

## 計算機での情報の表し方

計算機内では

全てのデータは **0,1** の組 (列) で表される

情報の最小単位:

0 か 1 か : **bit** (**binary digit**)

実際には幾つかの **bit** を組にして一度に扱う

(通常は 8 **bit** = 1 **byte** (**octet**))

1 **byte** で  $2^8 = 256$  通りの値を表せる

## 計算機での数の表し方

計算機内の **0,1** の列を、  
**二進表記** (二進法) で数値として扱う

正の整数の場合、1 byte で

$$0 \leq x \leq 2^8 - 1 = 255$$

の範囲の値が表せる

## 十進 $\longleftrightarrow$ 二進の変換

十進	二進
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
⋮	⋮
253	11111101
254	11111110
255	11111111

## 二進の九九 (一一?) の表

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

## 演習 (二進表記)

次の各式の被演算子 (operand) を  
二進表記で表した上で、  
それを用いて筆算で計算せよ。

(1)  $87 + 26$

(2)  $87 + 50$

(3)  $87 - 26$

(4)  $13 \times 11$

二進では桁数が多くて煩わしい

→ 十六進 (hexadecimal) 表記

十進	二進	十六進	十進	二進	十六進
0	<b>0000</b>	0	8	<b>1000</b>	8
1	<b>0001</b>	1	9	<b>1001</b>	9
2	<b>0010</b>	2	10	<b>1010</b>	a
3	<b>0011</b>	3	11	<b>1011</b>	b
4	<b>0100</b>	4	12	<b>1100</b>	c
5	<b>0101</b>	5	13	<b>1101</b>	d
6	<b>0110</b>	6	14	<b>1110</b>	e
7	<b>0111</b>	7	15	<b>1111</b>	f

十六進表記を示すのに、屡々0xを頭置する

十進	二進	十六進
0	<b>00000000</b>	0x00
1	<b>00000001</b>	0x01
2	<b>00000010</b>	0x02
3	<b>00000011</b>	0x03
4	<b>00000100</b>	0x04
5	<b>00000101</b>	0x05
⋮	⋮	⋮
253	<b>11111101</b>	0xfd
254	<b>11111110</b>	0xfe
255	<b>11111111</b>	0xff

## 計算機での数の表し方

計算機の内部では、

1 つの整数値を 8 bit で表している。

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array} = 87$$

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array} = 26$$

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array} = 50$$



## 負の数 (符号付き整数) の表現

符号 — も 0,1 で表す

単純に考えると、

- 符号に 1 bit (正なら 0・負なら 1)
- 絶対値に残り 7 bit ( $0 \leq |x| \leq 2^7 - 1$ )

→ 欠点あり

- 0 が 2 通りに表される (+0, -0)
- 演算で正負の場合分けが面倒

→ 他にもっと良い方法はないか?

“ 2 の補数表示 ”

## 負の数 (符号付き整数) の表現

符号 — も 0,1 で表す

単純に考えると、

- 符号に 1 bit (正なら 0・負なら 1)
- 絶対値に残り 7 bit ( $0 \leq |x| \leq 2^7 - 1$ )

→ 欠点あり

- 0 が 2 通りに表される (+0, -0)
- 演算で正負の場合分けが面倒

→ 他にもっと良い方法はないか?

“ 2 の補数表示 ”

## 負の数 (符号付き整数) の表現

符号 — も 0,1 で表す

単純に考えると、

- 符号に 1 bit (正なら 0・負なら 1)
- 絶対値に残り 7 bit ( $0 \leq |x| \leq 2^7 - 1$ )

→ 欠点あり

- 0 が 2 通りに表される (+0, -0)
- 演算で正負の場合分けが面倒

→ 他にもっと良い方法はないか?

“ 2 の補数表示 ”

## 負の数 (符号付き整数) の表現

符号 — も 0,1 で表す

単純に考えると、

- 符号に 1 bit (正なら 0・負なら 1)
- 絶対値に残り 7 bit ( $0 \leq |x| \leq 2^7 - 1$ )

→ 欠点あり

- 0 が 2 通りに表される (+0, -0)
- 演算で正負の場合分けが面倒

→ 他にもっと良い方法はないか?

“ 2 の補数表示 ”

## 負の数 (符号付き整数) の表現

符号 — も 0,1 で表す

単純に考えると、

- 符号に 1 bit (正なら 0・負なら 1)
- 絶対値に残り 7 bit ( $0 \leq |x| \leq 2^7 - 1$ )

→ 欠点あり

- 0 が 2 通りに表される (+0, -0)
- 演算で正負の場合分けが面倒

→ 他にもっと良い方法はないか?

“ 2 の補数表示 ”

## 2 の補数表示

$$-128 = -2^7 \leq x \leq 2^7 - 1 = 127$$

の範囲の整数値が表せる

$$x' := \begin{cases} x & (0 \leq x \leq 2^7 - 1) \\ x + 2^8 & (-2^7 \leq x \leq -1) \end{cases}$$

とおくと、

$$0 \leq x' \leq 2^8 - 1$$

→  $x'$  を符号無し整数として表す

$$x \equiv x' \pmod{2^8}$$

## 2 の補数表示

$$-128 = -2^7 \leq x \leq 2^7 - 1 = 127$$

の範囲の整数値が表せる

$$x' := \begin{cases} x & (0 \leq x \leq 2^7 - 1) \\ x + 2^8 & (-2^7 \leq x \leq -1) \end{cases}$$

とおくと、

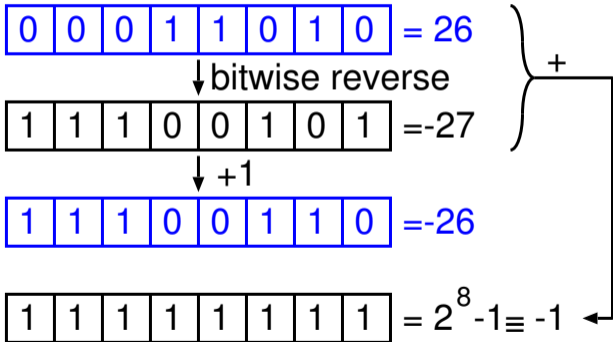
$$0 \leq x' \leq 2^8 - 1$$

→  $x'$  を符号無し整数として表す

$$x \equiv x' \pmod{2^8}$$

## 2 の補数表示

$x$  ( $0 \leq x \leq 2^7 - 1$ ) から  $-x$  を求めるには ...





## 符号付き整数の加算の桁溢れ判定

先程の演習の計算結果を、

2 の補数表示として読んでみよう

$-2^7 \leq x \leq 2^7 - 1$  の範囲の整数値が表せる

- $87 + 26$ 
  - 結果が上の範囲に収まる
  - 正しく読み取れる
- $87 + 50$ 
  - 結果が上の範囲に収まらない。
  - 正しく読み取れない・・・「桁溢れ」

## 符号付き整数の加算の桁溢れ判定

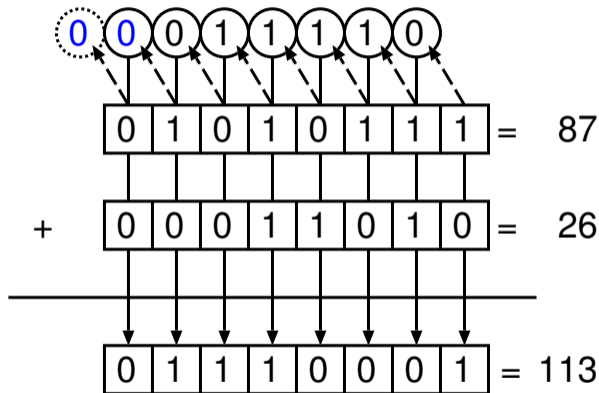
先程の演習の計算結果を、

2 の補数表示として読んでみよう

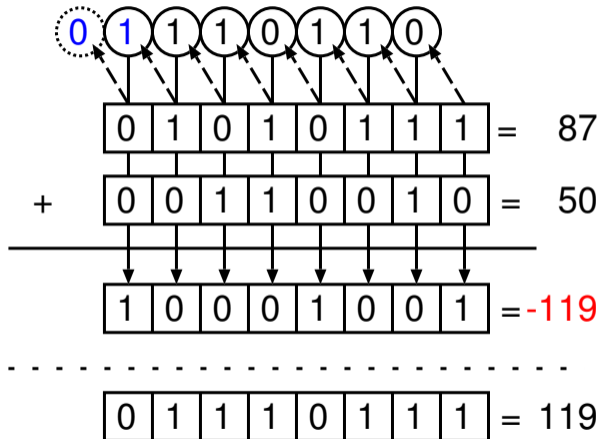
$-2^7 \leq x \leq 2^7 - 1$  の範囲の整数値が表せる

- $87 + 26$ 
  - 結果が上の範囲に収まる
  - 正しく読み取れる
- $87 + 50$ 
  - 結果が上の範囲に収まらない。
  - 正しく読み取れない・・・「**桁溢れ**」

## 符号付き整数の加算の桁溢れ判定



## 符号付き整数の加算の桁溢れ判定



## 符号付き整数の減算

$$87 - 26$$

→ 減算を別の演算として扱うのは面倒

→  $87 + (-26)$  として計算してみよう

- (1)  $-26$  を 2 の補数表示で表す
- (2)  $87 + (-26)$  を二進表記の筆算で (普通に) 足す
- (3) 9 bit 目への繰上りは捨て、下 8 bit を取る



## 演習問題

(1)  $\pm 87, \pm 50, \pm 26$  を、  
8 bit の “2 の補数表示” で表せ。

(2) 上の表示を用いて、次を筆算で計算せよ。

- 9bit 目への繰上がりが発生するものは？
- 桁溢れが発生しているものは？

(1)  $(+87) + (+26)$       (2)  $(+87) + (-26)$

(3)  $(-87) + (+26)$       (4)  $(-87) + (-26)$

(5)  $(+87) + (+50)$       (6)  $(+87) + (-50)$

(7)  $(-87) + (+50)$       (8)  $(-87) + (-50)$

(3) 桁溢れの発生を判定するには？

## 桁ずらし

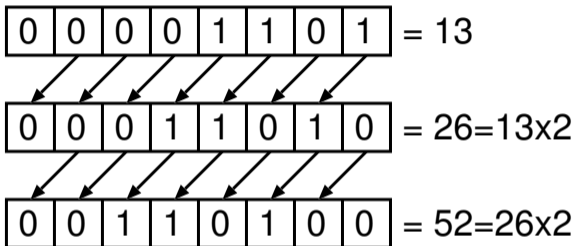
$$13 \times 11$$

乗算は桁をずらして加える

「桁ずらし (**bit shift**)」が基本的な操作

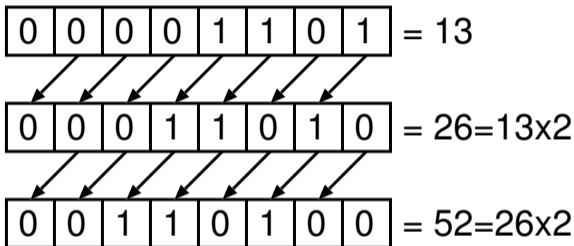


左シフト:  $x \mapsto 2x$



- 桁溢れの判定は？
- 負の数でも大丈夫か？

左シフト:  $x \mapsto 2x$



- 桁溢れの判定は？
- 負の数でも大丈夫か？

右シフト:  $x \mapsto \left\lfloor \frac{x}{2} \right\rfloor$

0 0 0 0 1 1 0 1 = 13

0 0 0 0 0 1 1 0 = 6 = [13/2]

0 0 0 0 0 0 1 1 = 3 = [6/2]

- 桁溢れは起きない。
- 負の数でも大丈夫にするには？  
(符号 bit の扱い)

以上のような、

- データ (bit 情報) の保持
- 基本的な演算

を計算機で実現するには？

→ 「論理回路」

以上のような、

- データ (bit 情報) の保持
- 基本的な演算

を計算機で実現するには？

→ 「論理回路」

## 論理回路

- **組合せ回路** :  
入力 (の組) によって出力が決まる  
→ 演算に用いる
- **順序回路** :  
内部状態を保持し、  
入力と入力前の状態とによって  
出力と出力後の状態とが決まる  
→ データ (bit 情報) の保持に用いる

## 論理回路

- **組合せ回路** :  
入力 (の組) によって出力が決まる  
→ **演算**に用いる
- **順序回路** :  
内部状態を保持し、  
入力と入力前の状態とによって  
出力と出力後の状態とが決まる  
→ **データ (bit 情報) の保持**に用いる