

# 計算機構成論

## Lecture 3

### 計算機での数の表現 2進数

2023年度春学期

情報理工学部 Rクラス担当

越智裕之

# 内容

- 2進数の復習（とても大事なので）
  - 2進数
    - 基数変換 ⇐ NEXT
  - 2の補数
    - 正負反転
    - 符号拡張
    - シフト
- 教材：教科書2.4節（一部内容は違う）

# 基本のチェック：2進数、10進数、16進数

\* 大丈夫？ 忘れてる人は、寝たらだめ！

10進数から指定されたビット幅の2進数へ変換せよ

- $17_{10} \rightarrow$   (5ビットの2進数)
- $18_{10} \rightarrow$  (5ビットの2進数)
- $0_{10} \rightarrow$  (6ビットの2進数)

16進数から2進数へ変換せよ

- $\text{eca8}_{16} \rightarrow$

2進数から16進数へ変換せよ

- $11010101 \rightarrow$

# 2進数, 16進数, 10進数

復習？

2進数	16進数	10進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

# 符号なし整数の2進数表現 (1/2)

復習？

$2^{n-1}$                        $2^3$   $2^2$   $2^1$   $2^0$       桁に重みを割り当て

--	--	--	--	--	--	--	--	--	--

- 0, 1 の並びで整数を表す方法
- 2 を  (radix/base number) として整数を表す方法
  - 0000, 0001, 0010, 0011, ..., 1110, 1111

正の整数Xに対するn桁の2進数

$$(x_{n-1} x_{n-2} \dots x_1 x_0)_2 \quad x_i \in \{0, 1\}$$

$$X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12 + x_0$$

各桁をビット(bit: binary digit)

最上位桁を

最下位桁を

と呼ぶ

# 符号なし整数の2進数表現 (2/2)

復習？

10進数	2進数	16進数
0	00000	00
1	00001	01
2	00010	02
3	00011	03
4	00100	04
5	00101	05
6	00110	06
7	00111	07
8	01000	08
9	01001	09
10	01010	0a
11	01011	0b
12	01100	0c
13	01101	0d
14	01110	0e
15	01111	0f
16	10000	10
17	10001	11

nビットだと0から  までの数  
を表せる

8ビットの例

0 から 255 まで

00000000

11111111

00101010

など

$2^7$   $2^6$   $2^5$   $2^4$   $2^3$   $2^2$   $2^1$   $2^0$   
☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

# 符号付き整数の表現

復習？

- 最上位の桁に負の重みを持たせる表現法
- $n$  ビットで、 $-2^{n-1}$  から  $2^{n-1}-1$  までの数を表す
- 8 ビットの例
  - $-128$  から  $127$  まで
  - 10000000, 01111111, 10000001, 11111111

$-2^7$   $2^6$   $2^5$   $2^4$   $2^3$   $2^2$   $2^1$   $2^0$   
■ □ □ □ □ □ □ □

$$1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 = -2^7 = -128$$

$-2^{n-1}$

$$1\ 0\ 0\ 0\ 0\ 0\ 0\ 1 = \boxed{\phantom{00000000}}$$

$$0\ 1\ 0\ 0\ 0\ 0\ 0\ 1 = 2^6 + 2^0 = 65$$

$$0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 = \boxed{\phantom{00000000}}$$

$2^{n-1}-1$

$$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 = \boxed{\phantom{00000000}}$$

最上位ビットは、符号ビットと呼ばれる

# 演習問題 その①：2進から10進への変換

教科書p75

次の32ビットの2の補数表現で表された値を10進数で表せ

1111 1111 1111 1111 1111 1111 1111 1100<sub>2</sub>

2の補数表現の実際の値の計算法(32ビットの場合)

$$(x_{31} \times (-2^{31})) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

ここで、 $x_i$ は、数値 $x$ の $i$ 番目のビット

答)

$$\begin{aligned} & (1 \times (-2^{31})) + (1 \times 2^{30}) + (1 \times 2^{29}) + \dots + (0 \times 2^1) + (0 \times 2^0) \\ = & -2^{31} + 2^{30} + 2^{29} + \dots + 2^2 \\ = & -2,147,483,648_{10} + 1,073,741,824_{10} + 536,870,912_{10} + \dots + 4_{10} \\ = & -2,147,483,648_{10} + 2,147,483,644_{10} \\ = & -4_{10} \end{aligned}$$

馬鹿正直にやらない方法は？



以下のポイント知らなかった人は、自分の理解でメモしてください

ポイント: 1が続く2進数の値の求め方(以下のX, Yの値は?)

ポイントの具体例: all 1の2進数の値は?

Y = 1111 1111 1111 1111 1111 1111 1111 1100<sub>2</sub>

X = 0111 1111 1111 1111 1111 1111 1111 1100<sub>2</sub>

## 演習問題 その②： 2進数から16進数へ変換

次の16進数を2進数に変換せよ

eca8 6420<sub>16</sub>

次の2進数を16進数に変換せよ

0001 0011 0101 0111 1001 1011 1101 1111<sub>2</sub>

## 演習問題 その③:10進数から2進数へ変換

−1023<sub>10</sub>を32ビットの2の補数表現の2進数に変換せよ。

答)

1023を作ってから後述する方法で、正負の反転が普通だが、以下のような方法もある

−1023 = −1024 + 1 = −2<sup>10</sup> + 2<sup>0</sup> に気づけば、

1111 1111 1111 1111 1111 1100 0000 0001<sub>2</sub>

なぜなら、以下のように1が先頭から並んでいるものは−2<sup>k</sup>となる（スライド8のポイント）

$$Y = 1111\ 1111\ 1111\ 1111\ 1111\ 1100\ 0000\ 0000_2 = -2^{10}$$

↑  
2<sup>10</sup>

( Y + 2<sup>10</sup> = 0 (スライド8のポイント)となるので)

# 内容

- 2進数の復習（とても大事なので）
  - 2進数
    - 基数変換
  - 2の補数
    - 正負反転 ⇐ NEXT
    - 符号拡張
    - シフト
- 教材：教科書2.4節（一部内容は違う）

# 2の補数とは

## 定義

$n$ ビットの $X$ の2の補数が $Y$   $\Leftrightarrow X+Y = 2^n$

## 求め方

各ビット反転して、さらに+1を加えると2の補数となる

ミニクイズ：この求め方で定義を満たすことを説明せよ

# 2の補数について重要なこと

正負の変換 は、2の補数を求めることと同じ (演習⑤)

32bitの場合

~  (2,147,483,647) まで表現可能 ( $2^{32}$ 通り)

0000 0000 0000 0000 0000 0000 0000 0000	=	0
0000 0000 0000 0000 0000 0000 0000 0001	=	1
0000 0000 0000 0000 0000 0000 0000 0010	=	2
....		
0111 1111 1111 1111 1111 1111 1111 1101	=	2,147,483,645
0111 1111 1111 1111 1111 1111 1111 1110	=	2,147,483,646
0111 1111 1111 1111 1111 1111 1111 1111	=	2,147,483,647
1000 0000 0000 0000 0000 0000 0000 0000	=	-2,147,483,648
1000 0000 0000 0000 0000 0000 0000 0001	=	-2,147,483,647
1000 0000 0000 0000 0000 0000 0000 0010	=	-2,147,483,646
....		
1111 1111 1111 1111 1111 1111 1111 1101	=	-3
1111 1111 1111 1111 1111 1111 1111 1110	=	-2
1111 1111 1111 1111 1111 1111 1111 1111	=	-1

2進数

10進数

## 演習問題 その④

教科書p77

$2_{10}$ および $-2_{10}$ を正負反転せよ。

答)

$$2_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2$$

各ビットを反転させ、1を足す。

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 \\ +) \qquad \qquad \qquad 1_2 \\ \hline 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 \end{array}$$

$$-2_{10} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2$$

各ビットを反転させ、1を足す。

$$\begin{array}{r} 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 \\ +) \qquad \qquad \qquad 1_2 \\ \hline \text{元に戻る } 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 \end{array}$$

# 自己確認問題

32ビットの2進数で-17を求めよ

- ①  $17_{10}$ を求めて、前のページのように正負を反転する
- ② -1は、all 1であることは知っているので、それから16を引く



## 演習問題 その⑤

講義で説明したことを理解しておいてください

与えられた数の正負の反転は、2の補数を求めればよい。  
つまり、**ビット毎の否定を行い、最下位ビットに 1 を加えれば良い**  
ということになる。この操作の正当性を示せ  
問題を言い換えると、  
 $X$ の2の補数が2進数の定義（符号付）に従うと $-X$ となることを示せ。  
という問題です。（つまり、定義を知っていると簡単です）

### 略証

$X + \bar{X}$  は全ビットが  である

つまり、符号付き2進数と見れば  $X + \bar{X} =$

よって、 $\bar{X} + 1 = -X$  が成り立つ

$X$	01101010
$\bar{X}$	10010101
<hr/>	
$X + \bar{X}$	11111111

# 演習問題その⑤のちゃんとした説明

$n$ ビットの符号付き2進数 $X, Y$ の各ビットを

$$X = (x_{n-1}, x_{n-2}, \dots, x_0), Y = (y_{n-1}, y_{n-2}, \dots, y_0)$$

と表すと、符号付き2進数として表す数は

$$X = -2^{n-1}x_{n-1} + 2^{n-2}x_{n-2} + \dots + 2^0x_0$$

$$Y = -2^{n-1}y_{n-1} + 2^{n-2}y_{n-2} + \dots + 2^0y_0$$

である。ここで、

$$s_X = -2^{n-1}x_{n-1}, f_X = 2^{n-2}x_{n-2} + \dots + 2^0x_0$$

$$s_Y = -2^{n-1}y_{n-1}, f_Y = 2^{n-2}y_{n-2} + \dots + 2^0y_0$$

とおくと、 $X = s_X + f_X, Y = s_Y + f_Y$ である。

以下、 $Y$ が $X$ の2の補数であるとする。

**(1)  $x_{n-1} = 0, f_X > 0$  の場合 ( $X = f_X$ )**

$Y = \bar{X} + 1$ を求める際、符号ビットへの繰り上がりが生じないから、 $s_Y = -2^{n-1}, f_Y = 2^{n-2}\overline{x_{n-2}} + \dots + 2^0\overline{x_0} + 1$ となる。よって  $f_X + f_Y = 2^{n-1}$  から  $f_Y = 2^{n-1} - f_X$  である。よって、 $Y = s_Y + f_Y = -2^{n-1} + 2^{n-1} - f_X = -X$

**(2)  $x_{n-1} = 1, f_X > 0$  の場合 ( $X = -2^{n-1} + f_X$ )**

(1) と同様の議論から、 $Y = -X$  が言える。

**(3)  $x_{n-1} = 0, f_X = 0$  の場合 ( $X = 0$ )**

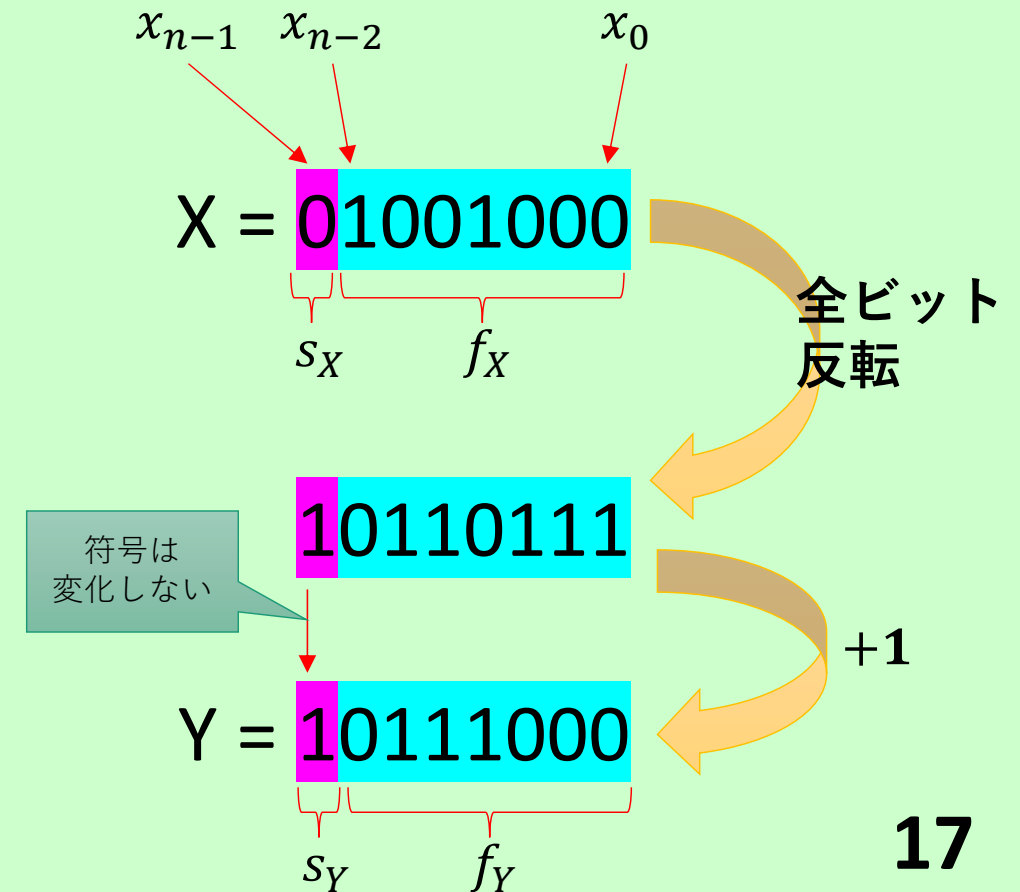
この時は、 $Y = \bar{X} + 1$ を求める際に符号ビットへの

繰り上がりが生じ、 $f_Y = 0, s_Y = 0$  となる（最上位桁からの繰り上がりは無視する）。よって  $Y = 0$  となるから、 $Y = -X$  が成り立つ。

**(4)  $x_{n-1} = 1, f_X = 0$  の場合 ( $X = -2^{n-1}$ )**

この時も、 $Y = \bar{X} + 1$ を求める際に符号ビットへの繰り上がりが生じ、 $f_Y = 0, s_Y = -2^{n-1}$ となる（最上位桁からの繰り上がりは無視する）。よって  $Y = -2^{n-1}$  となり、 **$Y = -X$  とはならない。**

## 場合分け(1)の具体例



# 内容

- 2進数の復習（とても大事なので）
  - 2進数
    - 基数変換
  - 2の補数
    - 正負反転
    - 符号拡張 ⇐ NEXT
    - シフト
- 教材：教科書2.4節（一部内容は違う）

# 符号拡張の方法

16ビット2進数 → 32ビット2進数へ変換(符号拡張という)

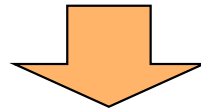
短い方の数値を長い方の右側の部分に単純にコピーし、  
長い方の余る部分を短い方の最上位ビットつまり符号ビットで埋める

# 演習問題 その⑥:符号拡張の方法

教科書p77

16ビット2進数の $2_{10}$ および $-2_{10}$ を32ビットの2進数に変換せよ。

答) 16ビット表現の2 : 0000 0000 0000 0010<sub>2</sub>  
16ビット表現の-2 : 1111 1111 1111 1110<sub>2</sub>



32ビット表現の2 :  
0000 0000 0000 0000 0000 0000 0000 0010<sub>2</sub>

32ビット表現の-2 :  
1111 1111 1111 1111 1111 1111 1111 1110<sub>2</sub>

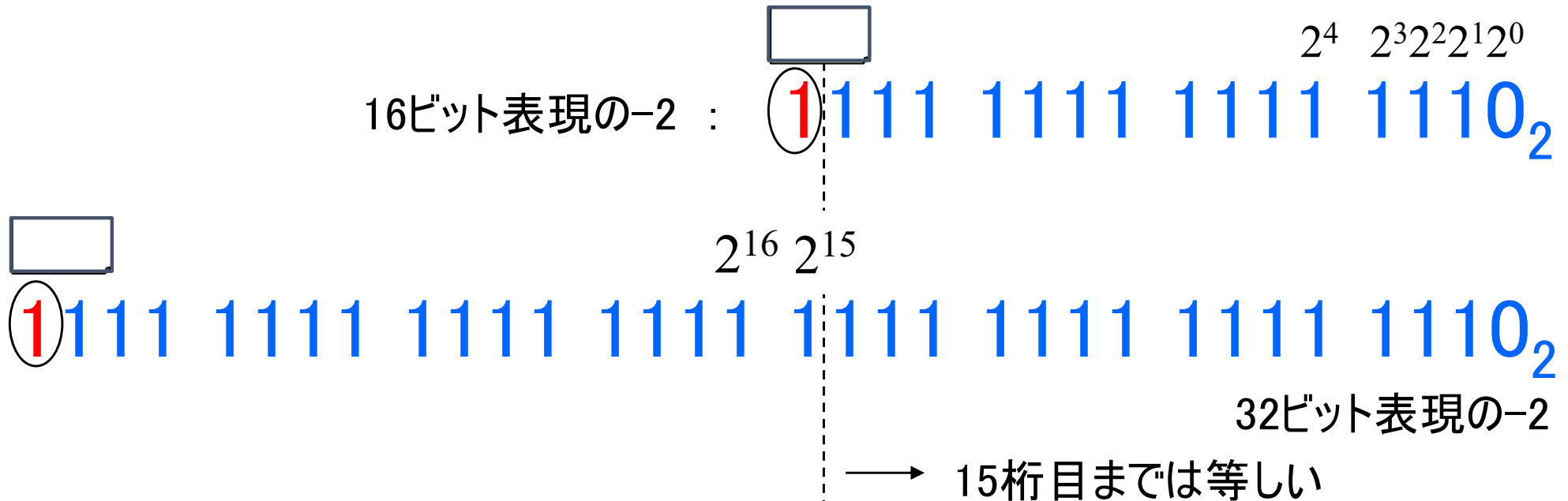
# 符号拡張の方法の正当性

16ビット2進数 → 32ビット2進数へ変換(符号拡張という)

短い方の数値を長い方の右側の部分に単純にコピーし、  
長い方の余る部分を短い方の最上位ビットつまり符号ビットで埋める

**ミニクイズ:** 前のページの操作(=上の説明)が正当であることを説明せよ。

• 正の場合は自明、負の場合は？



16桁より上も、 $-2^{15} = -2^{31} + 2^{30} + 2^{29} + \dots + 2^{16} + 2^{15}$  なので、OKである。

なぜなら、 $2^{31} = 2^{30} + 2^{29} + \dots + 2^{16} + 2^{15} + 2^{15}$  なので

# 理解度チェック

$-2^{31}$   $2^{16}$   $2^{15}$   
**1**111 1111 1111 1111 1000 0000 0000 0000<sub>2</sub>

この2進数の値は？

# 内容

- 2進数の復習（とても大事なので）
  - 2進数
    - 基数変換
  - 2の補数
    - 正負反転
    - 符号拡張
    - シフト ⇐ NEXT
- 教材：教科書2.4節（一部内容は違う）



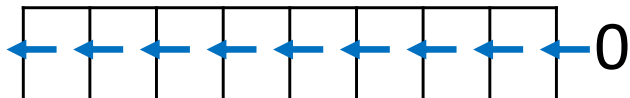
# シフト演算 (論理シフト演算)

## 論理シフト演算

- ・符号なし整数を想定し、語中のビットを左または右にずらす

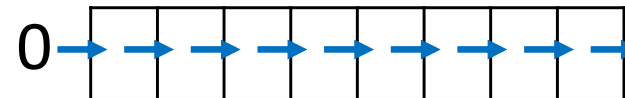
### 論理左シフト

- ・語中のビットを左にずらす
- ・空いたビットには0を挿入



### 論理右シフト

- ・語中のビットを右にずらす
- ・空いたビットには0を挿入



#### 3ビット左に論理シフト

000**01101** (13)

⇒ **01101000** (104)

C言語的な記述だと

$13_{10} \ll 3 = 104_{10}$

#### 1ビット右に論理シフト

**00011010** (26)

⇒ **00001101**

C言語的な記述だと

$26_{10} \gg 1 = \text{}$

10000001 (129)を左に1ビットシフトすると？

⇒

10000001 (129)を右に1ビットシフトすると？

⇒

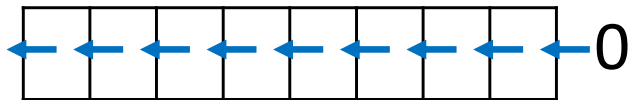
# シフト演算 (算術シフト演算)

## 算術シフト演算



- ・2の補数を想定し、語中のビットを左または右にずらす

### 算術左シフト (P.30に補足説明あり)


- ・語中のビットを左にずらす
- ・空いたビットには0を挿入



1ビット左に算術シフト

11101101 (-19)  値が2倍になった  
⇒ 11011010 

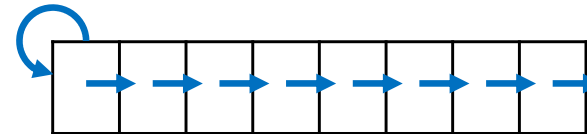
3ビット左に算術シフトすると

⇒ 01101000 


\*左シフトはオーバーフローが起こり得る

### 算術右シフト

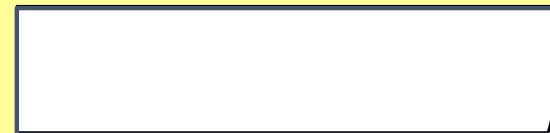
- ・語中のビットを右にずらす
- ・符号ビットは変えない



1ビット右に論理シフト

10011010 (-102)  値が1/2になった  
⇒ 01001101 (77)

1ビット右に算術シフトなら



- ・ 00010101 (+21)を1ビット右シフトすると?
- ・ 11101011 (-21)を1ビット右シフトすると?

\*正の数の右シフトは端数切捨てとなるが、  
負の数の右シフトは注意しよう。

# 自己確認問題

1100を右に1ビット算術シフト、論理シフトせよ

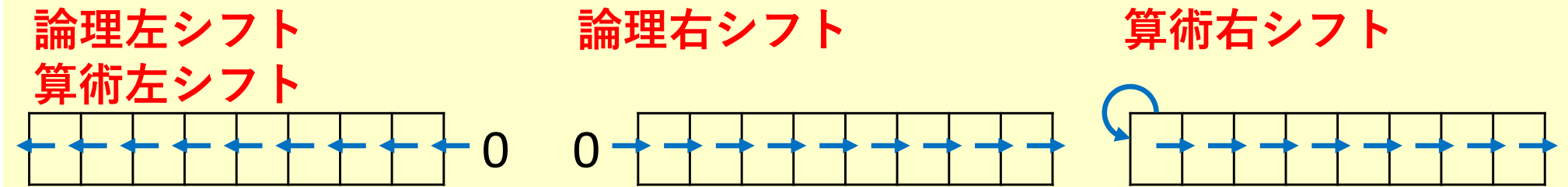
要は、一番右を1ビット捨てて、一番左に1ビット追加する

そして、その追加するビットは、

- 論理シフトは
- 算術シフトは

ということで、  
右に1ビット算術シフト  
右に1ビット論理シフト

# 自己確認○×クイズ



「符号ビットが0なら、算術シフトと論理シフトは同じ」は○か×か？

「1ビットの左算術シフトをした前後で正負が変わらない場合には、1ビット左算術シフト後の値は必ず元の値の2倍になる」は○か×か？

「kビットの左算術シフトをした前後で正負が変わらない場合には、kビット算術シフト後の値は必ず元の値の $2^k$ 倍になる」は○か×か？

## 演習問題 その⑦

11000110 を2ビット右算術シフト、2ビット左算術シフトせよ

元の値:  $11000110 = -58$

a. 2ビット右算術シフト

$1111000110 = -15$  ( $-58/4 = -14.5$  の切り下げ)

b. 2ビット左算術シフト

$1100011000 = +24$  ( $-58$  の4倍ではない)

そもそも、 $-58 \times 4 = -232$  は8ビットの2の補数で表現できる値  
( $-128 \sim +127$ ) の範囲外なので、正しく表せないのは当然  
(別途エラーとして処理すべきである)

# Lec. 3 の要チェック用語集

基数

MSB

LSB

2の補数

符号ビット

符号拡張

算術シフト

論理シフト

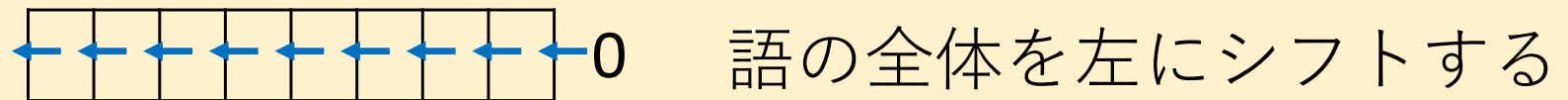
意味とか何の略かぐらいはチェックすべし

## 【補足】 算術左シフトの実際

- 算術左シフトには2種類の「流派」がある

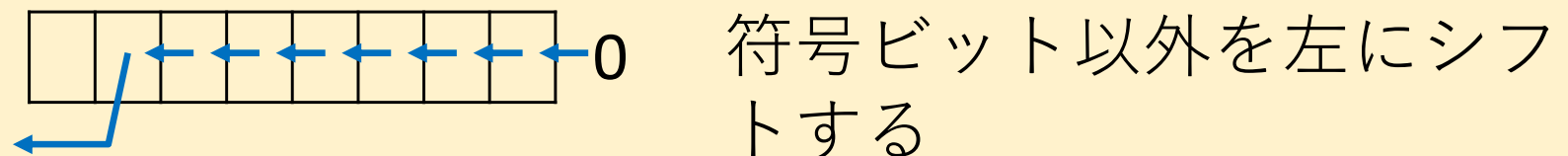
### 1. 「論理左シフトと同じ」派

- この授業のレジュメでは下のように説明した



### 2. 「符号ビットは変えない」派

- 下のように表される



- 実際のところどうなのか、調査した結果を示す

# 算術左シフトの実際

## (1) Intel 64 / IA-32

算術左シフト命令 (SAL) と論理左シフト命令 (SHL) は同じ処理を行う。

- Intel® 64 and IA-32 Architectures Software Developer's Manual より

The shift arithmetic left (SAL) and shift logical left (SHL) instructions perform the same operation; they shift the bits in the destination operand to the left (toward more significant bit locations). For each shift count, the most significant bit of the destination operand is shifted into the CF flag, and the least significant bit is cleared (see Figure 7-7 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*).

The shift arithmetic right (SAR) and shift logical right (SHR) instructions shift the bits of the destination operand to the right (toward less significant bit locations). For each shift count, the least significant bit of the destination operand is shifted into the CF flag, and the most significant bit is either set or cleared depending on the instruction type. The SHR instruction clears the most significant bit (see Figure 7-8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*); the SAR instruction sets or clears the most significant bit to correspond to the sign (most significant bit) of the original value in the destination operand. In effect, the SAR instruction fills the empty bit position's shifted value with the sign of the unshifted value (see Figure 7-9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*).



# Intel 64 / IA-32 続き

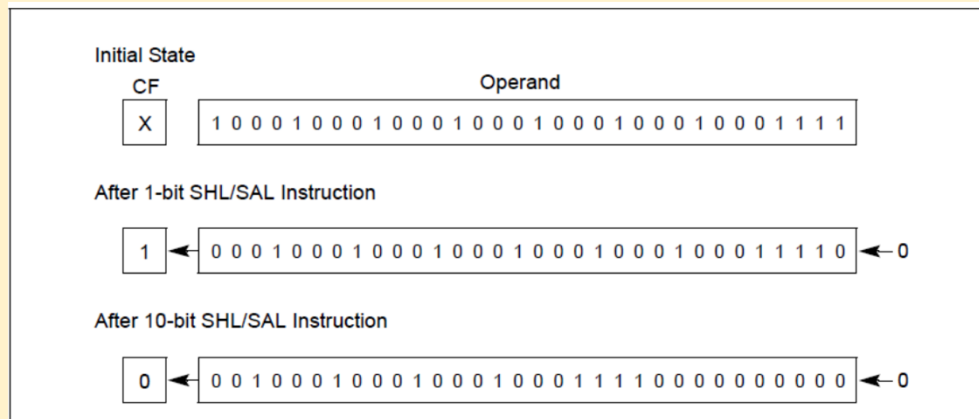


Figure 7-6. SHL/SAL Instruction Operation

左シフトは論理も算術も一緒

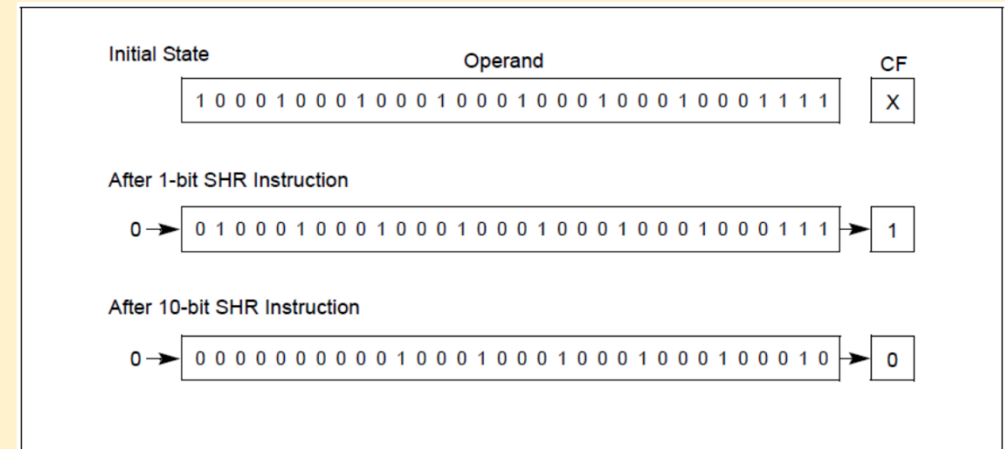


Figure 7-7. SHR Instruction Operation

論理右シフト

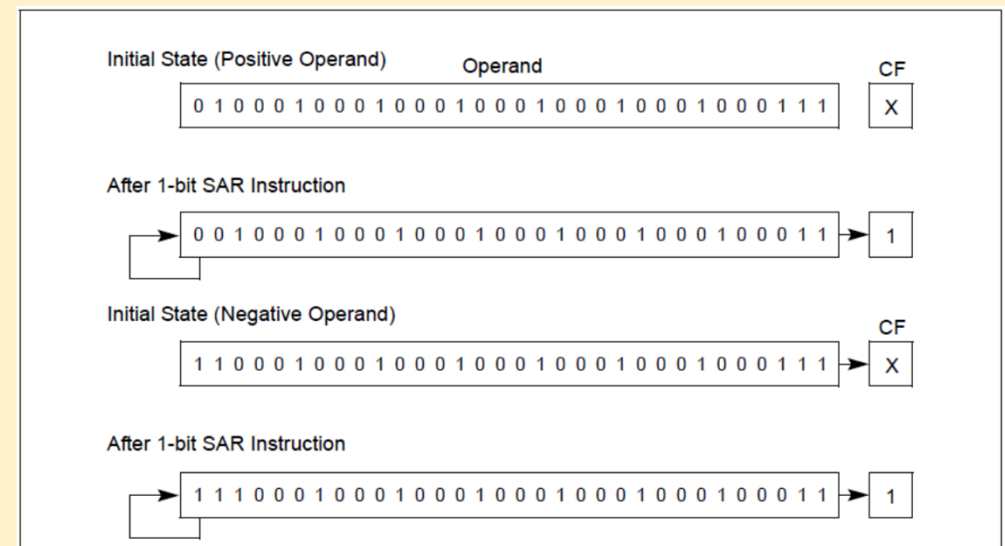


Figure 7-8. SAR Instruction Operation

算術右シフト

# 算術シフトの実際

## (2) ARMv7-M

- 「算術左シフト命令」は持っていない

### A4.4.2 Shift instructions

Table A4-3 lists the shift instructions in the Thumb instruction set.

Table A4-3 Shift instructions

Instruction	See	
Arithmetic Shift Right	<i>ASR (immediate)</i> on page A6-36	算術右シフト
Arithmetic Shift Right	<i>ASR (register)</i> on page A6-38	
Logical Shift Left	<i>LSL (immediate)</i> on page A6-134	論理左シフト
Logical Shift Left	<i>LSL (register)</i> on page A6-136	
Logical Shift Right	<i>LSR (immediate)</i> on page A6-138	論理右シフト
Logical Shift Right	<i>LSR (register)</i> on page A6-140	
Rotate Right	<i>ROR (immediate)</i> on page A6-194	
Rotate Right	<i>ROR (register)</i> on page A6-196	
Rotate Right with Extend	<i>RRX</i> on page A6-198	

# 算術シフトの実際

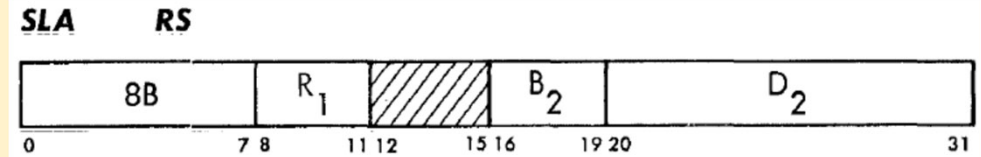
## (3) 昔のマイクロプロセッサ

- **MOTOROLA MC68000** （1980年代に出た16bitプロセッサ）
  - 算術左シフト（ASL）と論理左シフト（LSL）の動作は同じ（但し、**ASL**のみ、オーバーフロー判定フラグをセットする機能が付いている）
- **Zilog Z80 CPU** （1976年に発表された8bitプロセッサ）
  - 右シフト命令は**SRA**（算術）と**SRL**（論理）の2種類があるが、左シフト命令は**SLA**しか無い。**SLA**は、符号ビットを無視して左に1bitシフトする。

# 算術シフトの実際

## (4) 昔の「汎用機」

- IBM 360（1964年に発表された汎用機、つまり量産された最初の計算機）
  - SLA命令は符号ビットを変えずに、残りの31ビットを左にシフトする



The integer part of the first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of position 1, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is one.

# 算術シフトの実際

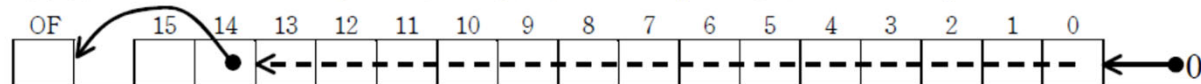
## (5) 架空のプロセッサ

- COMMET II (情報処理技術者試験で使用)
  - 算術左シフトでは符号ビットは変えずに残りのビットだけシフトする
  - IBM 360 を模したと思われる

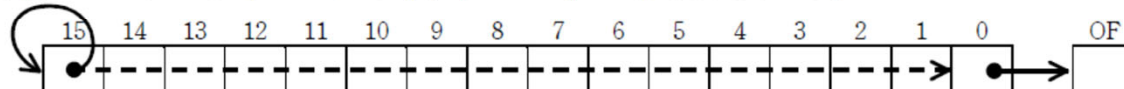
### 3. シフト演算命令におけるビットの動き

シフト演算命令において、例えば、1 ビットのシフトをしたときの動き及び OF の変化は、次のとおりである。

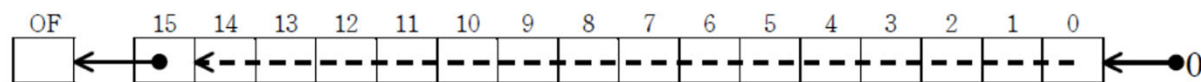
- (1) 算術左シフトでは、ビット番号 14 の値が設定される。



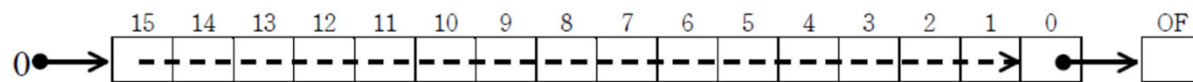
- (2) 算術右シフトでは、ビット番号 0 の値が設定される。



- (3) 論理左シフトでは、ビット番号 15 の値が設定される。



- (4) 論理右シフトでは、ビット番号 0 の値が設定される。



# 算術シフトの実際

## (6) プログラミング言語

- C言語

- 符号付き整数の左シフトの機能の詳細は、言語としては規定されていない（実装依存）
- Intel プロセッサ搭載PC上でgccでコンパイルすると、符号ビットは考慮せず、全ビットを左シフトするようである

```
#include <stdio.h>
void main( void ) {
    char a, b;      // 符号付き8bit整数変数を宣言
    a = -50;
    b = a << 1;      // -50 を 1bit 左シフト
    printf("%d\\n", b); // 結果を表示
    a = -100;
    b = a << 1;      // -100 を 1bit 左シフト
    printf("%d\\n", b); // 結果を表示
}
```

# C言語の実行結果の例

```
#include <stdio.h>
void main( void ) {
    char a, b;          // 符号付き 8bit 整数変数を宣言
    a = -50;
    b = a << 1;          // -50 を 1bit 左シフト
    printf( "%d\\n", b ); // 結果を表示
    a = -100;
    b = a << 1;          // -100 を 1bit 左シフト
    printf( "%d\\n", b ); // 結果を表示
}
```

```
$ gcc -o sla sla.c
```

```
$ ./sla
```

```
-100
```

```
56
```

```
$
```

上のようなC言語プログラムが書かれたファイル  
(ソースファイル) を実行形式に変換 (コンパイル)

コンパイルしたプログラムを実行

-50 ( $11001110_2$ ) を左に1bitシフトすると、  
-100 ( $10011100_2$ ) になった

-100 ( $10011100_2$ ) を左に1bitシフトすると、  
+56 ( $00111000_2$ ) になってしまった

# 算術シフトの実際 まとめ

- 算術左シフトには、確かに2つの「流派」があった
  1. 「論理左シフトと同じ」派
  2. 「符号ビットを変えない」派
- 調査した範囲では、以下の2つの例外を除き、「論理左シフトと同じ」派であった
  - a. IBM 360（1964年発表の汎用機）
  - b. COMMET II（情報処理技術者試験の為の架空プロセッサ）
- よって、現代のプロセッサの算術左シフトは「論理左シフトと同じ」派が主流である

日本のIT企業に就職するためには情報処理技術者試験に合格することは有益であるが、基本情報技術者試験ではCやJavaの問題を選択すれば、COMMET IIのアセンブラ（CASL II）の問題を解く必要は無い。



# 自己確認〇×クイズ（再）の答

「符号ビットが0なら、算術シフトと論理シフトは同じ」は〇か×か？

右シフトは、符号ビットが0なら算術も論理も同じ（〇）

左シフトは、

- 「論理シフトと同じ」派ならば、当然同じ（〇）
- 「符号ビットを変えない」派の場合、同じにならない（×）
  - 例えば、0110 を左論理シフトすると1100になるが、「符号ビットを変えない」派の左算術シフトの結果は0100になる

「1ビットの左算術シフトをした前後で正負が変わらない場合には、1ビット左算術シフト後の値は必ず元の値の2倍になる」は〇か×か？

- 1ビットの左論理シフトをした前後で正負が変わらないのは、元の値の最上位2ビットが「00」であった場合か、「11」であった場合である
- このような値に1ビットの左算術シフトを行ってもオーバーフローは発生せず、必ず元の値の2倍になる（〇）
  - 「論理シフトと同じ」派でも、「符号ビットを変えない」派でも結果は同じ

「kビットの左算術シフトをした前後で正負が変わらない場合には、kビット算術シフト後の値は必ず元の値の $2^k$ 倍になる」は〇か×か？

- 例えばk=2ビット左論理シフトをした前後で正負が変わらない値の例として0101を考えよう
  - 「論理シフトと同じ」派なら、0100となる（5が4になったので、×）
  - 「符号ビットを変えない」派も、0100となる（5が4になったので、×）

## 演習問題 その⑦（再）

11000110 を2ビット右算術シフト、2ビット左算術シフトせよ

元の値:  $11000110 = -58$

a. 2ビット右算術シフト

$1111000110 = -15$  ( $-58/4 = -14.5$  の切り下げ)



シフトした分2ビットを埋めるのは、元の符号ビットの値

b. 2ビット左算術シフト

- 「論理シフトと同じ」派の場合

$00011000 = +24$  ( $-58$ の4倍ではない)

- 「符号ビットを変えない」派の場合

$10011000 = -104$  ( $-58$ の4倍ではない)

そもそも、 $-58 \times 4 = -232$  は8ビットの2の補数で表現できる値  
( $-128 \sim +127$ ) の範囲外なので、正しく表せないのは当然  
(別途エラーとして処理すべきである)