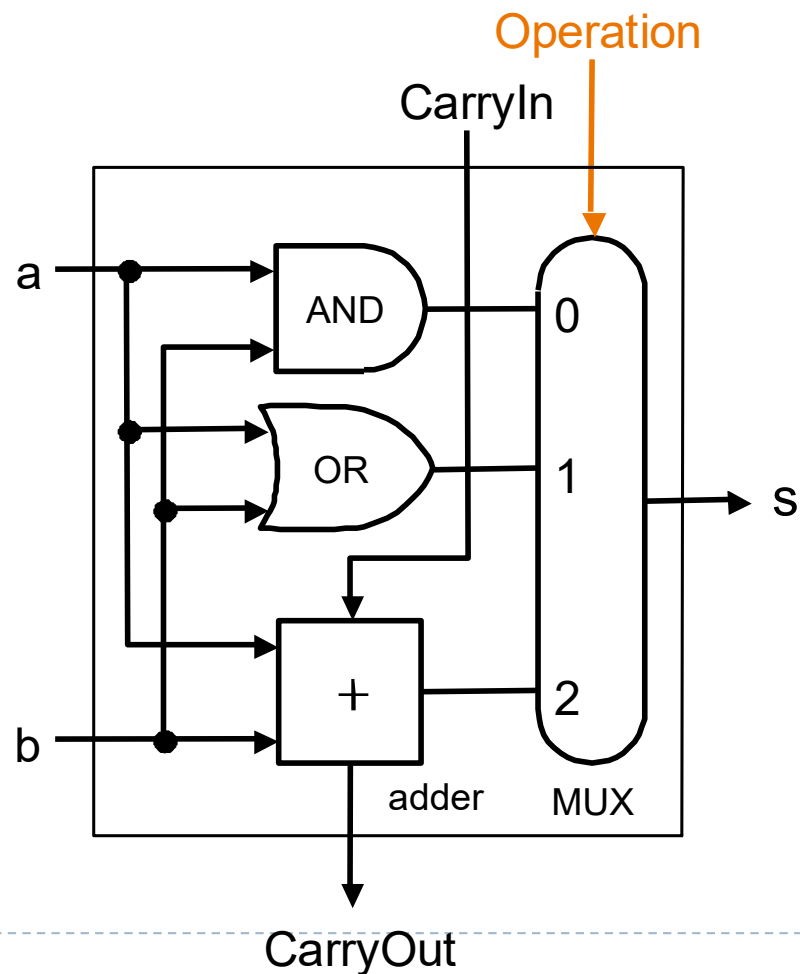
32ビット ALU完成 (Overflow)



1ビットALU (ver. 1)

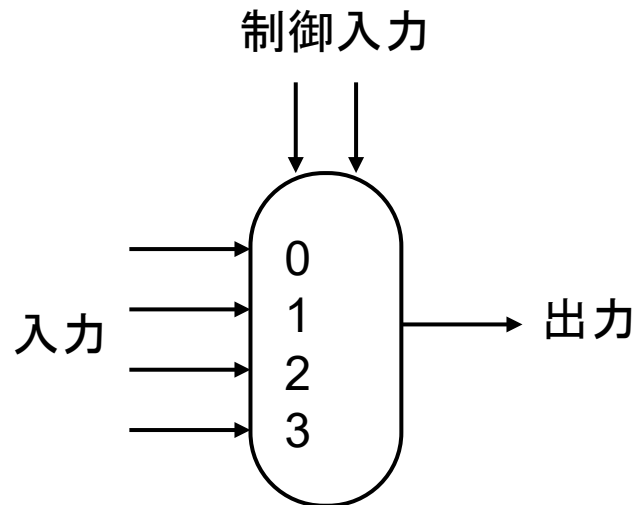
- ▶ まず、加算、論理積、論理和をサポート
- ▶ マルチプレクサ(MUX)でどの結果を出力するかを決定
 - ▶ ANDGate
 - ▶ ORGate
 - ▶ Adder

Operation	演算
00	AND
01	OR
10	加算
11	-



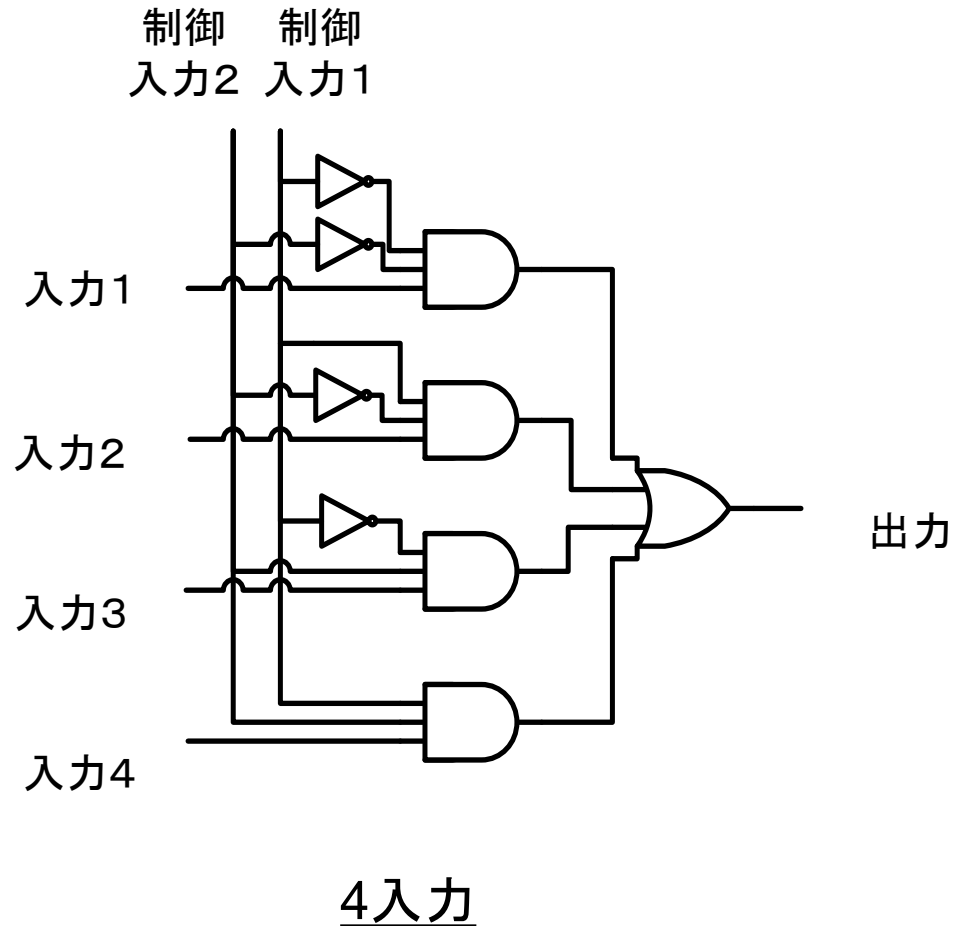
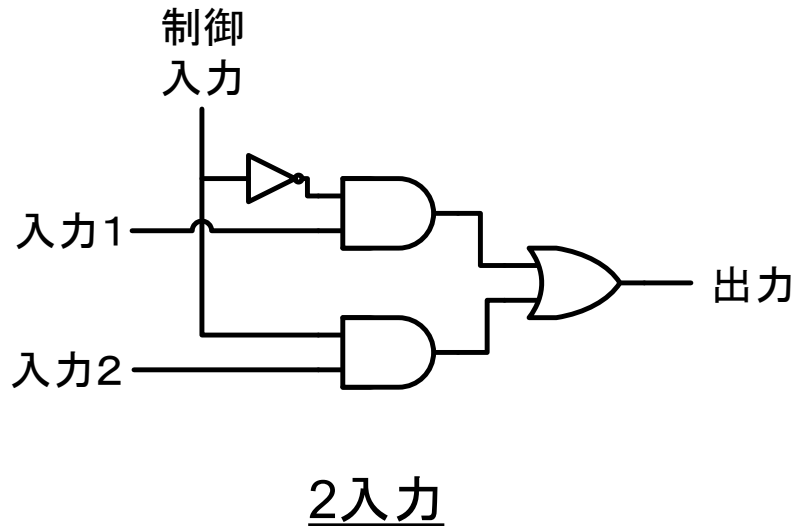
マルチプレクサ(MUX)

- ▶ 制御信号に従って、複数の入力から1つの出力を得る
 - ▶ 制御信号が表す二進数に対応する入力を選ばれる
 - ▶ 制御入力: n ビット $\Rightarrow 2^n$ 通りの選択



制御信号	選ばれる入力
00	0 番
01	1 番
10	2 番
11	3 番

マルチプレクサの回路図



MUX (2入力)、MUX4 (4入力)

```
void mux(Signal in1, Signal in2, Signal ctl, Signal *out1)
{
    // 回路を組み合わせる
}
```

```
void mux4(Signal in1, Signal in2, Signal in3, Signal in4,
          Signal ctl1, Signal ctl2, Signal *out1)
{
    // 省略
}
```


1ビットALU (ver. 1) (再)

- ▶ まず、加算、論理積、論理和をサポート
- ▶ マルチプレクサ(MUX)でどの結果を出力するかを決定

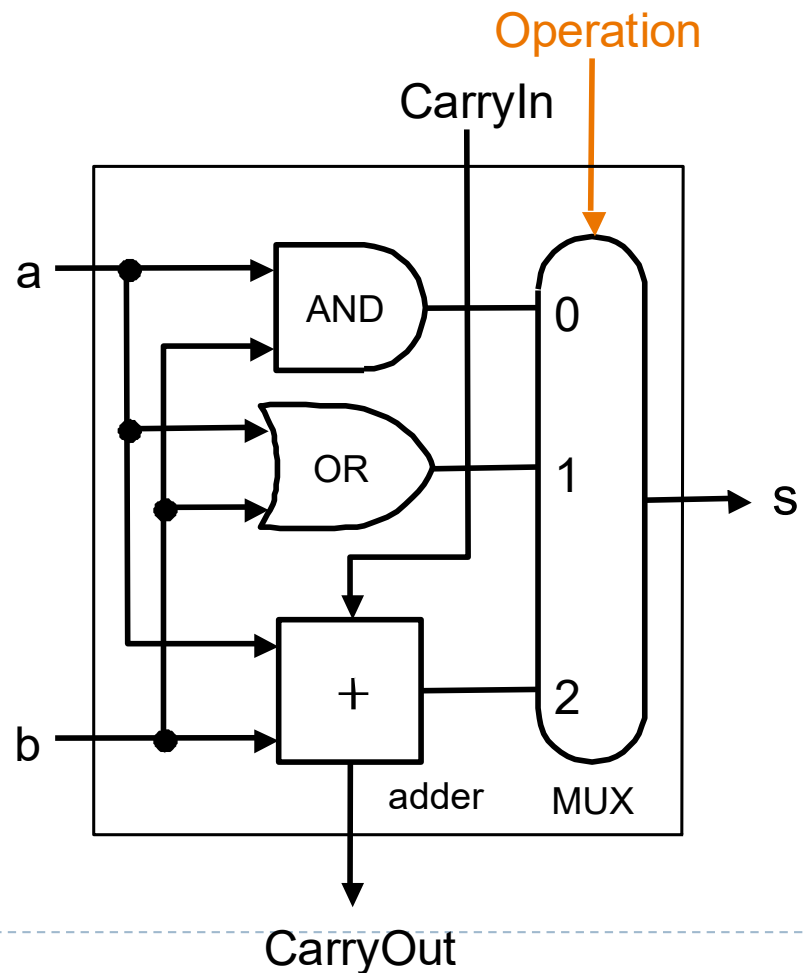
- ▶ ANDGate

- ▶ ORGate

- ▶ Adder

op[1], op[0]

Operation	演算
00	AND
01	OR
10	加算
11	-

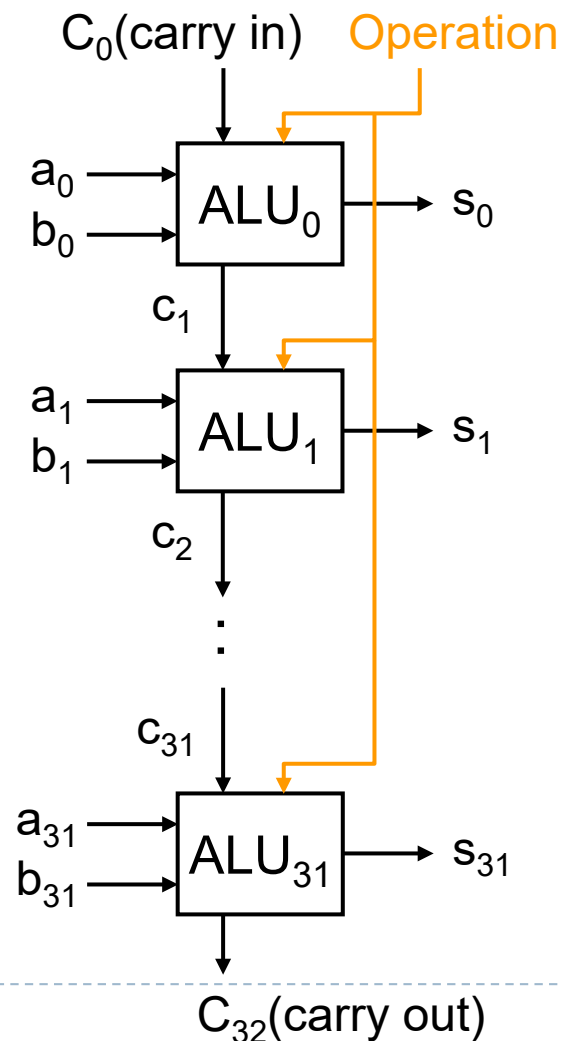


1ビットALU (ver. 1)

```
void alu(Signal *ops, //制御入力
        Signal a, Signal b, Signal carry_in, //入力
        Signal *s, Signal *carry_out) //出力
{
    // AND Gate, OR Gate, FAを実行
    // 最後にMUX4で選択
    // 各ゲートの出力とMUX4の接続は別途Signalを用意する
}
```

32ビットALU (ver. 1)

- ▶ 1ビットALU を 32 個つなぐ
 - ▶ Ripple Carry Adder で実装



32ビットALU (ver. 1) クラス

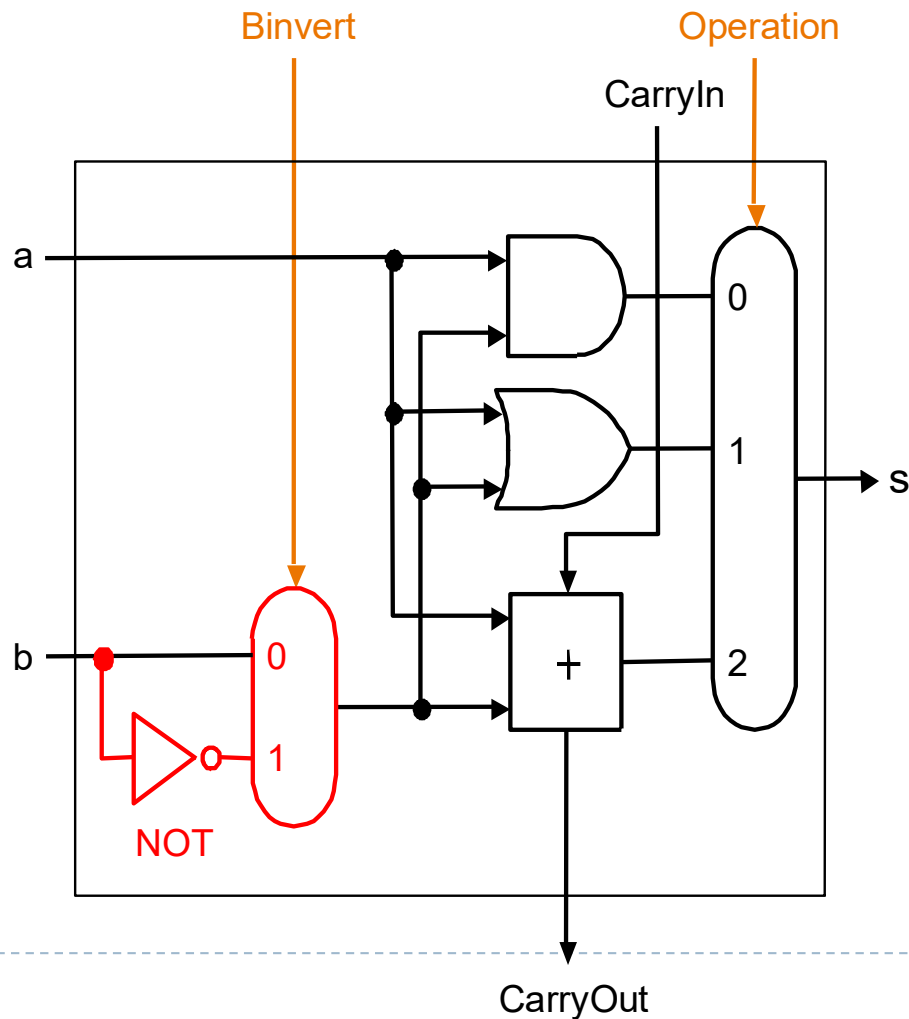
```
void alu32 (Signal *ops, //制御入力
           Word a, Word b, Signal carry_in, //入力
           Word *s) //出力
{
    // 入出力を適切につなぎながら、ALUを32回実行
}
```

1ビットALU (ver.2)

- ▶ 減算をサポート
- ▶ Binvert によるマルチプレクサを追加
- ▶ 制御入力と演算の関係は以下の通り

op[2], op[1], op[0]

Binvert	Operation	演算
0	00	AND
0	01	OR
0	10	加算
1	10	減算



1ビットALU (ver. 2)

```
void alu(Signal *ops, //制御入力 (3入力)
        Signal a, Signal b, Signal carry_in, //入力
        Signal *s, Signal *carry_out) //出力
{
    // NOT GateとMUX(2入力)を追加
    // AND Gate, OR Gate, FAを実行
    // FAへの入力を調整
    // 最後にMUX4で選択
    // 各ゲートの出力とMUX4の接続は別途Signalを用意する
}
```

32ビットALU (ver. 2)

▶ 減算をサポート

▶ 2の補数で計算

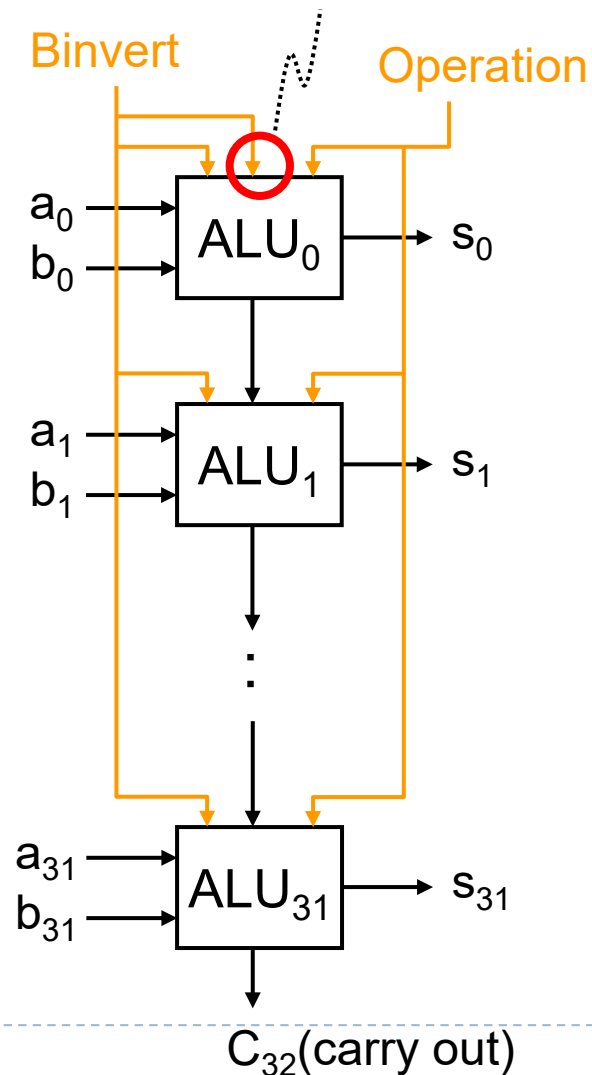
- ▶ 入力 b のビットを反転
 - Binvert が 1 の時
- ▶ a と b を加算
- ▶ 同時に 1 を加算
 - Binvert を CarryIn にすることで実現

▶ 4ビットでの例

- ▶ $0010 - 0001$
 $= 0010 + 1110 + 1$
 $= 0001$

2の補数

(= ビット反転して1足す)



32ビットALU (ver. 2) クラス

```
void alu32(Signal *ops, //制御入力 (3入力)
           Word a, Word b, //入力
           Word *s) //出力
{
    // 入出力を適切につなぎながら、ALUを32回実行
}
```

注意: 最下位ビットのCarryInにはbinvertをセット
(ver. 1にて用意していたcarry_inは削除)

1ビットALU (ver. 3)

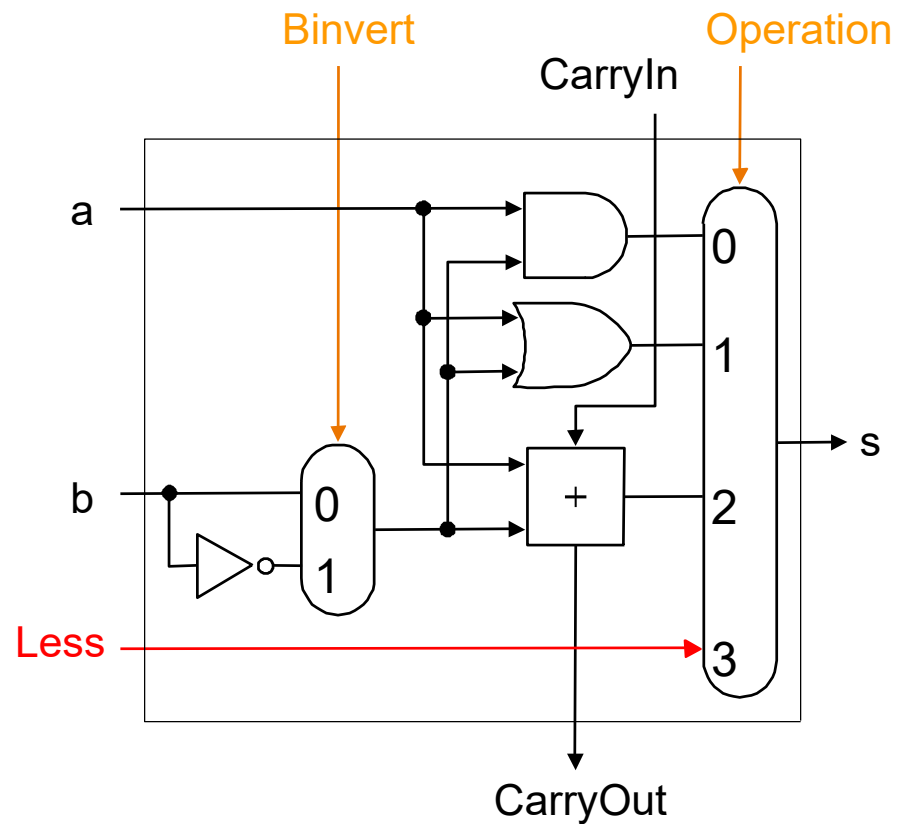
▶ sltのサポート

▶ $a < b \Rightarrow 1$

▶ 入力 Less を追加

▶ MUX で選択されると Less の値をそのまま出力

Binvert	Operation	演算
0	00	AND
0	01	OR
0	10	加算
1	10	減算
1	11	slt

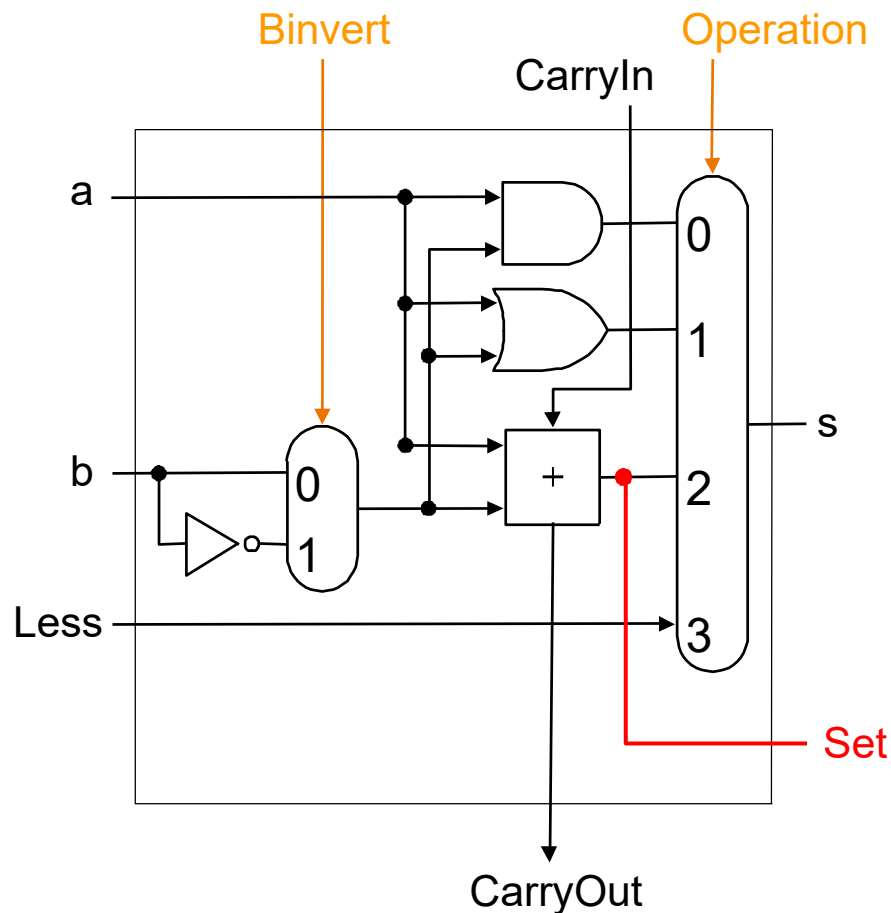
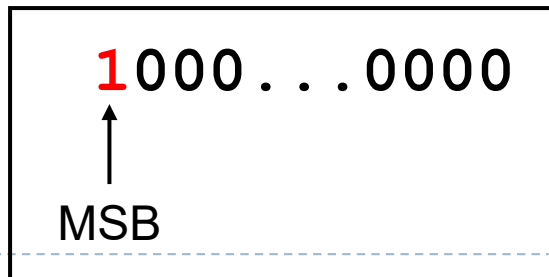


1ビットALU (ver. 3)

```
void alu(Signal *ops, //制御入力 (3入力)
        Signal a, Signal b, Signal less, //入力
        Signal carry_in, //入力
        Signal *s, Signal *carry_out) //出力
{
    // NOT GateとMUX(2入力)を追加
    // AND Gate, OR Gate, FAを実行
    // FAへの入力を調整
    // 最後にMUX4で選択
    // MUX4の入力にlessを
    // 各ゲートの出力とMUX4の接続は別途Signalを用意する
}
```

1ビットALU (ver. 3, MSB用)

- ▶ 最上位ビット(Most Significant Bit, MSB)のALUはSetも出力する
 - ▶ MSBは32ビットの数値の正負を表す(2の補数)
 - ▶ 0なら正
 - ▶ 1なら負
 - ▶ Setは $a - b$ の結果が負になると1になる

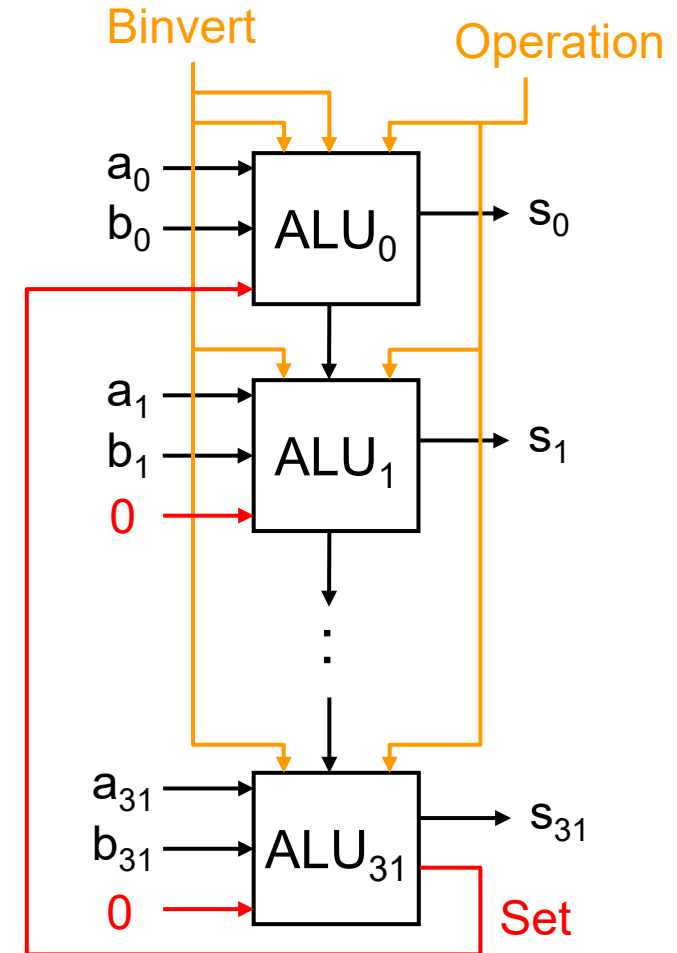


1ビットALU (ver. 3, MSB用)

```
void alu_msb(Signal *ops, //制御入力 (3入力)
             Signal a, Signal b, Signal less, //入力
             Signal carry_in, //入力
             Signal *s, Signal *carry_out
             Signal *set) //出力
{
    // NOT GateとMUX(2入力)を追加
    // AND Gate, OR Gate, FAを実行
    // FAへの入力を調整
    // FAの出力をsetにわたす
    // 最後にMUX4で選択
    // MUX4の入力にlessを
    // 各ゲートの出力とMUX4の接続は別途Signalを用意する
}
```

32ビットALU (ver. 3)

- ▶ 比較演算 (slt) をサポート
 - ▶ $a < b$ の時に 1 を出力
 - ▶ $a - b$ を計算
 - ▶ 結果が負なら Set の値は 1
 - ▶ MUX4で3 つ目の入力を入力する
 - ▶ ALU を 2 回通る (2回実行)
 - ▶ 1回目は減算を実行
 - Set の値を計算するため
 - ▶ 2回目は 3 つ目の入力を入力
 - つまり、Set の値



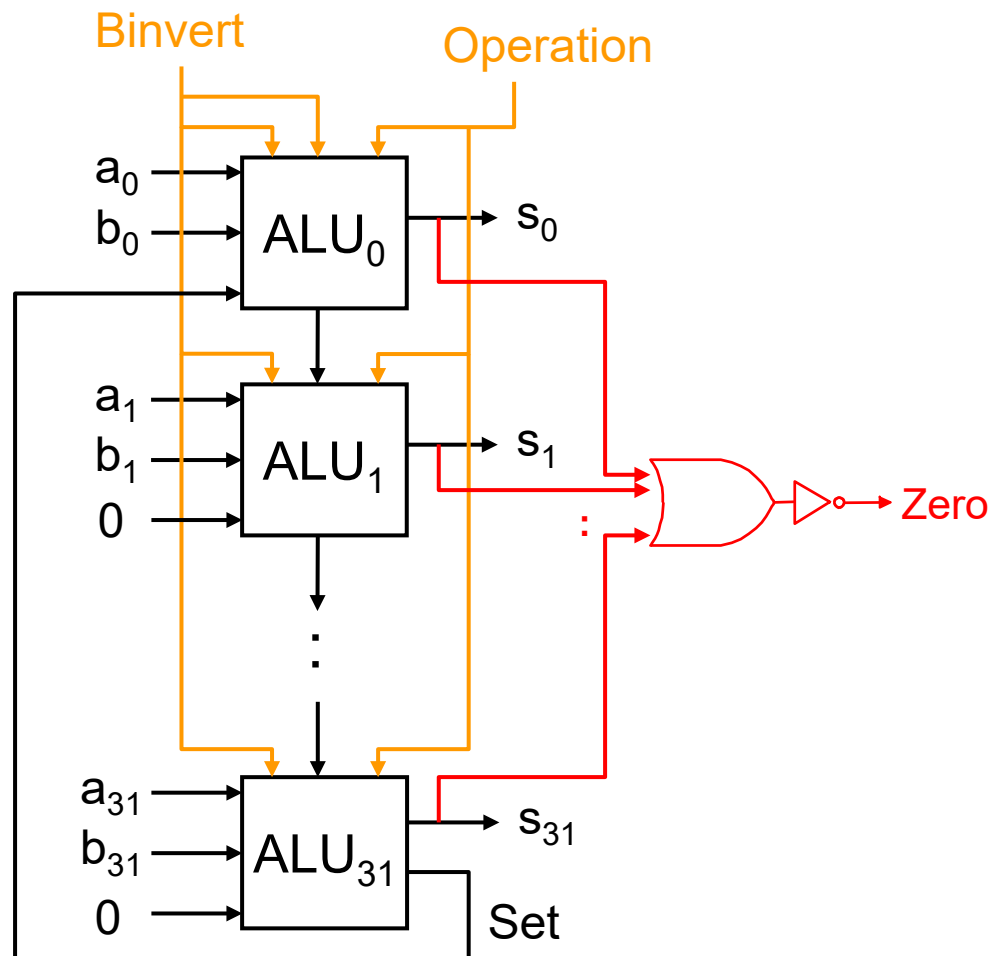
32ビットALU (ver. 3)

```
void alu32(Signal *ops, //制御入力 (3入力)
           Word a, Word b, //入力
           Word *s) //出力
{
    // 入出力を適切につなぎながら
    // ALUを31回実行
    // ALU_MSBを実行 (setの計算)
    // sltの場合を考慮して、再度0ビット目のALUを実行 (lessの値の出力)
}
```

32ビットALU (ver. 4)

▶ 等号演算をサポート

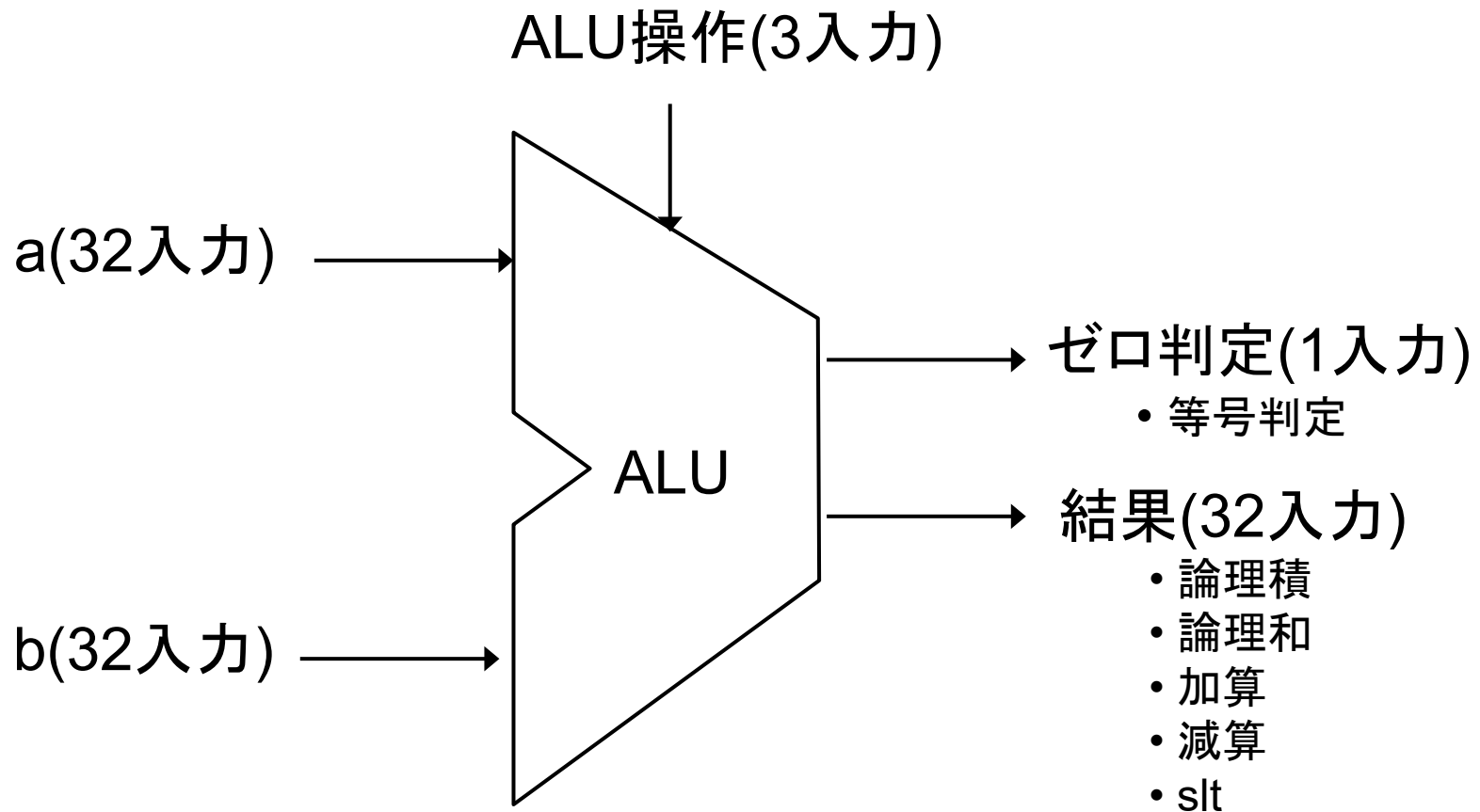
- ▶ 条件分岐等で使用
- ▶ 減算演算結果が 0 なら Zero を 1 にする
 - ▶ すべてのビットが 0 になった時
- ▶ 制御入力
 - ▶ Binvert: 1
 - ▶ Operation: 10



32ビットALU (ver. 4)

```
void alu32 (Signal *ops, //制御入力 (3入力)
           Word a, Word b, //入力
           Word *s, Signal *zero) //出力
{
    // 入出力を適切につなぎながら
    // ALUを31回実行
    // ALU_MSBを実行 (setの計算)
    // sltの場合を考慮して、再度0ビット目のALUを実行 (lessの値の出力)
    // 出力sを入力としてOR-N Gateを実行
    // 更に、その結果をNOT Gateで反転
}
```


32ビット ALU完成図



課題

課題1

- ▶ 32ビットALU(ver. 4)を作成せよ
 - ▶ 以下を作る必要がある
 - ▶ MUX、ALU、ALU_msb、ALU32 を実装
 - ▶ test関数を作成。全ての演算についてテストすること
 - ▶ 入力値の設定は適切に行なって下さい
 - ▶ ver.1, ver.2, ver.3は提出不要です

課題2

- ▶ 32ビットALUでのオーバフロー判定の追加
 - ▶ MSB の ALU にて判定可能
 - ▶ オーバフローをおこすと 1 (true) を出力
 - ▶ オーバフローは加減算を行った時に以下の条件で起こる

演算	a	b	オーバフローを起こした時の演算結果s	binvert	a (msb)	b (msb)	s (msb)
a+b	≥ 0	≥ 0	< 0				
a+b	< 0	< 0	≥ 0				
a-b	≥ 0	< 0	< 0				
a-b	< 0	≥ 0	≥ 0				

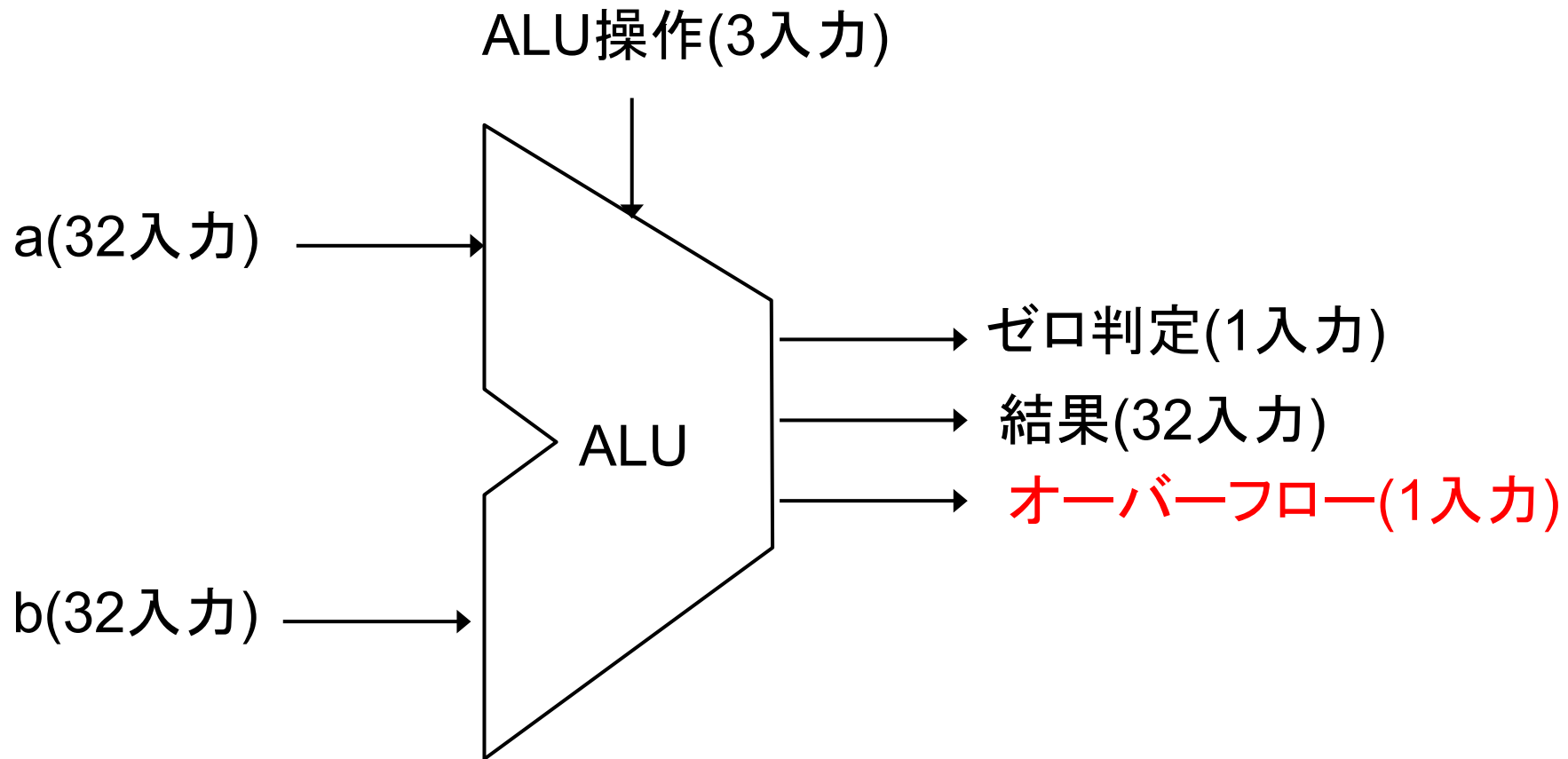
課題2 つづき

- ▶ 32ビットALUでのオーバーフロー判定の追加
 - ▶ 前ページの右の表を埋めよ
 - ▶ オーバーフローが発生条件をMSBのALUの binvert , a , b , s を用いた積和標準形で表せ

課題3（オプション課題）

- ▶ 32ビットALUにオーバフローを判定する回路を追加せよ
 - ▶ MSB 用の ALU に追加すればよい
 - ▶ オーバフローをおこすと1を出力
 - ▶ 課題2で求めた積和標準形を参考に
 - NOT Gateに加えて、AND-N GateとOR-N Gateを用いて作成せよ
 - ▶ あわせて32ビットALUの出力Signalにオーバーフロー用のものを追加

32ビット ALU完成図



課題提出

- ▶ ✕切: 1/22 (火) 23:59
- ▶ 提出物: 以下のファイルを1つのファイルに圧縮したもの
 - ▶ プログラムソース
 - ▶ シミュレータ全体ではなく、関係するコード(変更したファイル)のみ提出すること
 - ヘッダファイル等を編集していない場合は、alu.cのみで
 - ▶ 注意: 作成したプログラムは今後も使用するため、十分にテストすること
 - ▶ ドキュメント
 - ▶ 実行結果
 - ▶ 課題2
 - ▶ 感想等
- ▶ 質問等があれば compsys18@el.gsic.titech.ac.jp まで
 - ▶ 課題のレポートやコメントに書かれていると、返信が遅くなります

参考: 積和標準形

- ▶ リテラル: 変数またはその否定 (例: a, \bar{a})
- ▶ 積和標準形: 論理式をリテラルの積の和で表したもの
 - ▶ (例: $\bar{a}bc + a\bar{b}c + ab\bar{c}, ab\bar{c} + c$)
- ▶ 和積標準形: 論理式をリテラルの和の積で表したもの
 - ▶ (例: $(\bar{a} + b + c)(a + \bar{b} + c)(a + b + \bar{c})$)
- ▶ 論理回路を実現するには、論理関数を簡単な積和標準形や和積標準形で書ければよい
 - ▶ 積和: $(\text{not} \rightarrow) \text{and} \rightarrow \text{or}$
 - ▶ 和積: $(\text{not} \rightarrow) \text{or} \rightarrow \text{and}$
 - ▶ Cf: 論理圧縮、QM(Quine-McCluskey)法