

# 計算機構成論 第10回 —算術演算の実行(1)—

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

# 講義内容

---

## ■ 算術演算の実行

- ➡ ■ 2進数の加算・減算・AND/OR演算  
シフト演算、符号拡張
- 算術演算ユニット(ALU)

# 2進数の加算 復習

## ■ 正数 + 正数

$$\begin{array}{r} 00001111 \quad 15_{10} \\ + 00000010 \quad 2_{10} \\ \hline 00010001 \quad 17_{10} \end{array}$$

正しい

$$\begin{array}{r} 01111110 \quad 126_{10} \\ + 00000010 \quad 2_{10} \\ \hline 10000000 \quad 128_{10} \end{array}$$

桁あふれ  
(オーバーフロー: overflow)

✖ これは-128

どのように対処するかは、  
プログラミング言語等による

# 確認問題

---

- $00011110_2 + 00010010_2$  を  
2進数の筆算で計算し、10進数で答えよ。
- $01100110_2 - 00011000_2$  の結果を  
10進数で答えよ。



# 2進数の負数表現 -2の補数- 復習

10進数	2進数
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
-0	0000

10進数	2進数
8	-
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000

4ビットでは  
表現不可能

1の補数+1

## 2進数の減算 復習

■ 正数 + 負数 を行えば良い

$$\begin{array}{r} 00001111 \quad 15_{10} \\ + \quad 11111110 \quad -2_{10} \\ \hline 00001101 \quad 13_{10} \end{array} \quad \text{正しい}$$

この繰り上がりは無視

$$\begin{array}{r} 00001111 \quad 15_{10} \\ + \quad 11101101 \quad -19_{10} \\ \hline 11111100 \quad -4_{10} \end{array} \quad \text{正しい}$$

正数 + 負数ではオーバーフローは発生しない

# 2進数の減算 復習

## ■ 負数 + 負数 の場合

$$\begin{array}{r} 11110001 \\ + 11111110 \\ \hline 11101111 \end{array} \quad \begin{array}{l} -15_{10} \\ -2_{10} \\ -17_{10} \end{array} \quad \text{正しい}$$

$$\begin{array}{r} 10000011 \\ + 11111100 \\ \hline 01111111 \end{array} \quad \begin{array}{l} -125_{10} \\ -4_{10} \\ -129_{10} \end{array} \quad \text{オーバーフロー}$$



これは127



# AND, OR演算

---

## ■ AND演算

$$\begin{array}{r} 11110001 \\ \& 00110110 \\ \hline 00110000 \end{array}$$

## ■ OR演算

$$\begin{array}{r} 11110001 \\ | 00110110 \\ \hline 11110111 \end{array}$$

# 確認問題

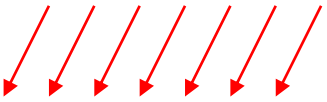
- $01010101_2 \& 11110000_2$  を計算せよ。
- $01010101_2 | 11110000_2$  を計算せよ。
- レジスタ  $\$s0$  に  $15_{10}$  が入っている。  
命令 `sll $s0, $s0, 2` を実行すると、  
 $\$s0$  の中身はどうなるか、2進数で答えよ。
- レジスタ  $\$s0$  に  $-2_{10}$  が入っている。  
命令 `sra $s0, $s0, 1` を実行すると、  
 $\$s0$  の中身はどうなるか、2進数で答えよ。  
※sraは右算術シフトを行う命令である。



# シフト演算 (shift operation) 復習

## ■ 2進数のビットを左右にずらす演算

$$00001100_2 = 8+4 = 12_{10}$$




$$00011000_2 = 16+8 = 24_{10}$$

最上位ビット  
は捨てる

最下位ビット  
には0が入る

$$00001100_2 = 8+4 = 12_{10}$$



$$00000110_2 = 4+2 = 6_{10}$$

# シフト演算 (shift operation) 復習

## ■ 左端、右端に注意

左に1ビット  
シフト

$$11000000_2 = 128 + 64 = 192_{10}$$



$$10000000_2 = 128 = 128_{10}$$



2倍にならない

右に1ビット  
シフト

$$00001001_2 = 8 + 1 = 9_{10}$$



$$00000100_2 = 4 = 4_{10}$$



1/2倍  
(小数点以下  
切り捨て)

# 算術シフト 復習

## ■ 符号付きの場合

### ■ 算術シフト

右に1ビット  
シフト

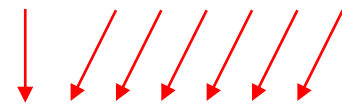
$$11111000_2 = -8_{10}$$



$$11111100_2 = -4_{10}$$

左に1ビット  
シフト

$$11111000_2 = -8_{10}$$



$$11110000_2 = -16_{10}$$

※論理左シフトとみなしても良い


# 論理シフト 復習

## ■ 符号なしの場合

### ■ 論理シフト

右に1ビット  
シフト


$$11111000_2 = 248_{10}$$


$$01111100_2 = 124_{10}$$

符号ビットを  
考慮しない

左に1ビット  
シフト

$$01111000_2 = 120_{10}$$


$$11110000_2 = 240_{10}$$

# 確認問題

- (1) 命令 `addi $s0, $zero, -1` を機械語に直せ。

※ `addi` は I 形式で opcode は  $8_{16}$

※ `$s0` は  $10_{16}$  番, `$zero` は  $0_{16}$  番

op	rs	rt	immediate
6bit	5bit	5bit	16bit

- (2) (1) の命令の結果、`$s0` の中身はどうなるか、2 進数で示せ。





# 2進数の符号拡張 復習

- より多いビット数で同じ値を表現する方法
  - e.g., 8ビット2進数→16ビット2進数
- 新しい値の上位ビットを、元の値の最上位ビットで埋めればOK
  - 負数は2の補数表現が前提

8bit                      00001111                       $15_{10}$   
16bit    000000000000000001111

8bit                      11110001                       $-15_{10}$   
16bit    111111111111111110001

# 講義内容

---

## ■ 算術演算の実行

- 2進数の加算・減算・AND/OR演算

- シフト演算、符号拡張

➡ ■ 算術演算ユニット(ALU)

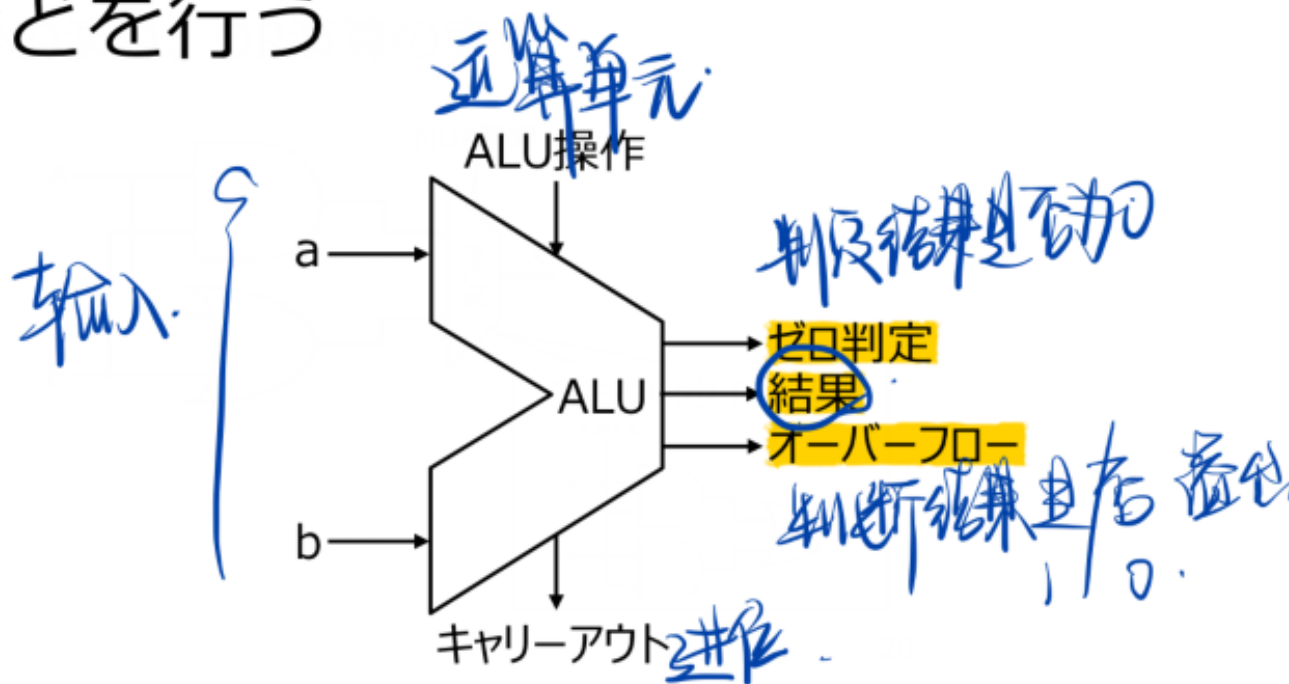
# 算術演算ユニット

■ ALU (arithmetic logic unit) 算術演算ユニット

■ 算術演算 (加算、減算)

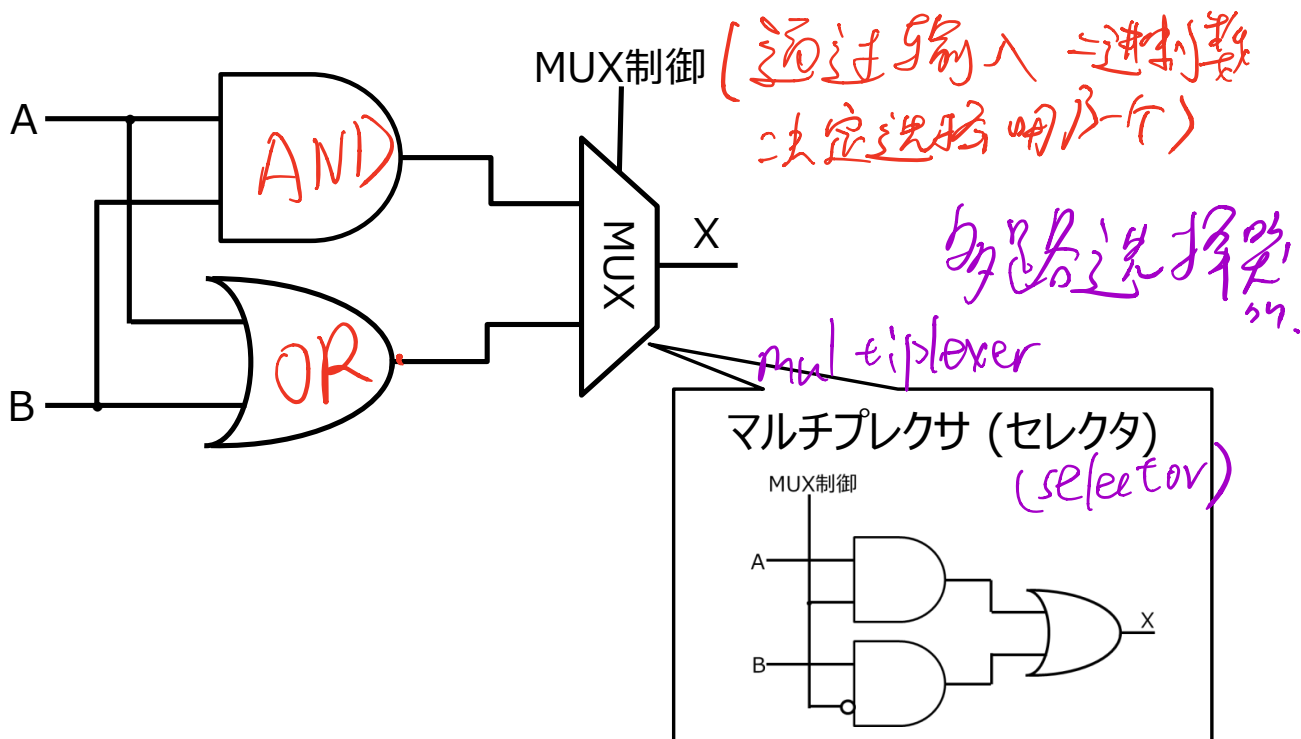
■ 論理演算 (AND, OR)

などを行う



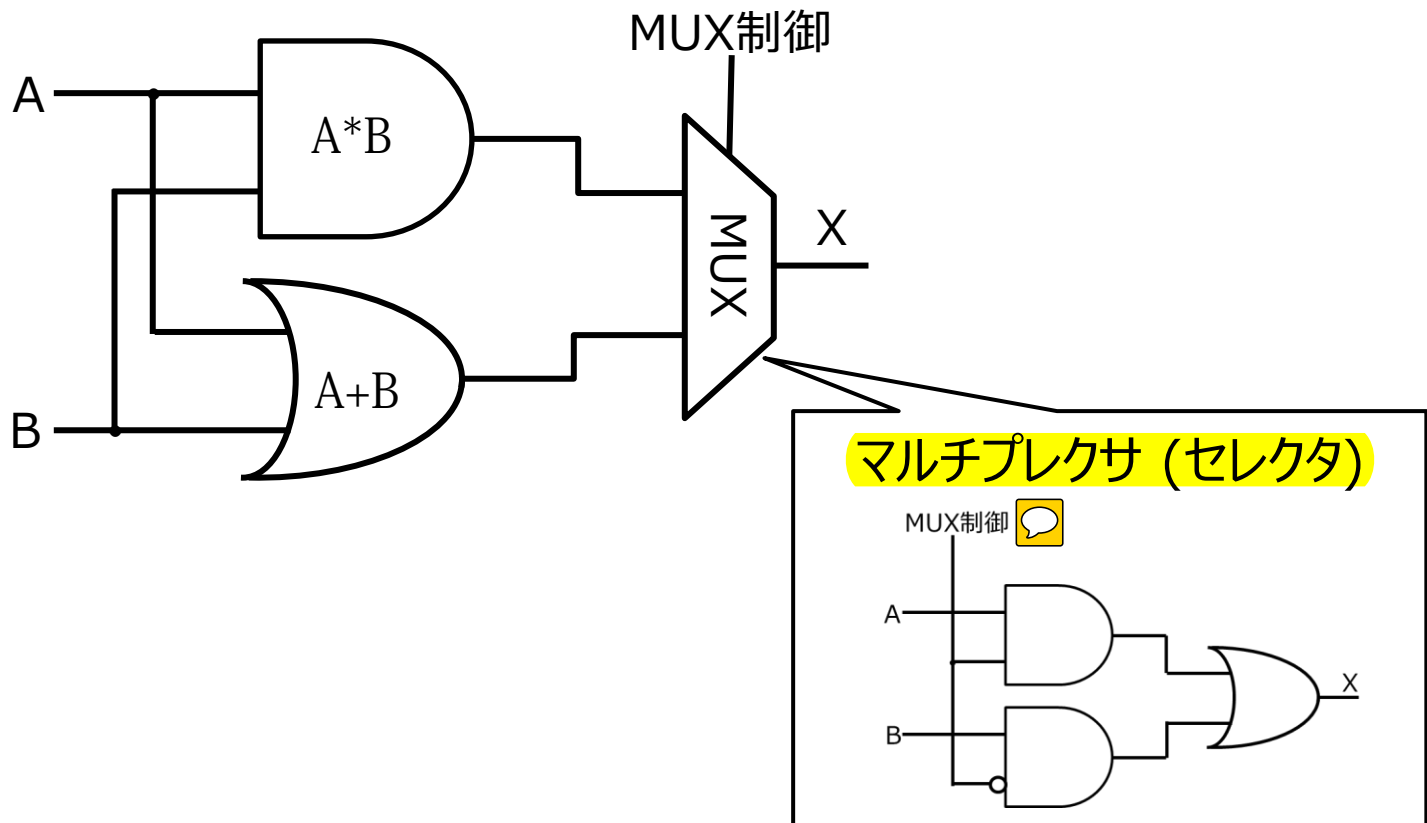
# 論理演算の実現

## ■ AND演算、OR演算の実現



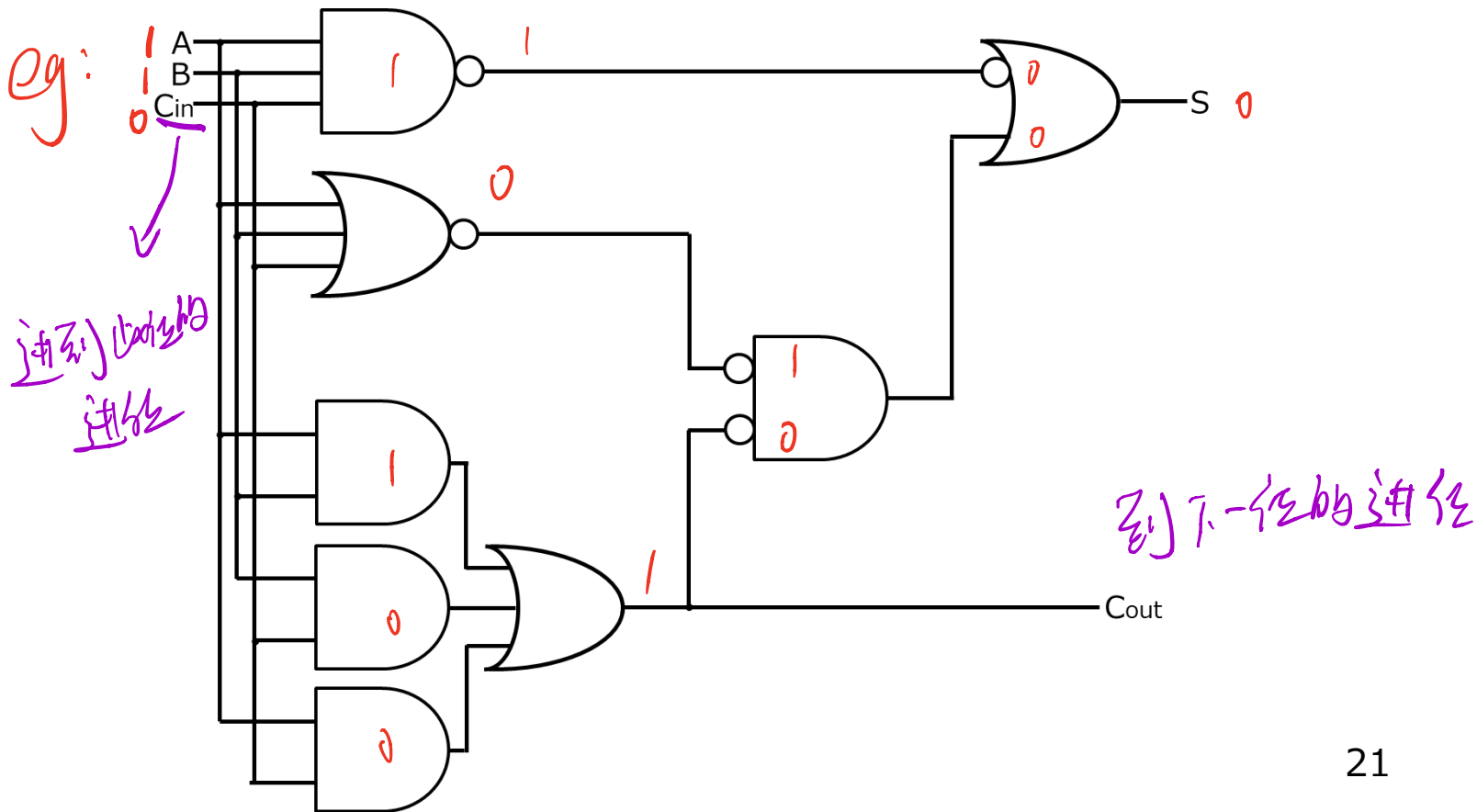
# 論理演算の実現

## ■ AND演算、OR演算の実現



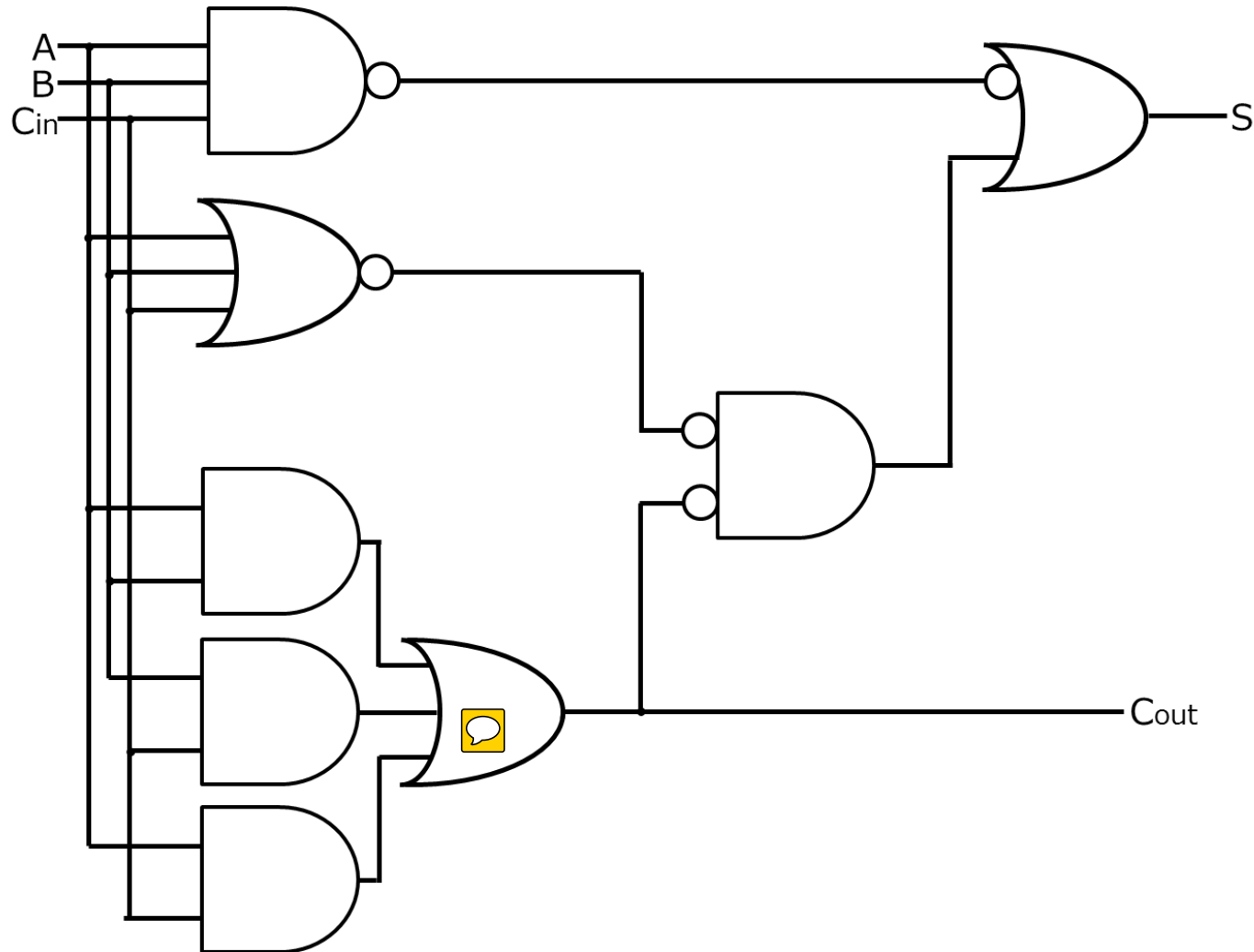
# 1ビット加算器の実現

## ■ 全加算器 (下位からの繰り上げあり)



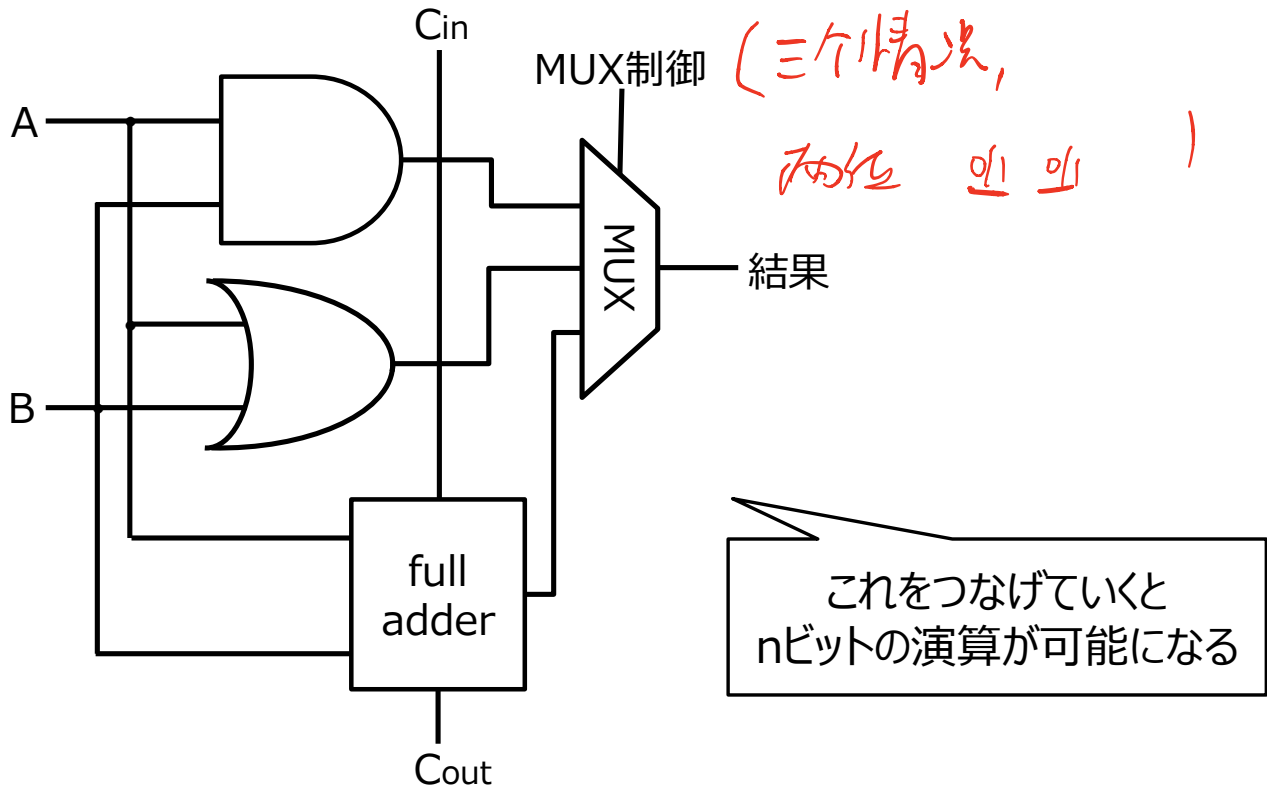
# 1ビット加算器の実現

## ■ 全加算器 (下位からの繰り上げあり)

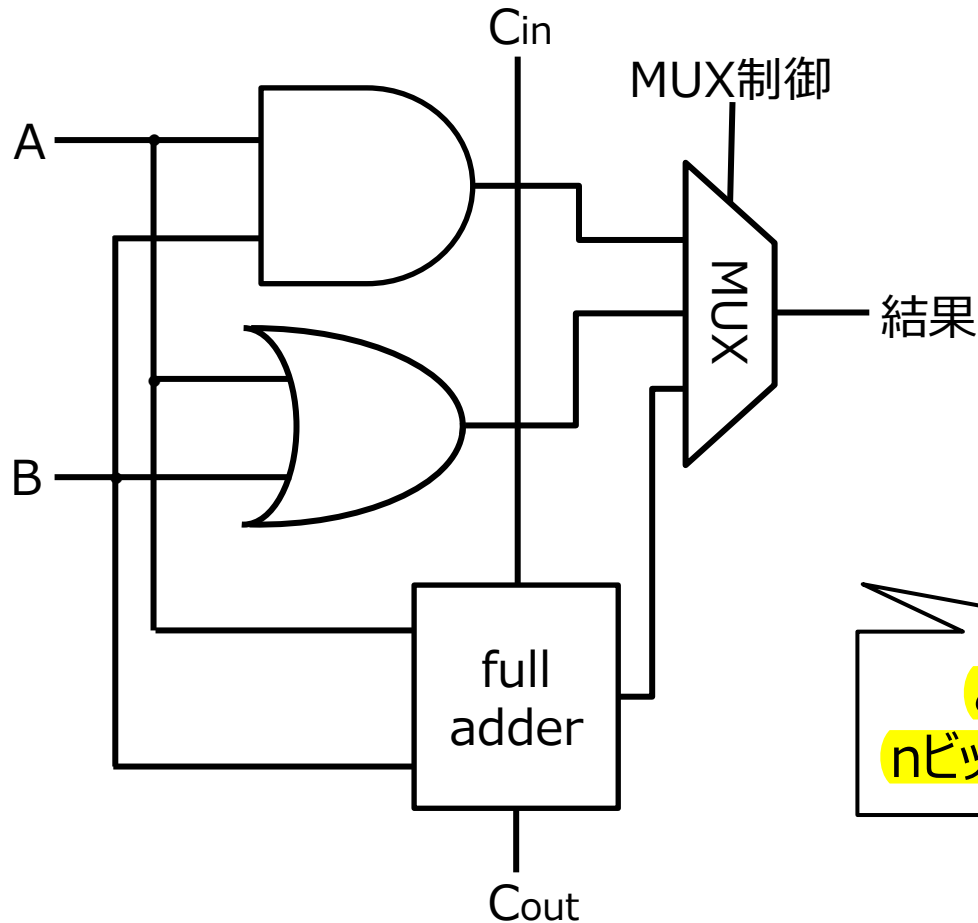




# 論理演算と加算が可能なALU



# 論理演算と加算が可能なALU



これをつなげていくと  
nビットの演算が可能になる

# 減算器の実現

---

- 「入力値の2の補数を取る」という指定ができる制御信号を導入すれば良い  
→ あとは加算器で加算する

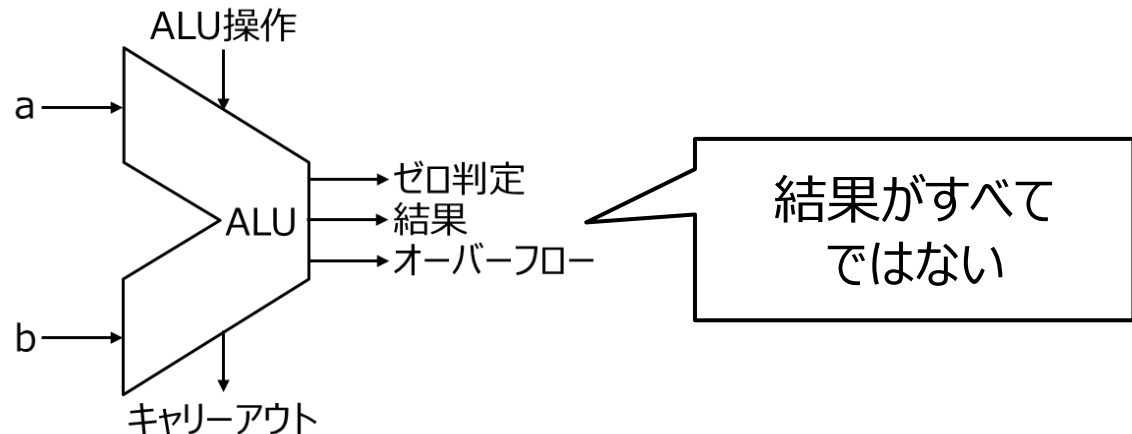
-  各ビット反転

-  最下位ビットに1を加算

# 加減算におけるオーバーフロー判定

操作	オペランドA	オペランドB	結果
A+B	非負	非負	負
A+B	負	負	非負
A-B	非負	負	負
A-B	負	非負	非負

→実際には、もっと簡単に判定可能



# 加減算におけるオーバーフロー判定

2の補数を使えば、減算も加算として扱えるので、加算のみ考慮

## ① 正 + 正

$$\begin{array}{r} 00111111 \quad 63 \\ + 00000011 \quad 3 \\ \hline 001000010 \quad 66 \end{array}$$

## ② 負 + 負

$$\begin{array}{r} 11000001 \quad -63 \\ + 11111101 \quad -3 \\ \hline ① 110111110 \quad -66 \end{array}$$

無視

先不考虑符号位 < 结果正常加法算  
和结果的  
符号位和符号位的进位不同  
⇒ overflow (异常)

# 加減算におけるオーバーフロー判定

最上位への繰り上がり、  
最上位からの繰り上がりに着目

## ① 正 + 正

$$\begin{array}{r} 00111111 \quad 63 \\ + 00000011 \quad 3 \\ \hline 001000010 \quad 66 \end{array}$$

↑↑  
00

## ② 負 + 負

$$\begin{array}{r} 11000001 \quad -63 \\ + 11111101 \quad -3 \\ \hline (1)10111110 \quad -66 \end{array}$$

↑↑  
11

# 加減算におけるオーバーフロー判定

最上位への繰り上がり、最上位からの繰り上りに着目

## ① 正 + 正

$$\begin{array}{r} 01111111 \quad 127 \\ + 00000011 \quad 3 \\ \hline 010000010 \quad -126 \end{array}$$

↑↑  
0 1

## ② 負 + 負

$$\begin{array}{r} 10000001 \quad -127 \\ + 11111101 \quad -3 \\ \hline 101111110 \quad 126 \end{array}$$

↑↑  
1 0

異なるとオーバーフロー

2つのビットの**比較**を行えば判定可能  
→ XORゲート(排他的論理和)が使える

# MIPSにおけるオーバーフロー対応

- 加算(add)、即値加算(addi)、減算(sub)時にオーバーフロー例外が発生
- 符号なし加算(addu)、符号なし即値加算(addiu)、符号なし減算(subu)は、オーバーフローが発生しても例外にならない
  - 通常、符号なし整数はメモリアドレスに使用される
- オーバーフローを認識するかどうかは状況依存
  - **状況**によって、認識したり、無視できるように命令を選択
- 例外プログラムカウンタ(**EPC**)に例外を起こした命令のアドレスが格納される



# 確認問題

---

- CPUの中に存在する様々な算術演算を行う回路を、(1)と呼ぶ。
- 加減算におけるオーバーフローを判定するには、(2) と、(3) を見れば良い。



# 参考文献

---

## ■ コンピュータの構成と設計 上 第5版

David A.Patterson, John L. Hennessy 著、  
成田光彰 訳、日経BP社

## ■ コンピュータの構成と設計 下 第5版

David A.Patterson, John L. Hennessy 著、  
成田光彰 訳、日経BP社

## ■ 山下茂 「計算機構成論 1」 講義資料