～浮動小数点輪講～

よしみ まさと *

概要

浮動小数点の概念について触れ，丸め，例外処理を含めた浮動小数点演算の手法の実際について解説します．一般的な四則演算に加え，開平，指数，対数計算への応用について実装の面から議論します．

1 浮動小数点の基本

1.1 浮動小数点の形式

浮動小数点演算は IEEE 標準規格 (IEEE Std 754-1985) に基づいて定められている．一般的に浮動小数点形式は，3 つの部分に分割された 2 種類の形式がサポートされている．

単精度浮動小数点　32 ビット：符号ビット，指数部 (8 ビット)，仮数部 (23 ビット)
倍精度浮動小数点　64 ビット：符号ビット，指数部 (11 ビット)，仮数部 (52 ビット)

浮動小数点表現は，2 進数の全ての数について，

$$x = \pm 1.m_1, m_2 \cdots \times 2^e \tag{1}$$

のような形式で表現したものである．

符号ビット：　s
指数部：　$e = E +$ バイアス値(単精度 → 0x7f，倍精度 → 0x3ff)
仮数部：　$f = .m_1 m_2 \cdots$ (小数点以下第 1 位以下の値)

仮数部が小数点以下のみを使用しているのは，整数部第一桁が 1 であると暗黙のうちに取り決めており，自明な 1 を省いて表現しているためである (正規化数：ノーマル数)．符号ビットで値の正負を表現し，指数部と仮数部で値の絶対値を表現する．指数部は単精度で -126～127 の値を取りうるため，適当な数のバイアスを加算し，指数部が正負の値を取る煩雑さを回避する．

整数部が 0 であるデータ $\pm 0.m_1 m_2 \cdots \times 2^E$ を非正規化数 (デノーマル数) と呼ぶ．

IEEE754 では他に 3 種類の特殊なデータ型を定義している．

1. **非数 (NaN)**
   指数部が最大値で，仮数部が 0 以外のデータ．仮数部の最上位ビットが 1 のときを Signaling

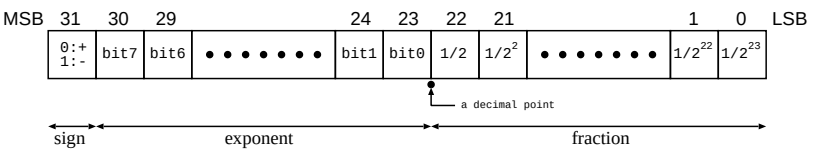

図 1　単精度浮動小数点データ形式

* 443@am.ics.keio.ac.jp

NaN(SNaN)，0 のときを Quiet NaN(QNaN) と呼ぶ．浮動小数点で無効な演算の結果として用意された記号

2. **無限大**
   指数部が最大値で，仮数部が 0 のデータ．符号により正と負の無限大が存在する．数学的な無限大と同義
3. **ゼロ**
   指数部と仮数部が 0 のデータ．正と負のゼロがある (比較演算的には同値)

表 1 に浮動小数点表現の一例を示す．

$+1.0$	`0x3f800000`	単精度
	`0x3ff0000000000000`	倍精度
-1.0	`0xbf800000`	単精度
	`0xbff0000000000000`	倍精度
`QNaN`	`0x7fb00000`	単精度
	`0x7ff7ffffffffffff`	倍精度
`SNaN`	`0x7fffffff`	単精度
	`0x7fffffffffffffff`	倍精度
$+\infty$	`0x7f800000`	単精度
	`0x7ff0000000000000`	倍精度
$-\infty$	`0xff800000`	単精度
	`0xfff0000000000000`	倍精度
$+0$	`0x00000000`	単精度
	`0x0000000000000000`	倍精度
-0	`0x80000000`	単精度
	`0x8000000000000000`	倍精度

表 1　浮動小数点表現の一例

1.2　浮動小数点の丸めモード

丸めとは，数値をあるデータ範囲内に入るように仮数部の最下位ビットに対して切り上げか切り捨てを行って近似値を求めることである．

- 浮動小数点形式は，10 進数の値を正確に表現することができない
  - 0.5_{10} は浮動小数点形式では `0x3f000000`
  - 0.1_{10} は浮動小数点形式では $1.\dot{1}00\dot{1} \times 2^{-4}$ という循環無限小数 (有限ビットでは正確な表現不可)

IEEE754 では 4 種類の丸めモードを定義している (表 2)．

- IEEE754 では丸めの具体的な実装方法は規定していない
- 一般的には，FP データに 3 ビット丸め用補助ビットを追加して実装する (図 2)
  - Guard，Round ビットは演算精度を高めるために追加される
    - 近い値同士の減算において，精度の下落を防ぐ
  - Sticky ビットは，シフトアウトされるあふれビット全ての論理和

記号	RM	意味
RN	00	表現可能な最も近い値に丸める 2 つある場合は最下位ビットが 0 の値
RZ	01	0 の方向に丸める (切り捨て)
RP	10	$+\infty$ の方向に丸める
RM	11	$-\infty$ の方向に丸める

表 2　IEEE754 の丸めモード

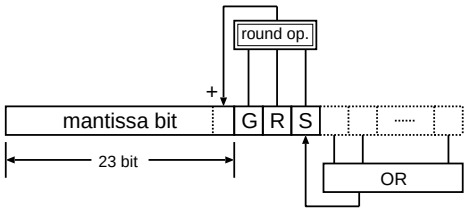

図 2　丸め用追加ビット

- 3 つの追加ビットの初期値は全て 0
- 計算の最後に，3 つの追加ビットと丸めモードによって適切な値が選択され，仮数部最下位ビット (LSB) に加算される (表 3)
- 実装によっては，Guard ビットと Sticky ビットで丸めが行われる場合もある

丸めモード	符号	仮数部 LSB	Guard	Round	Sticky	切上 or 切捨
RN = G &(LSB | R | S)	×	×	0	×	×	↓
	×	×	1	1	×	↑
	×	×	1	0	1	↑
	×	1	1	0	0	↑
	×	0	1	0	0	↓
RZ = 0	×	×	×	×	×	↓
RP =(~ 符号)&(G| R |S)	1	×	×	×	×	↓
	0	×	1	×	×	↑
	0	×	0	1	×	↑
	0	×	0	0	1	↑
	0	×	0	0	0	↓
RM =符号& (G|R|S)	0	×	×	×	×	↓
	1	×	1	×	×	↑
	1	×	0	1	×	↑
	1	×	0	0	1	↑
	1	×	0	0	0	↓

表 3　丸めの実現

2　浮動小数点の演算

2.1　共通処理

- 浮動小数点演算全体を通して，共通する処理手順は以下の通り

1. **オペランド処理**
   オペランドを符号・指数部・仮数部に分割する．同時に，特殊データ型の判定を行う．無限大または NaN ならば専用の処理を，非正規化数の場合は処理前に正規化を行う
2. **演算処理**
   (追加 3 ビットを含めた) 演算を実行する．例外が検出された場合，例外処理が実行される．結果を正規化する
3. **丸め処理**
   得られた結果を丸めモードに従って丸める
4. **結果の分類**
   指数部 (バイアス無し) の値が 0 以下の場合はアンダーフロー処理，最大値以上の場合はオーバーフロー処理を実行する
   – 結果 (演算・丸め) が 0 になる場合は 0 処理が行われる
5. **結果の格納**
   符号，指数，仮数の結果が出力される
6. **トラップ発生判定**
   なんらかのトラップが発生すると判断された場合，所定の例外ハンドラに制御が移る

- IEEE754 では，5 つの例外が定義され，これらが検出されるべきであるとしている
  1. 無効浮動小数点演算例外 (Invalid Floating Operation Exception)
     無限大や NaN との演算，0/0 などの演算などで発生
  2. 浮動小数点ゼロ除算例外 (Floating Zero Divide Exception)
     0 で除算が行われる場合に発生
  3. 浮動小数点オーバーフロー例外 (Floating Overflow Exception)
  4. 浮動小数点アンダーフロー例外 (Floating Underflow Exception)
  5. 浮動小数点精度落ち例外 (Floating Precision Exception)
     追加の 3 ビットのいずれかが 1 であったときに発生

2.2 浮動小数点演算器

- **共通処理**
  – 浮動小数点演算の流れを図 3 に示す
- **加算**
  1. 両オペランドの指数部の値を大きい方にそろえる
  2. 小さな方が変化した指数部の値だけ，仮数部の値を右シフトする
  3. 仮数部同士を加算する
  4. 仮数部の加算結果の影響を指数部に反映させる
  5. 結果を丸める
- **減算**
  減算は，第 2 オペランドの符号を反転しての加算処理で実現できる
- **乗算**
  1. オペランドが $\mathrm{a.bbbb} \times 10^{\mathrm{c}}$ と $\mathrm{d.eeee} \times 10^{\mathrm{f}}$ とすると，乗算結果は $(\mathrm{a.bbbb} \times \mathrm{d.eeee}) \times 10^{(\mathrm{c+f})}$
  2. 指数部同士を加算，仮数部同士を乗算する
  3. 仮数部を正規化し，指数部を変更する
  4. 結果を丸める

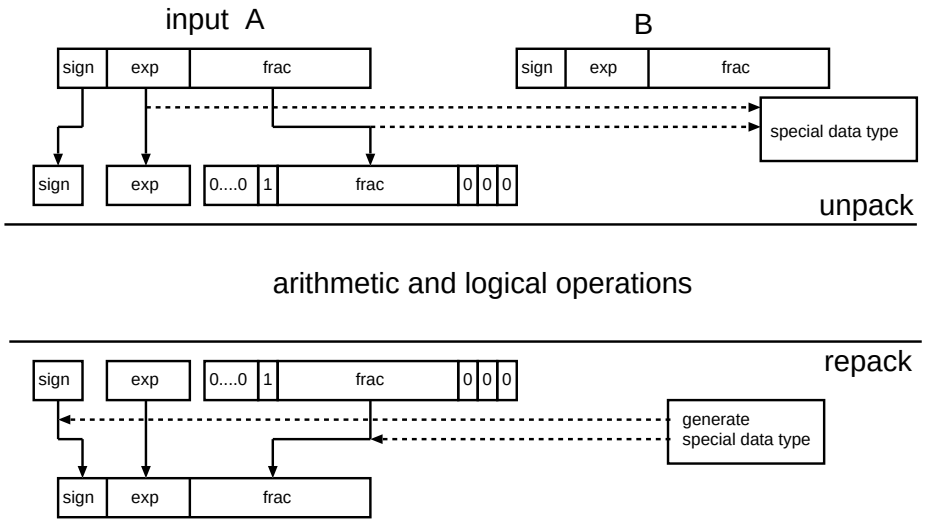

図 3　FP 演算の流れ

- **除算**
  1. オペランドが $\mathrm{a.bbbb} \times 10^{\mathrm{c}}$ と $\mathrm{d.eeee} \times 10^{\mathrm{f}}$ とすると，除算結果は $(\mathrm{a.bbbb/d.eeee}) \times 10^{(\mathrm{c}-\mathrm{f})}$
  2. 指数部同士を減算，仮数部同士を除算する
  3. 仮数部を正規化し，指数部を変更する
  4. 結果を丸める

3　初等超越関数

sin や cos などの三角関数，および指数 e^x，対数 $\log x$ などの初等超越関数について，演算器の実装について述べる．

初等超越関数回路の実現方法として，以下の手法がある．

- **漸近近似** (ニュートン・ラプソン法など)
  所望の精度が得られるまでに繰り返される演算数が不定
- **級数展開** (テイラー展開など)
  低次項では低精度・高次項の実現は困難
- **テーブル引き**
  精度検証必須・メモリ消費

いずれも回路実装に有効とは言えない方法と言える．そのため，所望の精度を得られる程度のテーブルを用意し，そこからインデックスとして値を取り出し，補正を加えることで精度を上げる方法を考える．

3.1　対数関数器

3.1.1　回路実現の理論

$y = \log x$ から，入力 x から出力 y を求める演算器を実現したい．対数関数の特徴から，以下のような実装で log 回路を実装する．

- 入力 x を浮動小数点 $x = m \times 2^E$ $(1 \leq m < 2)$ 形式に変更すると，

$$y = \log_a(x) = \frac{\log_2(m \times 2^E)}{\log_2(a)} = \frac{\log_2(m) + \log_2(2^E)}{\log_2(a)} = \frac{\log_2(m) + E}{\log_2(a)} \tag{2}$$

  - $1/\log_2(a)$ は対数の底 a によって決定される定数
  - E は浮動小数点の指数部に相当する符号付き整数
- $[0:1)$ の範囲で $\log_2(m)$ の値を計算できれば，加算と乗算により任意の底で入力値の対数値が得られる

対数は sin のような周期性が無いため，入力を浮動小数点実数と見なして計算すべき log の定義域を制限 (必要な値は $[1:2)$) することで実現できる．

3.1.2 実装〜線形補間

実装の前準備として 2 つのテーブルを作成する．

- 必要な定義域 $[0:1)$ について，適当な区間に等分する
- 各区間の始点 x_i における値 $y_i = \log_2(x_i)$ を求める
- (x_{i+1}, y_{i+1}) と (x_i, y_i) の 2 点を直線で結び，その傾きと切片を求める
- 各区間の傾きと切片の値をテーブルに格納する (Coefficient_table:CTable, Intercept_table:ITable)

対数演算回路は以下の処理を行うもとのして実装する．例として，単精度浮動小数点実数を対象に，$[1:2)$ 範囲を 32 分割して対数を求める回路について述べる．

- 入力 x は正規化された浮動小数点実数 x とする
- x の仮数部の最上位 5bit をアドレスとして CTable，ITable から値 C，I を取り出す
- x を $[1:2)$ の値 x_m に変更 (仮数部の上位に 0x0_0111_1111 を付加する)
- x_m と C を乗算し，I を加算する (双方浮動小数点演算) $\rightarrow x_t$
- x の指数部 E のバイアスを除いた値を浮動小数点実数 E_m に変更
- x_t と E_m を加算し，底の対数の逆数 $1/\log_2(a)$ を乗算し，結果の値 y を出力する

図 4 に処理の流れを示す．

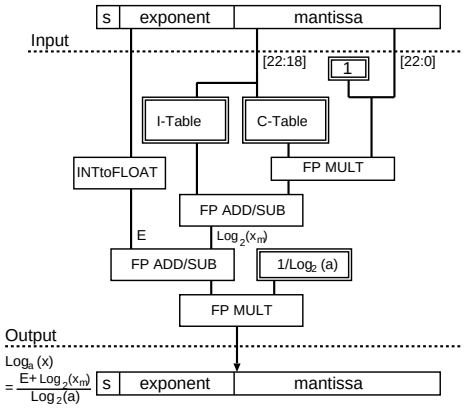

図 4　対数計算の流れ

3.1.3 線形補間の検証〜2 次補間

線形補間を行った場合の精度について検証する．線形補間では本来の対数に対して，値が小さくなる．誤差値 (本来の対数値 − 線形補間値) のグラフを図 5 に示す．10 進数において 4 桁 (12 ビット) 程度の精度が確保できる．

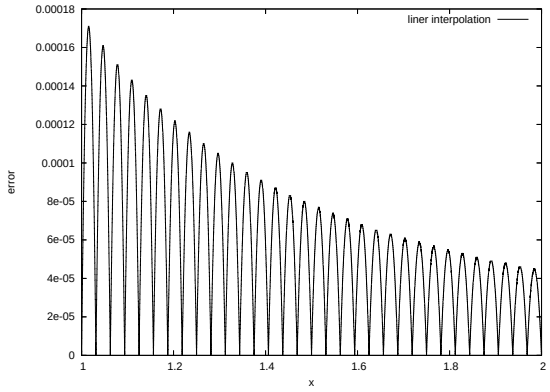

図 5　線形補間の誤差 (32 分割)

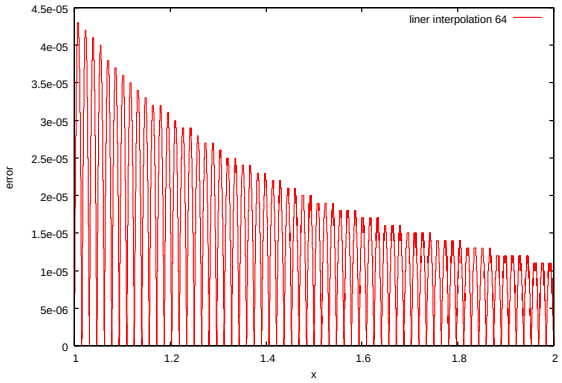

図 6　線形補間の誤差 (64 分割)

64 分割した場合は (図 6)，14 ビット程度の精度が期待できる．
さらなる精度を求める場合，誤差が最小値になるようにさらに補正することが考えられる．

- 線形補間において，誤差がもっとも大きくなる場所は，$1 \leq x < 2$ を分割した最初の区間である (7)
- 区間ごとに $(0, 1)$ の範囲の 2 次関数 $u = 4t(t-1)$ を作り，線形補正した値に加算する (図 8)
- 2 次補正により，21 ビット程度の精度が実現できる

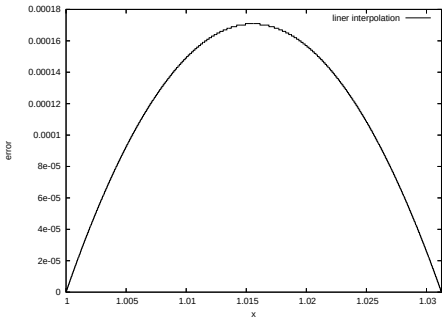

図 7　誤差が最大になる区間 (32 分割)

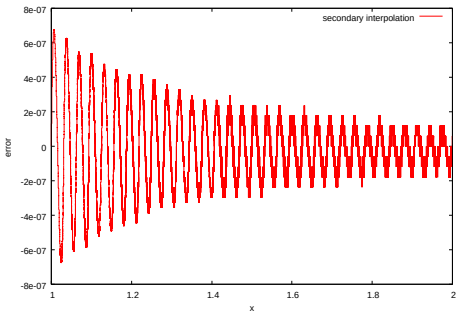

図 8　2 次補間の誤差 (32 分割)

2 次補正の実装は図 9 のような処理で実現できる．

- 分割した区間数分のエントリを持つ新しいテーブルを用意する
- 事前に計算されたその区間の誤差の最大値 (ゲイン) を格納しておく
- $u = 4t(t-1) = 4t^2 - 4t$ の計算を行う回路を作成する (t は浮動小数点なので指数部に 2 を加算する処理で代用)
- t は x_m から区切り開始値 (x の最上位ビットから第 5 仮数ビットまで等しくそれ以降が 0 のビット列) を減算することで求める
- 線形補正された値に得られた 2 次補正値 u を加算する

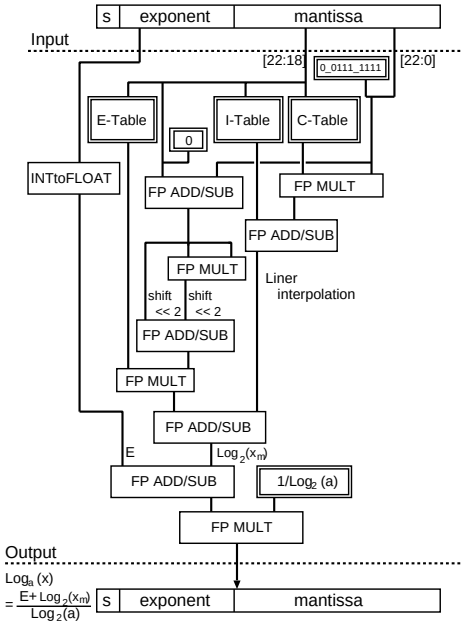

図 9　対数計算の流れ (2 次補正)

3.2　指数関数器

指数関数も，対数と同様の方法で計算することが可能である．

- $R = E \cdot \ln(2) + X$ と表せる実数 R を入力値として仮定する
- $\exp(R) = \exp(E \cdot \ln(2) + X) = 2^E \cdot \exp(X)$
- $\exp(R)$ は，R を $\ln(2)$ で割り，商 E (整数) と剰余 X $(0 \leq X < \ln(2))$ を求める
- $\ln(2)$ は定数なので，$\ln(X)$ $(0 \leq X < \ln(2))$ を対数と同様の近似方法で求める
- $1 \leq \ln(x) < 2$ なので，求めた $\ln(x)$ を E だけシフトする (指数に設定する) ことにより指数関数器が実現できる

3.3　正弦・余弦関数器

正弦・余弦関数器の実装にあたり，以下の特徴を考慮する．基本的には対数回路と同様の方法で実現できる．

- sin，cos のような関数は周期性がある
- 正弦関数は最初の 1/4 周期を実現できれば残りの出力値の符号反転と折り返しで実現できる (図 10)
- 8 ビット程度の精度の場合は全てをテーブルに格納すると効果的
- 15〜16 ビット以上の場合はインデックステーブルを作り，補正を掛ける
- 出力値のほとんどは $(0, 1)$ の範囲に収まる

例として，入力が m $(0 < m < 2\pi)$ であるときの $\sin(m)$ を実現する回路について述べる (直線補間)．

- m は整数部分が 0, 小数部分が 16 ビットの固定小数点実数であるとする $(m = (0.x_0x_1x_2 \cdots x_{15})_2)$
- 入力値 x の最上位 2 ビットで符号反転，折り返しの有無を決定する (どの象限かでテーブルを引く

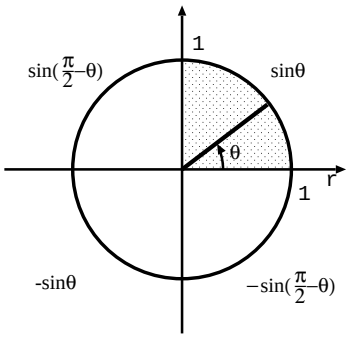

図 10　正弦関数の第一象限から全体への拡張

アドレスを変えたり，符号を変えたりする)

- θ $(0 < m < \frac{\pi}{2})$ を複数区間 (分割数は 2^d) に分割 (x_i) し，$\sin(x_{i+1})$ と $\sin(x_i)$ を結ぶ直線の傾きと，$\sin(x_i)$ の値をテーブルに格納する
- 入力値 x の上位から 3 ビット目から d ビット取り出し，テーブルを引く
- テーブルから得られた傾きと切片から $\sin(x)$ を求める

処理の流れを図 11 に示す.

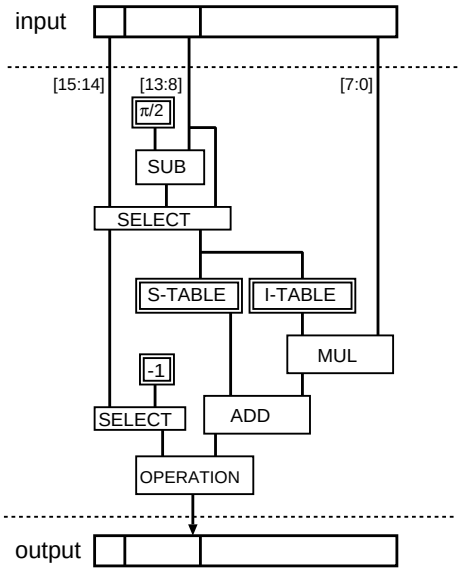

図 11　sin 回路の処理の流れ

SELECT モジュールでは反転操作や，符号反転などが行われる．2 次補間についても対数関数と同様，区分ごとに偶数次成分が大きい左右均等な誤差が発生するため，2 次関数の補正を加えることで誤差を小さくすることができる．直線補正で 15 ビット，2 次補正で 21 ビット程度の精度を得ることが可能である．同様の方法で余弦回路 (cos 回路)，tan 回路を実現することが可能である．

4 Reference

1.「マイクロプロセッサ・アーキテクチャ入門」中森 章 著 インターフェース増刊 TECH I CQ 出版社
2. Design Wave Magazine 2003 年 7 月号 特集 1「高速回路設計の極意」 CQ 出版社
3. “What Every Computer Scientist Should know About Floating-Point Arithmetic” ACM Computing Surveys,Vol 22,No 1,pp.5-48
4.「浮動小数点演算について計算機科学者は何を知っておくべきか」bit 別冊コンピュータ・サイエンス 1993 年 9 月号別冊 pp.5-42
5. C MAGAZINE 2003 年 5 月号 特集 3「数値計算の基礎」 SOFTBANK Publishing
6. 高木ら “冗長 2 進加算木を用いた VLSI 向き高速乗算器” 電子通信学会論文誌 1983.6 Vol.J66-D No.6 pp.683-690
7. 高木ら “冗長 2 進加算木を用いた VLSI 向き高速乗算器” 電子通信学会論文誌 1984.4 Vol.J67-D No.4 pp.450-457
8. 高木ら “冗長 2 進表現を利用した指数・対数関数計算用ハードウェアアルゴリズム” 電子通信学会論文誌 1986.1 Vol.J69-D No.1 pp.11-20