第11講　符号付き演算

符号（プラス、マイナス）を0,1で表す

可変長、固定長

符号付き数値データの表現

可変長、固定長

コンピュータで扱う**数値データ**

↓

ビット数がバラバラだと

何かと不都合。

可変長、固定長

一般的にはビット数を**「固定」**

固定長データで数値を扱う

固定長データの長さ

8bit = 1byte を基準

16bit, 32bit, 64bit, 128bit 等

詳細設計・開発時に選択

固定長データの長さが複数ある理由

bit数が少ないほど

① 処理速度が向上

② メモリが効率的

bit数が多いほど

表示範囲が広がる

固定長データの弱点

表示できるデータに

制約ができる（最大値、最小値）

↓

計算によっては**「エラー」**が起こる

固定長データの表示範囲を超えたら

オーバーフロー（けたあふれ）

エラーなので、そのまま処理を続けることはできない

負数の表現

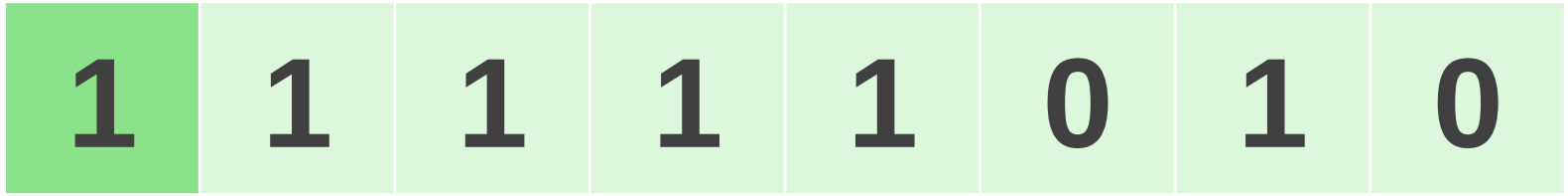
① 数値の**符号（±）**を1, 0で表示

② 固定長データ

③ **2の補数**で表示

負数を扱うデータ＝**「符号付き」**

符号ビット



先頭のビットを「符号」とする

先頭のビット＝**MSB**

符号ビットが１のとき

1	1	1	1	1	0	1	0

負数を**2の補数**で表示

↓

そのまま**10進数に変換できない**

符号付きデータの基数変換

1	1	1	1	1	0	1	0

① 符号を＋に変更する

② 10進数に変換

③ －を付けて表示

1と0を入れ替え（ビット反転）、1加算する

1	1	1	1	1	0	1	0

0	0	0	0	0	1	0	1

						+	1

0	0	0	0	0	1	1	0

手順②

$(6)_{10}$

手順③

1	1	1	1	1	0	1	0

$(-6)_{10}$

練習問題

次の**符号付き8bit**の**数値**データを
10進数に変換しなさい

① | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

② | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

特殊なケース　最小値

1	0	0	0	0	0	0	0

0	1	1	1	1	1	1	1

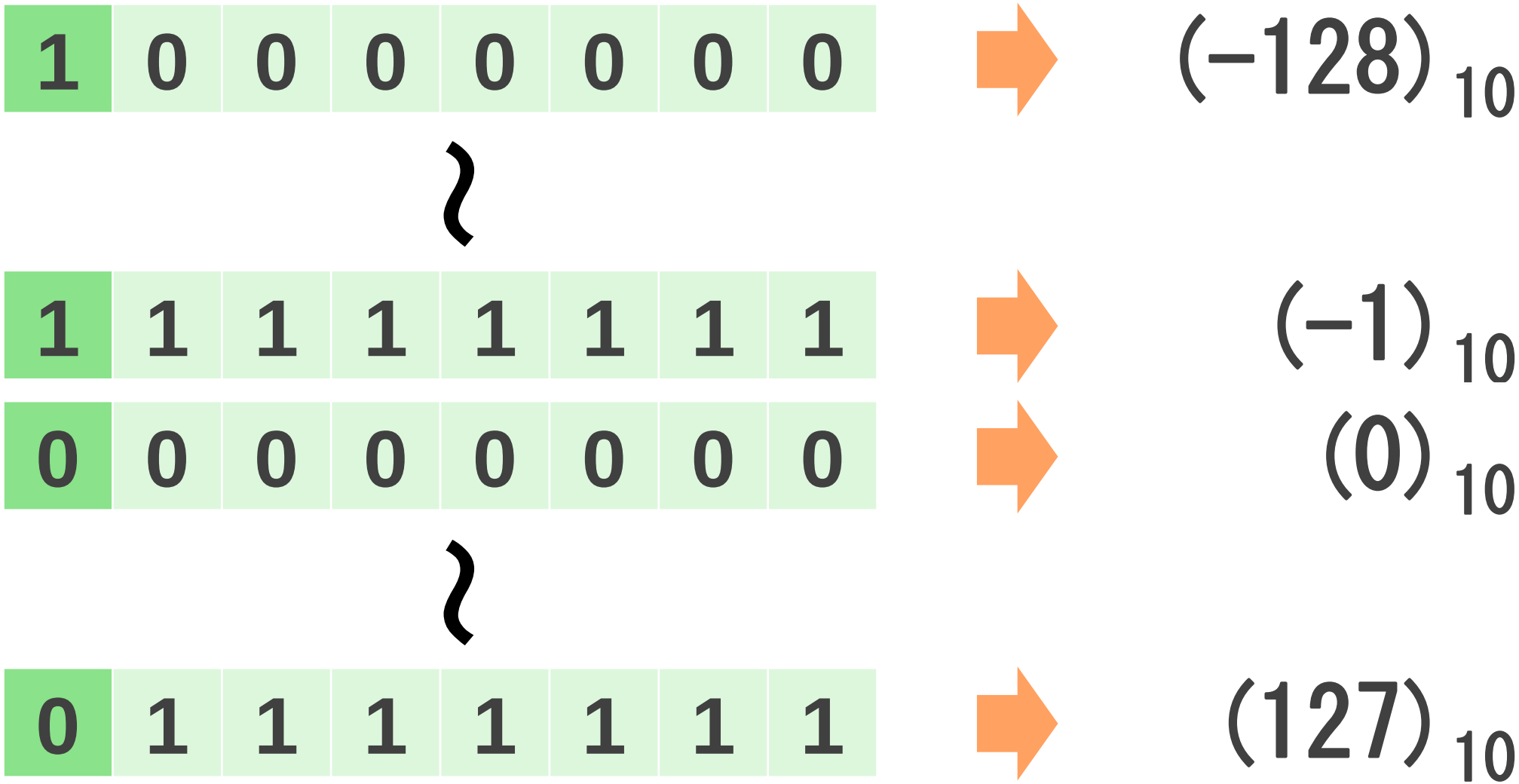
						+	1

1	0	0	0	0	0	0	0

符号ビットが1なので
ビット反転して
1加えると
元に戻ってしまう

$(-128)_{10}$

符号付き8bitの表示範囲



固定長データの表示範囲

符号を考えない8bit固定長

最小値: 00000000　（10進数でも 0）

最大値: 11111111　（10進数で 255）

固定長データの表示範囲

符号を考えない n bit固定長

最小値:　0

最大値:　2^n-1

固定長データの表示範囲

符号付き8bit固定長

最小値: 10000000　（10進数で -128）

最大値: 01111111　（10進数で 127）

固定長データの表示範囲

符号付き n bit固定長

最小値: 2^{n-1}

最大値: $2^{n-1} - 1$

10進数を符号付き2進数に変換

1.　0以上の数値

　　→　そのまま2進数に変換

2. 負の数値

　　①　符号なしで2進数変換

　　②　2の補数表示にする

例題

$(-15)_{10}$ を 符号付き8ビットに変換

負数なので、**そのまま変換できない**

① 符号を外してから2進数に変換

② 2の補数表示にする

手順①

① 符号を外してから2進数に変換

$(15)_{10}$ =

0	0	0	0	1	1	1	1

② 2の補数表示にする

0	0	0	0	1	1	1	1

$(15)_{10}$

1	1	1	1	0	0	0	0
						+	1
1	1	1	1	0	0	0	1

$(-15)_{10}$

2の補数表示のメリット

減算が加算だけで行える

$(20)_{10} - (15)_{10}$ ➡ $(20)_{10} + (-15)_{10}$

減算を加算で処理できる

	0	0	0	1	0	1	0	0
+	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	0	1

$(20)_{10}$

$(-15)_{10}$

$(5)_{10}$

オーバーフロー

演算結果が

表示範囲を超えたら**エラー！**

「オーバーフロー」（けたあふれ）

オーバーフローの条件

加算結果、符号が変わってしまった！

↓

表示範囲を超えた

「オーバーフロー」（けたあふれ）

オーバーフローの例

	0	1	0	0	0	0	0	1
+	0	1	0	0	0	0	0	1
	1	0	0	0	0	0	1	0

$(65)_{10}$

$(65)_{10}$

プラスの数値どうしを加えたのに

符号がマイナスになっている！

オーバーフローはエラー！

オーバーフローが発生したら

↓

そのまま計算できないので

ビット数を増やす（ビットの拡張）

ビットの拡張

	0	1	0	0	0	0	0	1
+	0	1	0	0	0	0	0	1

符号付き8bit　→　符号付き16bit

	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
+	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

$(65)_{10}$

$(65)_{10}$

$(130)_{10}$

負数のビット拡張

	1	0	0	0	0	0	1	1
+	1	0	0	0	0	1	0	0

$(-125)_{10}$

$(-124)_{10}$

符号付き8bit → 符号付き16bit

	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1
+	1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0
	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1

$(-125)_{10}$

$(-124)_{10}$

$(-249)_{10}$

ビットの拡張　まとめ

拡張して増えたビットには

符号bitをコピー

符号bitは不変

第11講　符号付き演算

符号（プラス、マイナス）を0,1で表す

可変長、固定長

符号付き数値データの表現

第11講はここまで