IEEE Standard for Binary Floating-Point Arithmetic

(IEEE, August 1985)

川口 貴裕

平成１０年１１月２８日 (土)

1 概要

1.1 目的

この規格の目的は,100% ソフト,100% ハード, 又はソフトとハードの組み合わせによって実現される, 規格を満足する浮動小数点システムの実装にある. この規格によりプログラマやシステムのユーザは, 利用環境が規格を満たしているかどうかという点を判断することができる.

1.2 規定事項

この規格が規定しているのは

1. 浮動小数点の基本フォーマット, 拡張フォーマット
2. 四則演算・平方根・剰余・比較命令
3. 整数・浮動小数点間のフォーマット変換
4. 異なる浮動小数点フォーマット間の型変換
5. 浮動小数点と 10 進文字列の変換
6. 非数 (NaN) を含めた浮動小数点例外

である.

1.3 規定外事項

以下のことについて, この規格は規定をしていない.

1. 10 進文字列と整数のフォーマット
2. NaN の符号と有効数字 (significand) に関する解釈
3. 拡張フォーマットについての 2 進 $\leftrightarrow$10 進変換

2 定義

この規格が定義している言葉は以下の通りである.

- biased exponent [げたばき指数]

  指数フィールドが非負になるように選択された定数 (bias と呼ぶ) と, 指数の値との和.

- binary floating-point number [2 進浮動小数点数]

  符号, 符号付き指数, 有効数字の 3 つの部分からなるビット列. もしそのビット列が数値として存在するならば, その値は 2 の指数乗と有効数字の, 符号つきの積で表す. この規格の中においては, ビット列と, そのビット列が表現している数というものを, 常に区別しているわけではない.

- denormalized number [デノーマル (不正規化数)]

  指数が予約された値 (通常は最も小さな値にする) の非ゼロの浮動小数点数のことで, 明示的でも暗示的でも最上位の有効数字のビットが 0 になる. 拡張フォーマットは, 有効数字のけち表現を採用しておらず, 明示的に MSB を 0 にすることで, 正規化されていない値を表現する.

- destination [デスティネーション]

  命令の結果のために用意されたロケーションのこと. デスティネーションは, ユーザが明示的に指定しても, システムによって暗示的に用意されていても良い.

- exponent [指数]

  2 進浮動小数点の構成要素の 1 つで, 表現している数値を決定する際に, 2 の肩に乗る整数. 通常この指数は, 符号付き指数またはバイアスされていない (unbiased) 指数と呼ばれる.

- fraction 有効数字フィールド

  2 進表現の有効数字の内, 小数部分を示すもの.

- mode [モード]

  ユーザがセット, 読み込み, 書き込みが行える値で, 次に続く数値演算命令に実行制御を渡すことのできるもの. デフォルトのモードとは, 明示的に反対を意図したステートメントを, プログラムまたはその仕様書に盛り込むことをしない限り, 反映されると (ユーザが) 仮定してもよいモードのことである. 丸め誤差の方向を制御する丸めモードは, 必ず実装しなければならない (shall). そして実装によっては, 結果の精度を落とすための丸めの精度が要求される. また, オプションとして, 例外を扱うためのトラップイネーブル/ディゼーブルのモードを実装してもよい.

- NaN [Not a Number : 非数]

  浮動小数点フォーマットに基づいてエンコードされるシンボリックな実体である. NaN には,Signaling NaN と Quiet NaN という 2 種類の NaN が存在する.Signaling NaN(s)(複数形で表記, 即ち NaN の表現は一意に決められていない) は, それがオペランドとして現れた時に, 無効演算例外のシグナルを発生する.Quiet NaN(s) は, シグナルを発生しないほとんど全ての数値演算命令で伝播していく.

- result [結果]

  デスティネーションによってもたらされる結果 (ビット列) で, そのビット列は通常, 数値を表す.

- significand [有効数字]

  2 進の浮動小数点の構成要素の 1 つで, 暗示的または明示的な小数点よりも左側にある先頭のビット (leading bit) と, 小数点より右側にあるビット列 (fraction) からなる.

- shall

  この規格の準拠に際して必ず実装しなければならないものについて, そのとき shall という言葉を使用している.

- should

  義務ではないが, 実装を強く推奨している時に, この単語が用いられている.

- status flag [ステータスフラグ]

  セットとクリアができる,2 つの状態をとる値のこと. ユーザはフラグをクリアしたり, コピーしたり, 前の状態に復帰させせることができる. それをセットしたとき, ステータスフラグはシステムに依存した情報を含み, あるユーザに対してアクセスができないようにすることができる. 規格の命令は, 以下のフラグのセットを行うことがある.

  - 不正確
  - アンダーフロー
  - オーバーフロー
  - 0 除算
  - 無効演算

- user [ユーザ]

  しかしながらこの規格では, 規格で定められたアクセスを行ったり, プロフラミング環境の命令を制御する人やハードウェアやプログラムを特定していない.

3 フォーマット

規格では,2 つのグループに分けることのできる全 4 種類の浮動小数点フォーマットを定義している. グループには基本フォーマットと拡張フォーマットがあり, 各々について単精度と倍精度を持っている. 規格の実装レベルは, フォーマットのサポートの組み合わせによって判断することができる.

3.1 値の集合

この節ではフォーマットの範囲内で表現される数値についてのみ規定し, どのようにビット列のエンコードを行うかについては規定しない. そのような数値について, 具体的にどのようなフォーマットを選択するかに関しては,

- 有効数字の桁数である仮数のビット数 p
- 指数の最大値 E_{max}
- 指数の最小値 E_{min}

の 3 つの整数で規定することができる. これらの値を含めたパラメータの一覧表を表 1に示す.

表 1: 規定されている各フォーマットのパラメータ

パラメータ	基本単精度	拡張単精度	倍精度	拡張倍精度
有効数字の桁数 p	24	32 以上	53	64 以上
指数の最大値 E_{max}	$+127$	$+1023$ 以上	$+1023$	$+16383$ 以上
指数の最小値 E_{min}	-126	-1022 以下	-1022	-16382 以下
指数のバイアス	$+127$	規定しない	$+1023$	規定しない
指数のビット長	8	11 以上	11	15 以上
全体のビット長	32	43 以上	64	79 以上

そして各々の実体は, 以下のような形で提供しなければならない (shall).

$$(-1)^s 2^E (b_0.b_1b_2\cdots b_{p-1})$$

但し

- s は 0 または 1
- E は E_{min} から E_{max} の間の任意の整数
- b_i は 0 または 1

である. また, 各フォーマットは

- $+\infty$ と $-\infty$ の 2 つの無限大
- 最低 1 種類の Signaling NaN
- 最低 1 種類の Quiet NaN

を持たなければならない.

以上のようなフォーマットの規定では,1 つの数値を表現するのには冗長である. その例として, $2^0(1.0) = 2^1(0.1) = 2^2(0.01)$ が挙げられる. しかし, このような冗長性は拡張フォーマットにおいてのみ起こり得る. そして $\pm 2^{E_{min}}(0.b_1b_2\cdots b_{p-1})$ で表わされる非ゼロの値を不正規化数と呼ぶ. 特定の指数の値は非数 (NaN), 無限大 (∞), ゼロと不正規化数のために予約されている. なお, ゼロには全てのフォーマットについて符号の情報が付加されており,0 除算のようなケースにおいてその効力を発揮する.

3.2 基本フォーマット

基本単精度と基本倍精度で表現される数は,3 つのフィールドから構成されている.

1. 1 ビットの符号 s
2. げたばき指数 $e = E + bias$ (E はバイアスされていない指数)
3. 有効数字フィールド (fraction) $f = .b_1b_2\cdots b_{p-1}$

3 つのフィールドの値から, エンコードされる値を表 2, 表 3に示し, ビットイメージを図 1に示す.

表 2: 単精度の記憶形式

s, e, f	値
$s = any, e = 255, f \neq 0$	NaN
$s = any, e = 255, f = 0$	$(-1)^s\infty$
$s = any, 0 < e < 255, f = any$	$(-1)^s 2^{e-127}(1.f)$
$s = any, e = 0, f \neq 0$	$(-1)^s 2^{e-126}(0.f)$
$s = any, e = 0, f = 0$	$(-1)^s 0$

表 3: 倍精度の記憶形式

s, e, f	値
$s = any, e = 2047, f \neq 0$	NaN
$s = any, e = 2047, f = 0$	$(-1)^s\infty$
$s = any, 0 < e < 2047, f = any$	$(-1)^s 2^{e-1023}(1.f)$
$s = any, e = 0, f \neq 0$	$(-1)^s 2^{e-1022}(0.f)$
$s = any, e = 0, f = 0$	$(-1)^s 0$

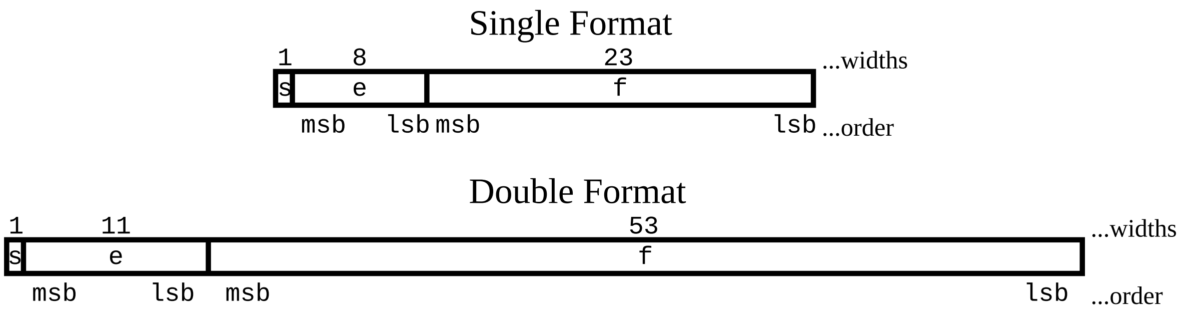

図 1: 基本フォーマットのビットイメージ

3.3 フォーマットの組み合わせ

必ずサポートしなければならないのは, 単精度だけであり (shall), 実装する基本精度 (単精度・倍精度) に対応して, 拡張フォーマット (拡張単精度・拡張倍精度) を実装することを推奨している (should). 何故このような推奨をしているのかについては, 拡張精度を導入する動機の一つとして,10 桁を表示しながらも, 内部では 13 桁の計算を行うような電卓を喩えに挙げて説明することができる. 基本精度に相当する 10 桁に対して,13 桁という拡張精度を導入することで, 電卓はユーザに対して正確で単純な計算モデルを提供することができる. そのため, 単体で拡張精度を導入するのではなく, 基本精度に対応した拡張精度の導入を推奨しているのである.

4　丸め

丸めとは, まず無限の精度で計算を行ってみて, そして必要ならば, デスティネーションのフォーマットに合わせた形に修正し, 同時に不正確の例外を発生させるものである. 但し,2 進 $\leftrightarrow$10 進変換において, 不正確の例外は発生しない. 実際には, この規格で述べられているように無限の精度で計算を行うことは不可能であり, 丸めた結果が常に無限の精度で計算したものと同じになることが保証されるように数ビットの保護ビットを用いてこれに対応する.

丸めモードは, 比較と剰余演算を除いた全ての演算に影響を与え, 結果が 0 になるような加 (減) 算の符号だけでなく, オーバーフロー, アンダーフローのしきい値についても影響する.

4.1　最近値への丸め

この規格では, 最近値への丸め (RN : Round to Nearest) をデフォルトの丸めモードとして提供しなければならない (shall). このモードでは表現できる値を, 無限の精度の中間結果に最も近い値にしなければならない (shall). もし, 中間結果に最も近い表現できる値が 2 つあるときは,LSB が 0 になる方を選択しなければならない (shall). しかし, 無限の精度で計算した中間結果が, $2^{E_{max}}(2-2^{-p})$ を越える場合, 丸めの精度を変更していなければ, 丸めた結果は ∞ となる.

4.2　有向丸め

最近値への丸めモードだけでなく, 有向丸めモードもユーザに選択権を与えなければならない (shall). 有向丸めモードとは,$+\infty$ 方向への丸め (RP : Round to $+\infty$),$-\infty$ 方向への丸め (RM : Round to $-\infty$),0 方向への丸め (RZ : Round to Zero) の 3 つのモードを指す.

$+\infty$ 方向への丸めは, 無限の精度の中間結果に対して,$+\infty$ の方向に向かってそのフォーマットで最も近い値をとらなければならず, また, 中間結果よりも小さな値になることはない (shall). $-\infty$ 方向への丸めは, 無限の精度の中間結果に対して,$-\infty$ の方向に向かってそのフォーマットで最も近い値をとらなければならず, また, 中間結果よりも大きな値になることはない (shall). 0 方向への丸めは, 無限の精度の中間結果に対して,0 の方向に向かってそのフォーマットで最も近い値をとらなければならず, また, 絶対値で比較して, 中間結果よりも大きな値になることはない (shall).

有向丸めにおいて,RP で丸めを行うと, 誤差の上限がわかり,RM は誤差の下限がわかる. また,RZ の丸めを行うと, 誤差の絶対値の上限がわかる. RP と RM の利用価値は, 上限と下限の両方を保証して結果を返す演算 (区間演算) にある. 区間演算を用いると, 誤差の範囲を正確に保証することができる.

4.3　丸めの精度

通常は, デスティネーションの精度と同じ精度で結果は丸められるものであるが, システムの中には, 倍精度もしくは拡張精度しかデスティネーションが選択できないものがある. そのようなシステムにおいてユーザ (大抵は高水準言語のコンパイラ) は, 単精度で丸められるような仕様にしておかなければならない.

5　命令

この規格では, 以下の命令を備えなければならない (shall).

- 加減乗除
- 平方根

- 剰余
- 浮動小数点の整数への丸め
- 異なる浮動小数点フォーマットへの変換
- 浮動小数点フォーマットと整数フォーマット間の変換
- 2 進 $\leftrightarrow$10 進変換
- 比較

これらの命令は, まず無限の精度で計算を行い, その後デスティネーションのフォーマットに合わせて修正を行わなければならない (shall).

5.1 算術演算

算術演算に関しては, 全てのフォーマットの 2 オペランドに対して, 加減乗除と剰余演算ができるように実装しなければならない (shall). また,2 つのオペランドのフォーマットが互いに異なるものでも演算ができることが好ましい (should). そして丸めの精度に関わらず, デスティネーションのフォーマットは 2 つのオペランドのフォーマットよりも精度の大きなものにしなければならない. 即ち, デスティネーションの精度は, 入力オペランドの持つ精度よりも粗い精度にしてはいけない. この精度に関して, 入力オペランドが倍精度の 2 次元ノルムである $\sqrt{x^2+y^2}$ の計算を例にとって規格外の演算というものを考える. この計算は自乗の和の計算と, 平方根の 2 段階で行われるので,(x^2+y^2) を拡張倍精度で計算して, それから, 平方根計算について精度を落として倍精度の結果を出したとする. このような演算を内部で行うと, 丸めの問題からシステムによって異なる答えを返す可能性がある. それを防ぐために精度を落とす演算環境は禁止されているのである.

剰余演算 $r = (x \text{ REM } y)$ において,$y \neq 0$ のとき,r は丸めモードに関わらず, 数学的な $r = x - y \times n$ (n は整数で,x/y の正確な最近値) という関係で定義される. ところで $|n - x/y|$ が正確に $\frac{1}{2}$ のときは, 最近値 n は 2 通りの整数を取り得ることになるが, 規格ではこのような時は,n は偶数とすると定められている. 即ち, 剰余演算は丸めモードや丸めの精度等によらず, 常に正確な一意の値を返す.

5.2 平方根演算

平方根演算は, 実装されている全てのフォーマットで計算できなければならない (shall). オペランドが正の値の時, 結果の符号は正であるが,-0 の平方根は,-0 としなければならない (shall).

5.3 浮動小数点フォーマット変換

サポートしている全てのフォーマット間について, フォーマットの変換ができなければならない (shall). 低い精度へのフォーマット変換については, 4節で規定した通りに丸めを行わなければならない (shall). また高い精度へのフォーマット変換については, 結果は常に正確である.

5.4 浮動小数点 $\leftrightarrow$ 整数フォーマット変換

サポートされている全ての浮動小数点と整数フォーマットについて変換ができなければならない (shall). 浮動小数点から整数への変換は,4節で規定された通りの丸めを行わなければならない (shall).7節で規定している例外が起きない限り, 浮動小数点 $\leftrightarrow$ 整数フォーマット変換は正確でなければならない (shall).

5.5 浮動小数点の整数への丸め

規格では全ての浮動小数点について, 値を整数に丸めて, オペランドと同じ浮動小数点フォーマットに結果を返せなければならない (shall). 丸めの規則は 4節で規定した通りとする.

5.6 2 進 $\leftrightarrow$10 進変換

最低 1 つの決められたフォーマットでの 10 進文字列とサポートされている全ての 2 進浮動小数点との間で, 変換命令を実装しなければならない (shall). 変換をサポートしなければならない値の範囲は, 表 4, 5 の通りとする. 表中の整数 M と N は,$\pm M \times 10^{\pm N}$ という値を示すパラメータとする. 但し,M を表の範囲内で, 連続している 0 のビット列 (trailing zeros) を最下位に加えたり, 取り除いたりすることで,N が最小になるように解釈しなければならない. 例えば, 単精度において, 仮に $M = 1, N = 107$ で表される数値 1×10^{107} は,M を調節することで,$M = 100000000, N = 99$ 即ち 100000000×10^{99} と解釈できるので, 変換の範囲内として取り扱わなければならない.

表 4: 10 進 $\rightarrow$ 2 進変換の範囲

フォーマット	MAX M	MAX N
単精度	$10^9 - 1$	99
倍精度	$10^{17} - 1$	999

表 5: 2 進 $\rightarrow$ 10 進変換の範囲

フォーマット	MAX M	MAX N
単精度	$10^9 - 1$	53
倍精度	$10^{17} - 1$	340

表 4,5と後述する表 6において, 整数 M が表の範囲を越えたときはオプションで, 単精度なら 9 桁目, 倍精度なら 17 桁目以降の桁を他の数字 (通常は 0) で置き換えるようにすることができる. 表 6で規定された範囲内において, 変換は 4節で規定した通りの正確な丸めを行わなければならない.

もし, 表 6の範囲外の変換を行う際, 丸めモードが最近値への丸めモードになっているときは変換による誤差が 0.47ulp(Unit in the Last Place : デスティネーションの LSB を 1 単位とした値) を越えてはならない (shall). 丸めモードが有向丸めモードのときは, 正確な符号と 1.47ulp 以下の誤差を保証しなければならない (shall).

表 6: 10 進 $\leftrightarrow$ 2 進変換における正確な丸めの保証範囲

フォーマット	MAX M	MAX N
単精度	$10^9 - 1$	13
倍精度	$10^{17} - 1$	27

変換命令は,2 進浮動小数点の値が増加したにも関わらず, それらを 10 進に変換を行った際, 値が減少する

ことは許されない (shall). 同様に, 増加していく 10 進の値を 2 進に変換する際, その 2 進の値が減少することは許されない (shall). 即ち,2 進で保証されている大小関係は,10 進に変換しても保たれなければならない.

最近値への丸めモードで, 値が表 4, 5の範囲内にある 2 進浮動小数点を 10 進に変換して, それをまた再び 2 進数に戻した時, 変換後の 2 進数から元の 2 進数を一意に識別できなければならない (shall). また, 表 4, 5の範囲を言い換えれば, この規則の適用範囲は単精度なら 10 進数で 9 桁, 倍精度なら 10 進数で 17 桁以内の数値ということになる.

もし,10 進から 2 進への変換でオーバーフロー/アンダーフローが起きた場合, 7節で定められた通りの例外処理を行う必要がある.2 進から 10 進の変換の際, オーバーフロー, アンダーフロー, 非数 (NaN), 無限大 (∞) と判断される場合は, それらを示す適切な文字列をユーザに提供することが好ましい (should). また,10 進文字列の NaN の表現は, この規格では規定しない.

一貫性を保つために,2 進 $\leftrightarrow$10 進変換のプロシージャは, 実際に変換を行うのが, インタプリタ・コンパイラ・アセンブラ等の言語変換なのかランタイム・対話的入出力等のプログラムの実行なのかによらず, 常に同じ結果を返すことが好ましい (should).

5.7 比較

比較命令は, サポートされている全ての浮動小数点について行えなければならない (shall). そしてそれは異なるフォーマット間の比較についても適用される. 比較は常に正確で, オーバーフローやアンダーフローを引き起こすことはない. 4 つの排他的な関係にある条件 *less than, equal, greater than, unordered* が設定できる. 最後のケースである *unordered* は, 少なくとも 1 つのオペランドが NaN のときの比較の際に発生する条件であり,NaN はどのような値との比較も *unordered* でなければならない (shall). これを利用した例として, ある値 x が NaN かどうか調べるときは,x と x を比較して *unordered* なら NaN であると判断できる. 比較に際しては, ゼロの符号は無視, 即ち $-0 = +0$ でなければならない (shall).

比較の結果は, 実装者の二者択一のオプションとして規定されている. 一つは, 上記の 4 つの条件を特定することができる条件コードを結果とするものであり, もう一つは, 特定した条件 (predicate) に対する比較に true/false を返す方法である. また,true/false 型の場合, 比較結果が *unordered* の時は, 表 7の条件 (最終列の YES/NO) に基いて, 後述する無効演算の例外を出さなければならない. 表中の特定条件の項目は,*ad hoc* 表現に加えて FORTRAN ライクにも書かれており, 関係の項目の T/F は,True/False を意味し, 比較不能の条件は ? で表記しており,4 つの T/F で (FORTRAN ライク表現と等価な)1 つの状態を表していると考えることができる.

比較命令としてサポートしなければならないのは, 表 7の内, 最初の 6 つであり (shall), 更に 7 つ目の条件 (*unordered*) の実装を推奨している (should). また, これに対応する NOT() の命令についても同様である.

表 7: 比較命令の対応

特定条件 (in FORTRAN)	関係 (GT,LT,EQ,UN)	例外発生?
= (.EQ.)	(F,F,T,F)	NO
? <> (.NE.)	(T,T,F,T)	NO
> (.GT.)	(T,F,F,F)	YES
>= (.GE.)	(T,F,T,F)	YES
< (.LT.)	(F,T,F,F)	YES
<= (.LE.)	(F,T,T,F)	YES
? (unordered)	(F,F,F,T)	NO
<> (.LG.)	(T,T,F,F)	YES
<=> (.LEG.)	(T,T,T,F)	YES
? > (.UG.)	(T,F,F,T)	NO
? >= (.UGE.)	(T,F,T,T)	NO
? < (.UL.)	(F,T,F,T)	NO
? <= (.ULE.)	(F,T,T,T)	NO
? = (.UE.)	(F,F,T,T)	NO
NOT(>)	(F,T,T,T)	YES
NOT(>=)	(F,T,F,T)	YES
NOT(<)	(T,F,T,T)	YES
NOT(<=)	(T,F,F,T)	YES
NOT(?)	(T,T,T,F)	NO
NOT(<>)	(F,F,T,T)	YES
NOT(<=>)	(F,F,F,T)	YES
NOT(? >)	(F,T,T,F)	NO
NOT(? >=)	(F,T,F,F)	NO
NOT(? <)	(T,F,T,F)	NO
NOT(? <=)	(T,F,F,F)	NO
NOT(? =)	(T,T,F,F)	NO

6 無限大, 非数, 符号付きゼロ

6.1 ∞ に関する演算

無限大 (∞,INF) は,$-\infty <$(あらゆる有限数)$< +\infty$ として解釈しなければならない (shall). ∞ に対する演算は, 常に正確であり, かつ 7節で規定されている演算を除いて, 例外のシグナルを発生させることはない (shall). ∞ に付随する例外は以下の通りである.

1. ∞ がオーバーフロー,0 除算によって生成され, かつトラップがディゼーブルであるとき
2. ∞ が無効演算である場合

6.2　NaN に対する命令

NaN については, 全ての命令に対して,Signaling NaN と Quiet NaN の異なる 2 種類の NaN をサポートしなければならない (shall). Signaling NaN は, 初期化されていない変数や, 規格で規定されていない, 虚数や巨大な範囲を扱うといった規格の拡張の際に用いられる. Quiet NaN は, 無効若しくは利用不可能なデータや演算結果を, さかのぼって判定する診断情報として利用することができる (should). それ以外の Quiet NaN の利用に関しては, 実装者の裁量に任せられる. 診断情報の伝播において要求されることは, NaN(s) に含まれている情報を, 数値演算命令と浮動小数点フォーマット変換命令の際に保存しておくことである (should).

Signaling NaN(s) は,5節で挙げた全ての命令について, 後述する無効演算例外のシグナルを送るオペランドとして予約されている (shall). フォーマットの変更なく,Signaling NaN をコピーしたときに, 無効演算例外を発生させるかどうかは, 実装者に選択権がある.

Signaling NaN を含んだ全ての演算と, 無効演算について, もしトラップが起こらないで, かつ浮動小数点の結果が返されるべきものであるならば, 結果として Quiet NaN を返さなければならない (shall).

入力に 1 つないし 2 つの NaN を含む全ての演算は, そのどちらも Signaling NaN でないときは, 例外のシグナルは発してはならない. そして, 浮動小数点の結果が返されるべきものであるならば, 結果として Quiet NaN を返さなければならない (shall). そして結果として返すその Quiet NaN は, 入力のうちのどちらかの NaN を, そのまま返すことが望ましい (should). しかしフォーマット変換の場合は, 全く同じ NaN を伝播させることはできないことに注意しなければならない. Quiet NaNs は, 浮動小数点フォーマットの結果を返さない命令における Signaling NaNs のふるまいと, ほぼ同じ効果を持っている (その命令とは即ち, 既に述べられている,NaN を伴わない比較, フォーマットの変換のことである).

6.3　符号ビット

規格では,NaN の符号を解釈することはない. その NaN 以外のケースにおいて

- 積と商の符号は,2 つのオペランドの符号の xor をとった値である
- 減算 $x - y$ を $x + (-y)$ という加算とみなすと, 被加算数の符号が 2 つとも同じ符号であるならば, 和の符号もそれと同じ符号をとらなければならない
- 浮動小数点を整数に丸める命令の結果の符号は, オペランドの符号と等しい
- 異符号同士の和 (または同符号同士の差) が丁度 0 になったとき, その和 (差) は
  - 丸めモードが RM($-\infty$ 方向への丸め) のときは -0
  - それ以外のモードの場合は $+0$

  を返す
- しかし,$x = \pm 0$ のときに $x + x$ または $x - (-x)$ を行ったら, その x の符号が保持されたままそれが結果となる, 即ち, 結果として x をそのまま返す

  上の 2 点をまとめると, 表 8のようになる

表 8: 結果が 0 となる加減算演算

計算	丸めモード	結果
(+0)ADD(+0) (+0)SUB(−0)	(don't care)	+0
(−0)ADD(−0) (−0)SUB(+0)	(don't care)	−0
(+0)ADD(−0) (+0)SUB(+0) (−0)ADD(+0) (−0)SUB(−0) x SUB x x ADD $(-x)$	RN,RZ,RP	+0
(+0)ADD(−0) (+0)SUB(+0) (−0)ADD(+0) (−0)SUB(−0) x SUB x x ADD $(-x)$	RM	−0

x は任意の非ゼロの有限数

- $\sqrt{-0} = -0$ であることを除いて, 全ての平方根演算は, 正の符号を返す

7 例外

規定された 5 種類の例外のいずれかを検出したときに, シグナルを発生しなければならない (shall). シグナルとは, ステータスフラグを立てる, または, トラップを発生すること, 若しくはその両方のことである. 各々の例外のトラップは,8節で規定しているように, ユーザの制御下におくことを推奨する (should). 例外に対するデフォルトの応答は, トラップ無しに処理を続行させることである (shall). 規格では, 結果について, トラップと非トラップの両方を規定している. そしてトラップがイネーブルかどうかで結果が変化するものがある (この代表的なものが, アンダーフローに対するふるまいである).

各々の例外について, 対応したトラップが発生しないとき, 代わりに値がセットされるようなステータスフラグを提供しなければならない (shall). このステータスフラグは, ユーザの要求によってのみリセットされる (スティッキーである). ユーザはステータスフラグを独立にテスト, 入れ替えができ, なおかつ 5 つのフラグを一度にセーブ, リストアできなければならない (shall).

複数同時に起こり得る例外は, 不正確とオーバーフローの例外と, 不正確とアンダーフローの例外の 2 つだけである (つまりそれ以外の例外は 1 つずつしか起こらない).

7.1 無効演算

規格では, 命令を遂行することが意味が無いようなオペランドを持つときに, 無効演算例外が発生する. その結果は, トラップがなく例外が発生したとき, Quiet NaN を浮動小数点に対応したフォーマットでデスティネーションに提供しなければならない (shall). 無効演算の種類は以下の通りである.

- Signaling NaN に関する全ての演算
- $(+\infty) - (+\infty)$ のような無効な $\pm\infty$ の加減算
- $0 \times \infty$
- $0/0$ と ∞/∞
- 剰余演算 x REM y において,x が ∞ または y が 0 であるとき
- 0 より小さな値の平方根演算
- 2 進浮動小数点から, 整数または 10 進のフォーマットに変換する際, オーバーフロー・∞・NaN によって, 正しい変換が行うことができないとき
- 比較不能 (*unordered*) な 2 数について,?(*unordered*) を含まない条件 ($<>$ 等) で比較命令を行ったとき

7.2 0 除算

除数が 0 で, 被除数が有限の非ゼロの数であるならば,0 除算例外をシグナルを発しなければならない (shall). そして, トラップが起こらないならば, 結果として適切な符号付きの ∞ を返さなければならない (shall).

7.3 オーバーフロー

オーバーフローの例外シグナルは, 指数を限りなくとれると仮定して丸められた浮動小数点の結果が, 絶対値で比較して, デスティネーションのフォーマットの最大の有限の数を越えた場合に発生する (shall). そしてその結果は, トラップが起こらないなら, 丸めモードと, 中間結果の符号によって決定される.

1. 丸めモードが RN なら, オーバーフローのとき, 中間結果の符号付き ∞ を返す.
2. 丸めモードが RZ なら, オーバーフローのとき, フォーマットで表現できる最大値を, 中間結果の符号をつけて返す.
3. 丸めモードが RM なら, 正のオーバーフローのとき, フォーマットで表現できる最大値を返し, 負のオーバーフローのとき,$-\infty$ を返す.
4. 丸めモードが RP なら, 負のオーバーフローのとき, フォーマットで表現できる負の最大値を返し, 正のオーバーフローのとき,$+\infty$ を返す.

以上をまとめると, 表 9のようになる.

表 9: オーバーフロー時における丸めモードと結果の関係

丸めモード	中間結果の符号	結果
RN	$+$	$+\infty$
	$-$	$-\infty$
RZ	$+$	$+$(表現しうる最大値)
	$-$	$-$(表現しうる最大値)
RP	$+$	$+\infty$
	$-$	$-$(表現しうる最大値)
RM	$+$	$+$(表現しうる最大値)
	$-$	$-\infty$

フォーマット等の変換命令を除く全ての命令について, トラップを発生させるオーバーフローは, 無限の精度で計算した中間結果を,2^{α} で割り, それを丸めたものをハンドラに渡さなければならない (shall). ここで α とは, 指数フィールドが 8 ビットの単精度なら 192 ,11 ビットの倍精度なら 1536,n ビットの拡張精度なら $3 \times 2^{n-2}$ である. つまり, 正確に計算した中間結果の指数フィールドから,$3 \times 2^{n-2}$ を引いて結果を返し, かつフォーマットに合うように仮数を正規化して返す. 2 進浮動小数点からのフォーマット変換において, トラップ型のオーバーフローが起きた場合には, できるだけデスティネーションの精度で丸めを行いながらも, 指数のバイアスを調整して, その 2 進のフォーマットと同じかそれよりも広範囲のフォーマットを結果としてハンドラに送らなければならない (shall). 10 進から 2 進への変換でトラップ型オーバーフローが起きたときは, できるだけ デスティネーションの精度で丸めを行いながらも, 指数のバイアスを調整して, サポートされている最も大きなフォーマットを結果としてハンドラに送らなければならない (shall). しかし結果が表現できる範囲から極めて離れていて調整することができない場合は,quiet NaN を代わりに送らなければならない (shall).

7.4 アンダーフロー

アンダーフローには 2 つの相関のあるイベントが関係している.1 つは $\pm 2^{E_{min}}$ の範囲内にある, 小さな非ゼロの結果の創出で, これは, 余りにも小さすぎて, 例えば割り算でオーバーフローを引き起こすように, 後で他の例外を引き起こす可能性があるというものである. もう 1 つはデノーマルによる, 非常に小さな数の近似のために, 極端に精度が悪化してしまうことである. 実装者はどのようにそれらのイベントを検出するかを, 与えられた方法の中から選択可能であるが, 全ての命令について一貫した方法でそれらのイベントを検出しなければならない (shall). その検出法は 4 種類に分けることができる. 最初の 2 つが値の小ささ (Tininess) についての判定基準で, 後半 2 つが精度落ち (Loss of Accuracy) についての判断基準である.

1. 丸め後検出 (After rounding)

   指数の範囲が無限であるかのようにして (有効数字は有限=丸めて) 計算した非ゼロの結果が, 厳密に $\pm 2^{E_{min}}$ の間に存在するとき

2. 丸め前検出 (Before rounding)

   指数と精度の両方の範囲が無限であるかのようにして (=丸めないで) 計算した非ゼロの結果が, 厳密に $\pm 2^{E_{min}}$ の間に存在するとき

3. デノーマルロスによる検出 (A denormalization loss)

計算した結果と, 指数の範囲が無限であると仮定した結果が異なっているとき, 即ち, ある値を表現するのに, 正規化数と非正規化数とで異なった場合に検出される

4. 不正確による検出 (An inexact result)

   計算した結果と, 指数と仮数の両方の範囲が無限であると仮定した結果が異なっているとき (この条件を不正確として後述している. つまり, 不正確であるならば検出するという方法である)

丸め前検出と, 丸め後検出で結果が異なる例としては,$2^{E_{min}}$ よりも僅かに小さいが, 丸めるとちょうど $2^{E_{min}}$ になってしまう数がある. このような数は, 丸め前検出のときは Tininess が検出されず, 丸め後検出では Tininess が検出される. アンダーフロートラップが実装されていないとき, 又はイネーブルされていないとき (これがデフォルトである), 値が小さく (Tininess) かつ精度が失なわれたとき (Loss of Accuracy) のみ, アンダーフローはシグナルを発生させる (アンダーフローフラグを立てる). つまり計算結果がデノーマルになっても, その結果が丸められておらず正確ならば, アンダーフローフラグを立ててはならない (shall). Tininess と Loss of Accuracy を検出するための方法は, 結果に影響を与えてはいけない. また, その結果とは, ゼロかデノーマルか $\pm 2^{E_{min}}$ のはずである. アンダーフロートラップが実装され, かつイネーブルの時,Tininess が検出されていたなら,Loss of Accuracy の検出の有無に関わらず アンダーフローのシグナルを発生しなければならない (shall). フォーマット等の変換命令を除く全ての命令について, トラップを発生させるアンダーフローは, 無限の精度で計算した中間結果に 2^{α} をかけ , それを丸めたものをハンドラに渡さなければならない (shall). ここで α とは, 指数フィールドが 8 ビットの単精度なら 192 ,11 ビットの倍精度なら 1536,n ビットの拡張精度なら $3 \times 2^{n-2}$ である. つまり, 正確に計算した中間結果の指数フィールドから, $3 \times 2^{n-2}$ を足して結果を返し, しかもフォーマットに合うように仮数を正規化して返さなければならない. 変換命令において, トラップを起こすアンダーフローは, トラップを起こすオーバーフローの時の動作とアナロジーな動作をしなければならない (shall).

7.5 不正確

命令の結果を丸めたものが正確ではないとき, 又はオーバーフローのトラップを起こさない状態で結果がオーバーフローしたとき, 不正確例外をシグナルしなければならない (shall). また, 丸められた若しくはオーバーフローした結果をデスティネーションに送られなければならない (shall). もし, 不正確のトラップが発生するならば, それをトラップハンドラに渡さなければならない (shall).

8 トラップ

ユーザは,5 つのうち任意の例外が起きたとき, 決められたハンドラに処理を渡すという形でトラップを要求できる (should). また, 存在しているハンドラについて, ディゼーブル, 保存, リストアを要求できる (should). ユーザは, 意図した例外のトラップハンドラがイネーブルされているかどうかを判定することができる (should). トラップがディゼーブルで, 例外シグナルが発生したときは, 7節で規定した方法に従わなければならない.(shall). トラップがイネーブルの状態で, 例外のシグナルが発生したときは, そのプログラムの実行はサスペンドされ, ユーザによって前もって用意されているトラップハンドラを実行しなければならない (shall), そして,7節で規定された例外の規則に従って結果を返さなければならない (shall).

8.1 トラップハンドラ

トラップハンドラは, 例外を発生させた命令に代わって結果の値を返すサブルーチンの能力を持たせることを推奨している (should). そしてトラップハンドラが結果を返さないならば, その値は未定義とする. 同

様に, トラップがイネーブルの状態でシグナルが発生したとき, 例外に対応した各々のフラグは, トラップハンドラによってセット又はリセットされていないならば未定義とする.

トラップが発生したとき, ハンドラは以下のことについて判断できる (should).

1. どの例外が起きたか (例外の種類)
2. どの命令で起きたか (命令の種類)
3. 結果のフォーマット
4. オーバーフロー, アンダーフロー, 不正確の例外の場合は, デスティネーションのフォーマットに合わないという情報を含めた, 正確に丸められた結果
5. 無効演算若しくは 0 除算による例外の場合は, オペランドの値

8.2 トラップの優先

また, トラップがイネーブルであるならば, オーバーフローとアンダーフローは, 同時に起きた不正確のトラップに対して優先権を持つ. これらの例外以外は同時に発生することがないので, これら 2 つのトラップのみ優先権を与えておけばよい.

9 付録：推奨関数

IEEE754 規格には付録として, 異なるシステム間でのプログラムの移植性の観点から実装を推奨する 10 個の関数が挙げられている. しかし同時に, この付録は規格の一部ではないことも主張している. 以下に示した関数の中で, (1),(2),(6) のようにフォーマットの変換が行われない関数は,signaling NaN に対して無効演算の例外を発生するかどうかについては実装者の権限に任せられる.

1. Copysign(x, y)

   y の符号を,x にコピーして返す. 例えば,Copysign$(a, 1.0)$ を実行すると,a の絶対値 abs(a) が返ってくる.

2. Reversesign(x)

   入力 x の符号を反転させて返す. この関数は $0-x$ とは異なる. 具体的には, $x = \pm 0, x =$NaN のときに違いが見られる[1].

   - $x = +0$ のとき,$0-x$ は, 丸めモードが RM 以外のとき,$+0$ を返す
   - $x =$NaN のとき,$0-x$ は,NaN を返す

3. Scalb(y, N)

   $y \times 2^N$ を返す関数.N は整数で, 具体的に 2^N という値を計算することなく, 直接浮動小数点を生成する.

4. Logb(x)

   x のげたばきでない指数 (符号付き整数) を返す. 但し

   - Logb(NaN)=NaN

[1]付録にはこのように書かれているが, 実際には,$x = -0$ のときは違いは無い

- Logb(∞)$= +\infty$
- Logb(0)$= -\infty$ (更に 0 除算例外を発生する)

である.

x が正の有限数のとき,Scalb$[x, -\text{Logb}(x)]$ は厳密に 0 と 2 の間にあり, 特に 1 より小さくなるのは不正規化数のときだけである

5. Nextafter(x, y)

   表現可能な数値を並べたとき,y への方向に対して,x の隣にある数値を返す.

   - $x = y$ ならば, いかなる例外も起こすことなく x を返す.
   - x, y どちらかが Quiet NaN なら, 入力と全く同じ NaN かそれ以外の NaNs を返す
   - x が有限でも,Nextafter(x, y) が ∞ のときは, オーバーフローと不正確の例外を出す
   - Nextafter(x, y) が, 厳密に $\pm 2^{E_{min}}$ の中にあるなら, アンダーフローと不正確の例外を出す

6. Finite(x)

   $-\infty < x < +\infty$ なら TRUE を, そうでなければ FALSE を返す.

7. Isnan(x)

   x が NaN かどうかを調べる. 実際には,x と x を比較してその結果, equal にならなければ NaN であると判断できる.

8. GraterOrLess(x, y)

   $x > y$ または $x < y$ のときのみ TRUE を返す.

   比較不能 (*unordered*) という条件があるので,「x と y は同じではない $(x \neq y)$」と区別しなければいけない

9. Unordered(x, y)

   比較不能のときに TRUE を返す.

10. Class(x)

    x を 10 種類のクラスに分類する.

    - Signaling NaN
    - Quiet NaN
    - $-\infty$
    - 非ゼロ, 負の正規化数
    - 負のデノーマル
    - -0
    - $+0$
    - 正のデノーマル
    - 非ゼロ, 正の正規化数
    - $+\infty$

    この関数は, 例え入力が Signaling NaN であったとしても, 決して例外は発生しない.