演算機能回路

第3週

2006/10/16

泉知論

立命館大学 理工学部 電子情報デザイン学科

加減算器の応用回路

comparator (CMP)

incrementor (INC), decrementor (DEC)

counter, accumulator

比較器

- CMP (comparator の略) などと表記する。
- 本質的には加減算器に等しい。
- zero で一致判定
- sgn で大小判定

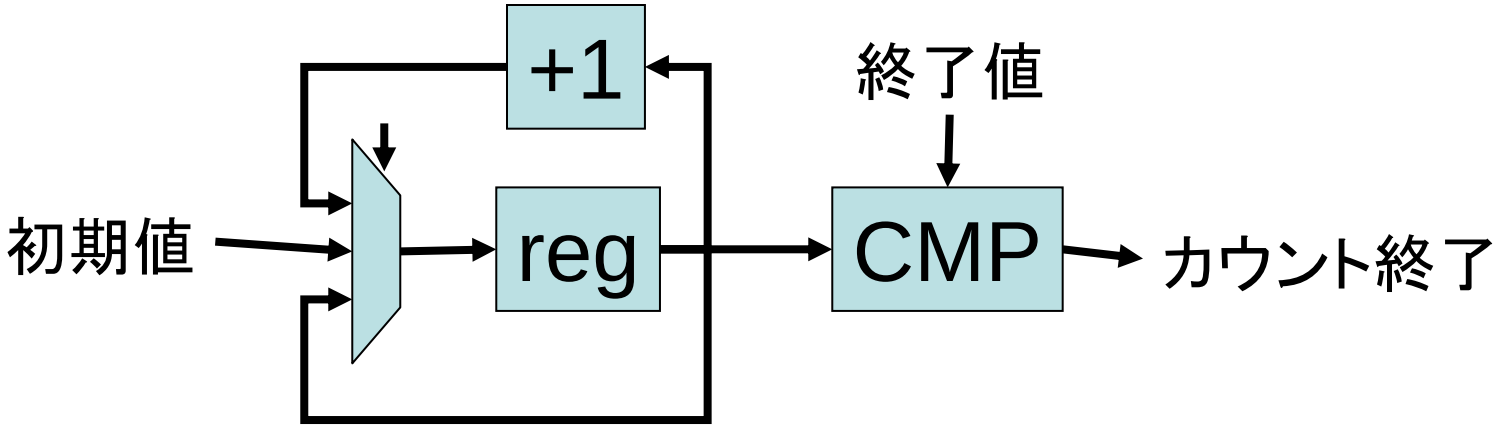
インクリメンタ、デクリメンタ

- +1(-1) 専用の加（減）算器
- カウントアップ（ダウン）などに用いる
- INC (incrementor の略) や DEC (decrementor の略) などと表記する。
- 回路構成によっては通常の加減算器よりも小さくできる。

カウンタ

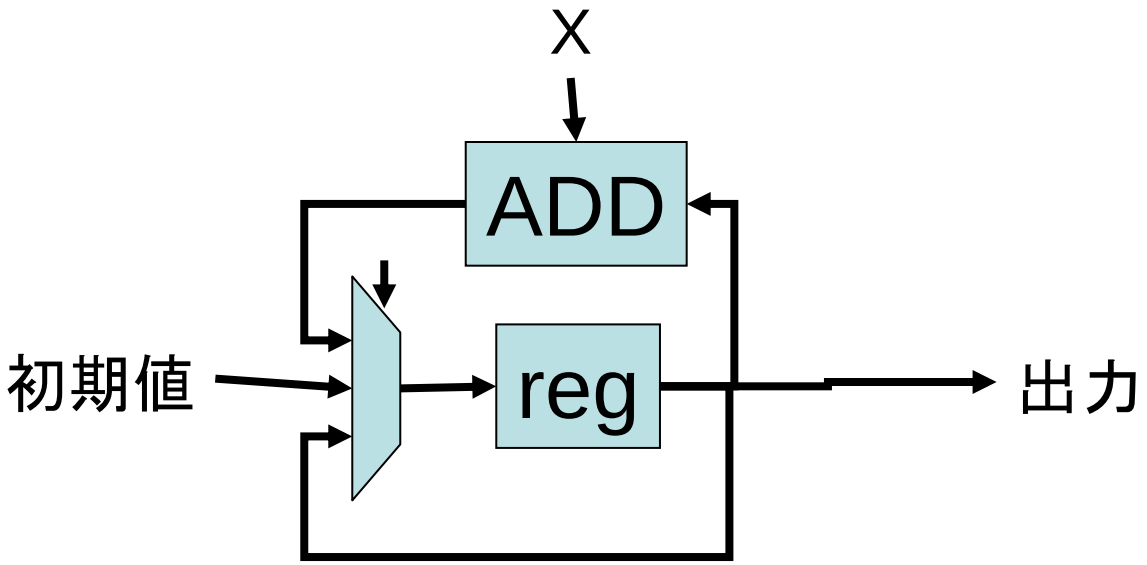
- カウント信号が入るとカウントアップ（ダウン）
- 特定の値になったら終了信号を出力
- レジスタ、セレクタ、インクリメンタ、比較器を組み合わせて実現できる

※セレクタのかわりにフリップフロップの制御信号として設計する場合もある。



積算器(積分器)

- 積算(累積加算、積分、ΣX_i)の計算を行う
- 合計、平均の計算など
- レジスタと加算器で実現可能
- accumulator などと呼ばれる



固定小数点表現
ビット幅変換

fixed point

cast

固定小数点 v.s. 浮動小数点

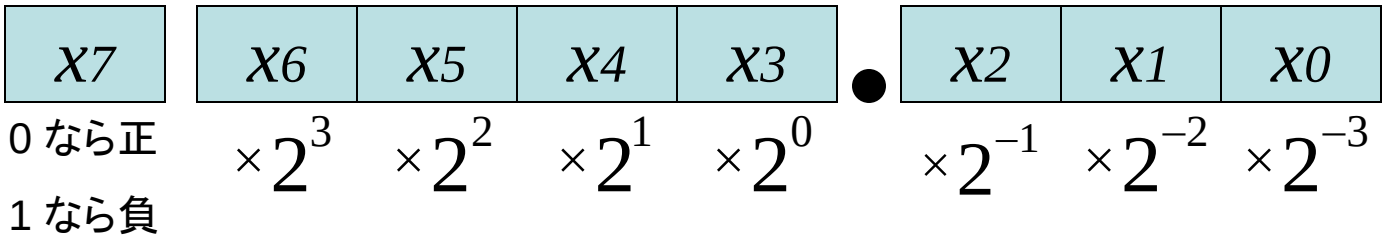
- 精度
- 回路コスト

- 緻密な解析、必要最小限の精度とコスト

固定小数点

- fixed point とも呼ばれる。
- n-bit ワードの第i番目のビットの値を xi とし、小数部をk ビットとすると

$$\text{値（符号無）} = \sum_{i=0}^{n-2} x_i \times 2^{i-k} \qquad \text{値（符号付）} = -x_{n-1} \times 2^{n-k-1} + \sum_{i=0}^{n-2} x_i \times 2^{i-k}$$



固定小数点の演算

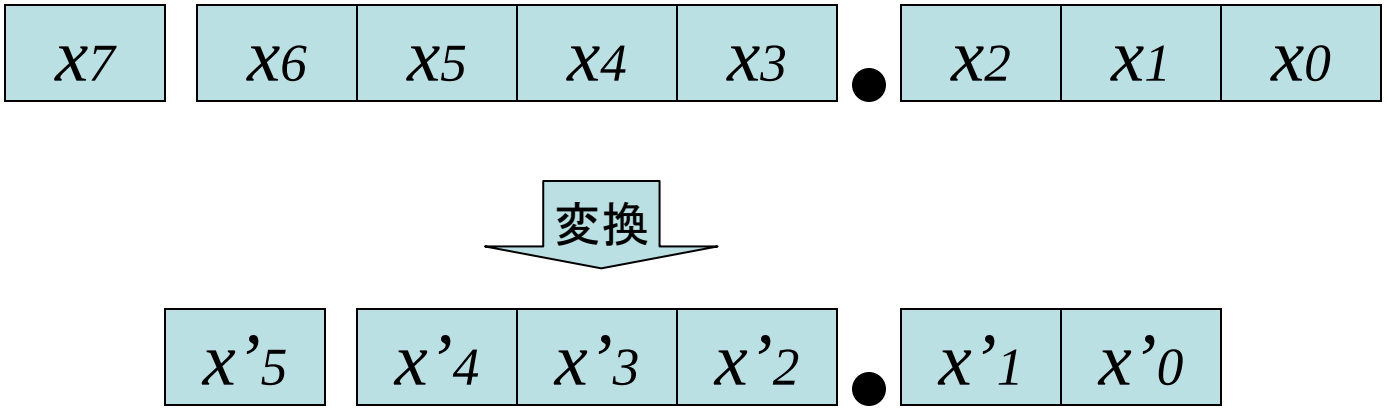
- 整数演算器をそのまま利用可能
- 値を $\frac{1}{2^k}$ 倍（kビット右シフト）して解釈すればよい
  ※左にシフトすることもありうる。
- 小数点の位置に注意すること
  ※普通の10進数の筆算と同じ

演習3A

- 次の2の補数符号付固定小数点表現（符号1ビット、整数部4ビット、小数部3ビット）の表す値を10進数で答えよ。
  1. 00000000
  2. 00001000
  3. 10000000
  4. 11111111
- ±1000の範囲の値を誤差0.01以内で表現したい。どのように値を表現すればよいか。

ビット幅変換

- ビット幅変換(cast)の必要性
  - 演算によるビット幅増加
  - ビット幅の異なる数どうしの演算
  - 回路規模削減などのためのビット数削減
  - 回路規模と精度のバランスが重要



上位ビット拡張(符号拡張)

- 上位にkビット拡張

  →扱える値の範囲を拡大

  – 符号無

    0 を k 個追加

  – 符号付

    符号ビット(もとの最上位ビット)の値をk個追加

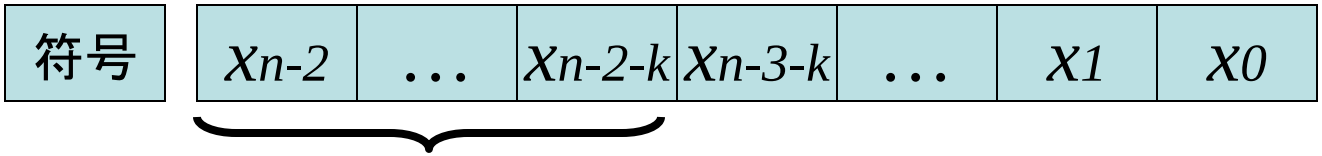
    ！！そうしないと符号がかわってしまう！！

下位ビット拡張

- 下位にkビット拡張
- 小数部の拡張→精度向上
- 整数部の拡張→ 2^k 倍
- 0をk個追加

上位ビット削減

- 上位ｋビットの削減
  →扱える値の範囲が縮小（精度悪化）
- 目的のビットで収まりきらない場合、対処が必要
  - 最大値を超えてしまう　→最大値
  - 最小値をしたまわってしまう→最小値
    （最大値、最小値＝符号ビット＋符号ビット反転）
- 飽和処理（saturation, SAT）、クリッピングなどと呼ばれる



このｋビットを削除→全て符号ビットに一致していれば溢れなし

下位ビット削減

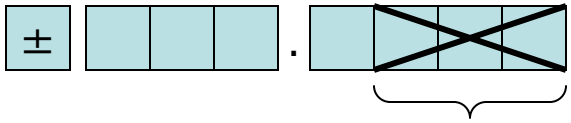
- 下位kビットの削減→精度悪化
- 削減方法
  - 現在の値以下の最大の値（切捨て）
    - 単純にｋビット捨てる
  - 現在の値以上の最小の値（切上げ）
  - 絶対値が現在の値のもの以下の最大の値（絶対値の切捨て）
  - 絶対値が現在の値のもの以上の最小の値（絶対値の切上げ）
  - 現在の値に最も近い値
    - 丸め(round)、0捨1入（10進数の四捨五入の要領）
    - 算術型丸めと銀行型丸め

いちばん簡単＝小さく速い

いちばん精度が良い

切捨て、切上げ

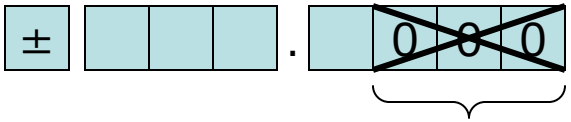
- 切捨て



下位 k ビットを単純に捨てる

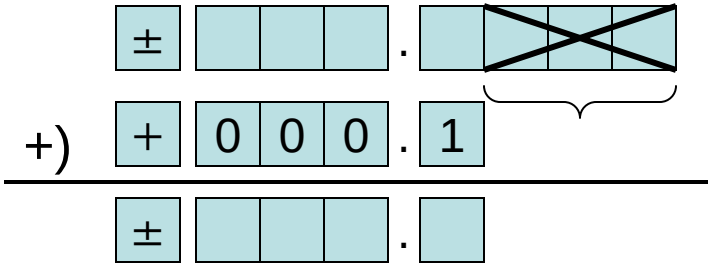
- 切上げ

※下位kビットが全て 0 のとき



下位 k ビットを単純に捨てる

※下位kビットに1を含むとき

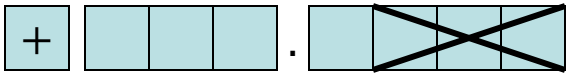


下位 k ビットを捨てて 0…01 を足す

絶対値の切捨て、切上げ

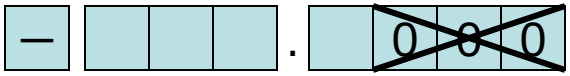
- 絶対値の切捨て

※値が正のとき



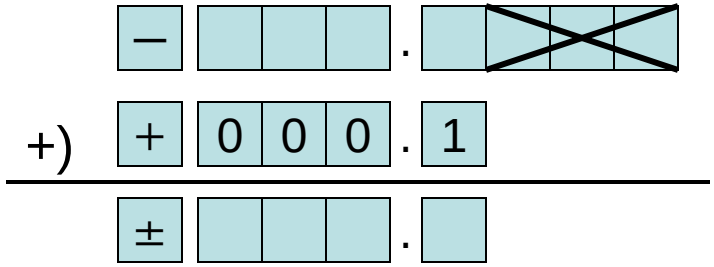
下位 k ビットを捨てる

※値が負で下位 k ビットが全て 0 のとき



下位 k ビットを捨てる

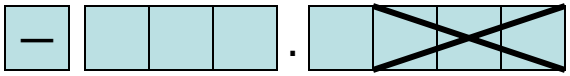
※値が負で下位 k ビットに1を含むとき



下位 k ビットを捨てて 0…01 を足す

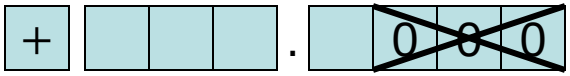
- 絶対値の切上げ

※値が負のとき



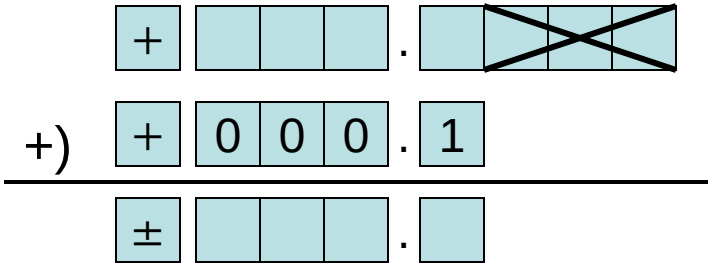
下位 k ビットを捨てる

※値が正で下位 k ビットが全て 0 のとき



下位 k ビットを捨てる

※値が正で下位kビットに1を含むとき

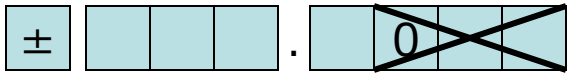


下位 k ビットを捨てて 0…01 を足す

丸め

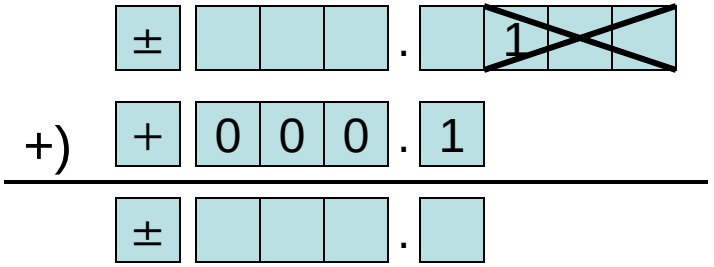
- 現在の値に最も近い値にする
- 0捨1入（算術丸め）

※下位ｋビット目が 0 のとき



下位ｋビットを捨てる

※下位ｋビット目が1のとき



下位ｋビットを捨てて 0…01 を足す

※ちょうど中間の値の扱いについてバリエーション有り

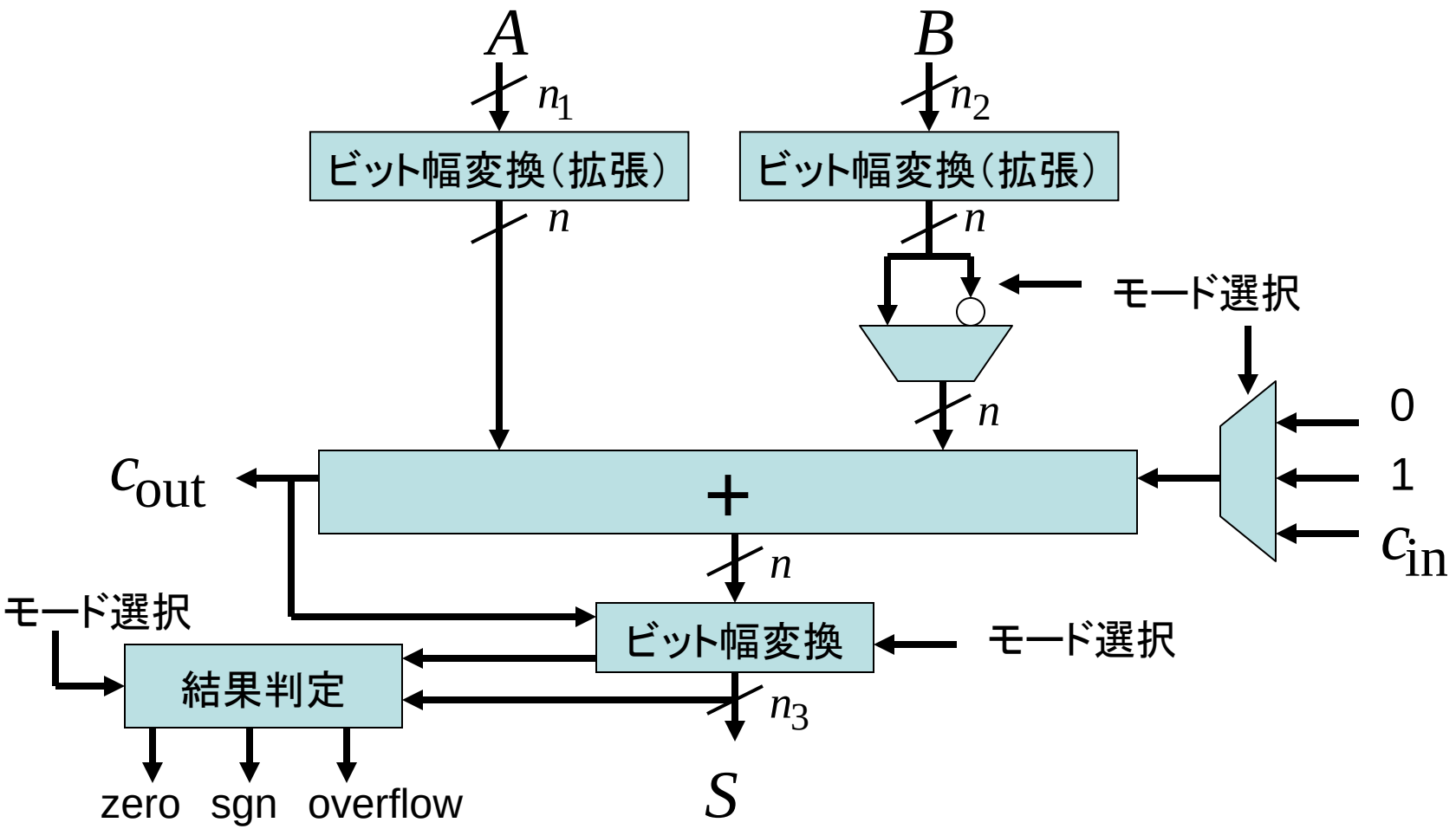
演習3B

- 符号1ビット、整数部2ビット、小数部1ビットの入力値を符号1ビット、整数部4ビット小数部3ビットに拡張し出力する論理回路を設計せよ。
- 符号1ビット、整数部5ビットの入力値を符号1ビット、整数部3ビットに削減し出力する論理回路を設計せよ。飽和処理を行い、overflow 信号を加えて溢れた場合には1、そうでなければ0を出力せよ。

演習3B（続き）

- 符号1ビット、整数部1ビット、小数部3ビットの入力値を符号1ビット、整数部1ビットに削減し出力する論理回路を設計せよ。値の切り上げ／切捨て、絶対値の切り上げ／切捨て、丸め、の全ての場合について設計せよ。なお、加減算器やインクリメンタを部品として使用してよい。（加減算の内容を AND/OR/NOT で書く必要はない。）

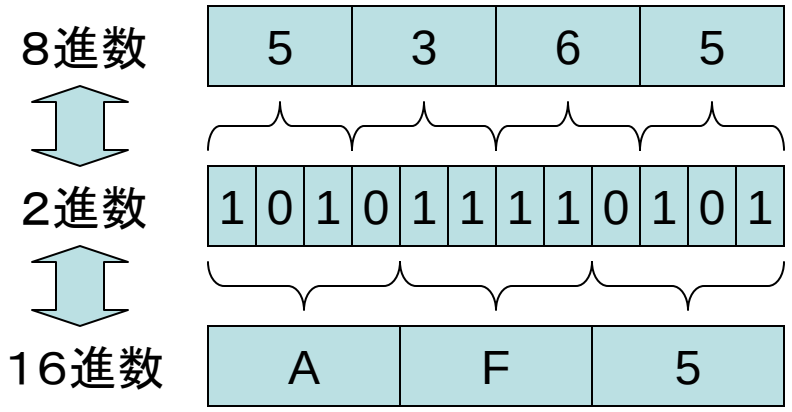
加減算器(複合)


A
B
n_1
n_2
ビット幅変換(拡張)
ビット幅変換(拡張)
n
n
モード選択
n
0
1
c_{in}
c_{out}
+
n
モード選択
ビット幅変換
モード選択
結果判定
n_3
zero
sgn
overflow
S

8進数、16進数

- 2進数は桁数が多く、読んだり、書いたり、覚えたりするのが面倒
- 数ビットずつまとめて扱う

例



2進数	8進数
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

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