

P1788

Draft International Standard for Interval Arithmetic and Complete Arithmetic

Sponsor:

Microprocessor Standards Committee

Abstract: This International Standard specifies representations and methods for interval arithmetic and complete arithmetic.

An implementation of a system conforming to this International Standard can be realized entirely in software, entirely in hardware, or in any combination of software and hardware. For operations specified in the normative part of this International Standard, numerical results and exceptions are uniquely determined by the values of the input data, sequence of operations, and destination formats, all under user control.

Keywords: computer, interval arithmetic, complete arithmetic.

Copyright ©2008 by the IEEE
Three Park Avenue
New York, New York 10016-5997, USA
All rights reserved.

This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. **USE AT YOUR OWN RISK!** Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of international standardization consideration. Prior to adoption of this document, in whole or in part, by another standards development organization permission must first be obtained from the Manager, Standards Intellectual Property, IEEE Standards Activities Department. Other entities seeking permission to reproduce this document, in whole or in part, must obtain permission from the Manager, Standards Intellectual Property, IEEE

Standards Activities Department.
IEEE Standards Activities Department
Manager, Standards Intellectual Property
445 Hoes Lane
Piscataway, NJ 08854, USA

Patent statement

Attention is called to the possibility that implementation of this International Standard might require use of subject matter covered by patent rights. By publication of this International Standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license might be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. The IEEE makes no representation as to the reasonableness of rates, terms, and conditions of the license agreements offered by any patent holders or patent applicants. Further information may be obtained from the IEEE Standards Department.

Introduction

This introduction is not part of P1788, Draft International Standard for Interval Arithmetic and Complete Arithmetic

This International Standard is a product of the Interval and Complete Arithmetic working group of, and sponsored by, the Microprocessor Standards Subcommittee of the IEEE Computer Society.

This International Standard provides a discipline for performing computation that yields results independent of whether the processing is done in hardware, software, or a combination of the two. For operations specified in the normative part of this International Standard, numerical results are uniquely determined by the values of the input data, the operation, and the destination, all under user control.

This International Standard defines a family of commercially feasible ways for systems to perform computation. Among the desiderata that guided the formulation of this International Standard were:

- a) Facilitate movement of existing programs from diverse computers to those that adhere to this International Standard as well as among those that adhere to this International Standard.
- b) Enhance the capabilities and safety available to users and programmers who, although not expert in numerical methods, might well be attempting to produce numerically sophisticated programs.
- c) Encourage experts to develop and distribute robust and efficient numerical programs that are portable, by way of minor editing and recompilation, onto any computer that conforms to this International Standard and possesses adequate capacity. Together with language controls it should be possible to write programs that produce identical results on all conforming systems.
- d) Provide arithmetic that is exact for very long sequences of computation.
- e) Enable rather than preclude further refinements and extensions.

In programming environments, this International Standard is also intended to form the basis for a dialog between the numerical community and programming language designers.

Participants

The following participants in the Interval and Complete Arithmetic working group contributed to the development of this International Standard:

R. Baker Kearfott

John Pryce

Nathalie Revol

The following individual members of the balloting committee voted on this International Standard. Balloters might have voted for approval, disapproval, or abstention.

R. Baker Kearfott

John Pryce

Nathalie Revol

Contents

1	Overview	7
1.1	Scope	7
1.2	Purpose	7
1.3	Inclusions	7
1.4	Exclusions	7
1.5	Programming environment considerations	8
1.6	Word usage	8
1.7	Normative references	9
2	Definitions, abbreviations, and acronyms	9
2.1	Definitions	9
2.1.1	Definitions from IEEE P754	9
2.1.2	lower bound	9
2.1.3	upper bound	9
3	Representation of intervals	9
4	Interval operations	10
4.1	General	10
4.2	Interval arithmetic operations	10
4.3	Interval comparison and lattice operations	14
5	Complete arithmetic	15
5.1	Complete formats	15
5.2	Complete interchange format encodings	16
6	Attributes and rounding	17
7	Operations of complete arithmetic	17
7.1	General	17
7.2	Operations in addition to those specified in IEEE P754	18
7.2.1	exactFormatOf arithmetic operations	18
7.2.2	Conversion operations for all formats	18
7.2.3	Conversion operations for binary formats	19

7.3	Sign operations	19
7.4	Comparisons	20
7.5	General operation	20
8	Infinity, NaNs, and sign bit	21
8.1	General	21
8.2	Infinity arithmetic	21
8.3	Operations with NaNs	21
8.4	NaN encodings in complete formats	22
8.5	Invalid operations	22
8.6	Overflow	22
8.7	Underflow	23

List of Tables

4.1	Result of intervalMultiplication	11
4.1	Result of intervalMultiplication (cont.)	12
4.2	Result of intervalDivision when <i>source₂</i> does not contain zero	12
4.3	Result of intervalDivision when <i>source₂</i> contains zero	13
4.3	Result of intervalDivision when <i>source₂</i> contains zero (cont.)	13
5.1	Complete format parameters for floating-point operands	15
5.1	Complete format parameters for floating-point operands (cont.)	16

List of Figures

5.1	Binary interchange, binary complete format	16
5.2	Binary interchange, decimal complete format	17

P1788

Draft International Standard for Interval Arithmetic and Complete Arithmetic

1 Overview

1.1 Scope

This International Standard specifies formats and methods for interval arithmetic and complete arithmetic.

1.2 Purpose

This International Standard specifies methods for computation with numeric intervals and with high-precision fixed-point numbers. Each method yields identical results whether the processing is done in hardware, software, or a combination of the two. The results of either computation will be identical, independent of implementation, given the same input data. Errors, and error conditions, in the mathematical processing shall be reported in a consistent manner regardless of implementation.

1.3 Inclusions

This International Standard specifies:

- representation of intervals in terms of floating-point number formats specified by IEEE P754
- formats for high-precision fixed-point data, for computation and data interchange
- addition, subtraction, multiplication, division, fused multiply add, compare, and other operations
- conversions between integer or floating-point formats and high-precision fixed-point formats
- conversions between different high-precision fixed-point formats
- conversions between high-precision fixed-point formats and external representations as character sequences

1.4 Exclusions

This International Standard does not specify:

- formats or methods for floating-point data, or computations using floating-point data
- floating-point exceptions and their handling, including data that are not numbers (NaNs).

1.5 Programming environment considerations

This International Standard does not define all aspects of a conforming programming environment. Such behavior should be defined by a programming language definition supporting this International Standard, if available, and otherwise by a particular implementation. Some programming language specifications might permit some behaviors to be defined by the implementation.

Language-defined behavior should be defined by a programming language standard supporting this International Standard. Then all implementations conforming both to this International Standard and to that language standard behave identically with respect to such language-defined behaviors. Standards for languages intended to reproduce results exactly on all platforms are expected to specify behavior more tightly than do standards for languages intended to maximize performance on every platform.

Because this International Standard requires facilities that are not currently available in common programming languages, the standards for such languages might not be able to conform fully to this International Standard if they are no longer being revised. If the language can be extended by a function library or class or package to provide a conforming environment, then that extension should define all the language-defined behaviors that would normally be defined by a language standard.

Implementation-defined behavior is defined by a specific implementation of a specific programming environment conforming to this International Standard. Implementations define behaviors not specified by this International Standard nor by any relevant programming language standard or programming language extension.

Conformance to this International Standard is a property of a specific implementation of a specific programming environment, rather than of a language specification.

However a language standard could also be said to conform to this International Standard if it were constructed so that every conforming implementation of that language also conformed automatically to this International Standard.

1.6 Word usage

In this International Standard three words are used to differentiate between different levels of requirements and optionality, as follows:

- **may** indicates a course of action permissible within the limits of this International Standard with no implied preference (“may” means “is permitted to”)
- **shall** indicates mandatory requirements strictly to be followed in order to conform to this International Standard and from which no deviation is permitted (“shall” means “is required to”)
- **should** indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not

necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (“should” means “is recommended to”).

Further:

- **might** indicates the possibility of a situation that could occur, with no implication of the likelihood of that situation (“might” means “could possibly”)
- **see** followed by a number is a cross-reference to the clause or subclause of this International Standard identified by that number
- **NOTE** introduces text that is informative (that is, is not a requirement of this International Standard).

1.7 Normative references

The following referenced standards are indispensable for the application of this International Standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced standard (including any amendments) applies.

IEEE P754 (2008-01), *Standard for floating-point arithmetic*.

2 Definitions, abbreviations, and acronyms

2.1 Definitions

2.1.1 Definitions from IEEE P754

For the purposes of this International Standard, the terms and definitions specified in IEEE P754 apply. In addition, the following terms and definitions apply.

2.1.2 lower bound

The boundary of an interval having the value nearest to negative infinity.

2.1.3 upper bound

The boundary of an interval having the value nearest to positive infinity.

3 Representation of intervals

Interval formats consist of two elements. Each element shall be represented by a floating-point number having an encoding specified by IEEE P754. Both elements shall have the same encoding. The first element is the lower bound of the interval, and the second is the upper bound. If a language

provides interval formats it shall provide them in all floating-point encodings it provides for point formats, it shall provide the interval arithmetic operations specified on intervals in subclause 4.2, and should provide the interval comparison and lattice operations specified in subclause 4.3.

[Note to committee: If it would make the proposal more attractive, it would be sufficient to provide interval support only for double precision, *viz.*

Interval formats consist of two elements. Each element shall be represented by a floating-point number having an encoding specified by IEEE P754. Both elements shall have the same encoding. The first element is the lower bound of the interval, and the second is the upper bound. If a language provides interval formats it shall provide them at least in 64 bit binary basic interchange format; it is recommended that the language provide them in all floating-point encodings it provides for point formats. It shall provide the interval arithmetic operations specified on intervals in subclause 4.2 for all interval formats it provides, and should provide the interval comparison and lattice operations specified in subclause 4.3.]

4 Interval operations

4.1 General

An interval is denoted by an ordered pair of objects enclosed in square and/or round brackets. The first object is the left bound of the interval and the second is the right bound. If the bracket adjacent to a bound is round, the bound is not included in the interval; if it is square the bound is included in the interval. Except for a result that represents two disjoint intervals (usually the result of the **intervalDivision** operation), or that represents the empty interval, the left bound shall not be greater than the right bound. Where an interval is denoted by a single letter, the left bound is denoted by that letter with the subscript 1 and the right bound is denoted by that letter with the subscript 2.

4.2 Interval arithmetic operations

In the description of the operations, the operators $+$, $-$, \times and $/$ are used to indicate the operations **addition**, **subtraction**, **multiplication** and **division**, respectively, as defined in subclause 5.4.1 of IEEE P754. An operator symbol with \vee above it indicates that the denoted operation is performed with rounding-direction attribute **roundTowardNegative**. An operator symbol with \wedge above it indicates that the denoted operation is performed with rounding-direction attribute **roundTowardPositive**. The operations **min** and **max** are defined by the operations **minNum** and **maxNum** in subclause 5.3.1 of IEEE P754, respectively.

Exceptions are not signaled until computations of both bounds are completed. If the same exception arises in both computations it is signaled only once. If several exceptions arise they are signaled as specified in subclause 7.1 of IEEE P754.

An implementation that provides interval formats shall provide the following *formatOf* general-computational operations, for destinations of all supported non-storage floating-point interval formats, and, for each destination format, for operands of all supported non-storage floating-point interval formats with the same radix as the destination format. The result of an operation is denoted by $r = [r_1, r_2]$. These operations never propagate non-canonical results.

[Note to committee: If it makes the project more acceptable, it is sufficient to require interval operations only for double-precision binary operands, and recommend them for all formats, *viz.*

An implementation that provides interval formats shall provide the following *formatOf* general-computational operations, for destinations of basic binary floating-point interval formats of 64 bit length for each bound. It is recommended that implementations provide operations for destinations of all supported non-storage floating-point interval formats, and, for each destination format, for operands of all supported non-storage floating-point interval formats with the same radix as the destination format. The result of an operation is denoted by $r = [r_1, r_2]$. These operations never propagate non-canonical results.]

— *formatOf* **intervalAddition** (*source*₁, *source*₂)

the operation **intervalAddition**(x, y) computes $r := x + y$, where r is defined by $r_1 := x_1 \overset{\vee}{+} y_1$ and $r_2 := x_2 \overset{\wedge}{+} y_2$ unless a bound of an operand is an infinity, in which case the corresponding bound of the result is the same infinity.

— *formatOf* **intervalSubtraction** (*source*₁, *source*₂)

the operation **intervalSubtraction**(x, y) computes $r := x - y$, where r is defined by $r_1 := x_1 \overset{\vee}{-} y_2$ and $r_2 := x_2 \overset{\wedge}{-} y_1$ unless a bound of an operand is an infinity. If a bound of the operand x is an infinity the corresponding bound of the result is the same infinity. If a bound of the operand y is an infinity the opposite bound of the result is the infinity with the opposite sign.

— *formatOf* **intervalMultiplication** (*source*₁, *source*₂)

the operation **intervalMultiplication**(x, y) computes $r := x \times y$, where r is defined by Table 4.1.

Table 4.1 – Result of **intervalMultiplication**

<i>source</i> ₁	<i>source</i> ₂			
	$[y_1, y_2]$ $y_2 \leq 0$	$[y_1, y_2]$ $y_1 < 0 < y_2$	$[y_1, y_2]$ $y_1 \geq 0$	$[0, 0]$
$[x_1, x_2], x_2 \leq 0$	$[x_2 \overset{\vee}{\times} y_2, x_1 \overset{\wedge}{\times} y_1]$	$[x_1 \overset{\vee}{\times} y_2, x_1 \overset{\wedge}{\times} y_1]$	$[x_1 \overset{\vee}{\times} y_2, x_2 \overset{\wedge}{\times} y_1]$	$[0, 0]$
$x_1 < 0 < x_2$	$[x_2 \overset{\vee}{\times} y_1, x_1 \overset{\wedge}{\times} y_1]$	$[\min(x_1 \overset{\vee}{\times} y_2, x_2 \overset{\vee}{\times} y_1),$ $\max(x_1 \overset{\wedge}{\times} y_1, x_2 \overset{\wedge}{\times} y_2)]$	$[x_1 \overset{\vee}{\times} y_2, x_2 \overset{\wedge}{\times} y_2]$	$[0, 0]$
$[x_1, x_2], x_1 \geq 0$	$[x_2 \overset{\vee}{\times} y_1, x_1 \overset{\wedge}{\times} y_2]$	$[x_2 \overset{\vee}{\times} y_1, x_2 \overset{\wedge}{\times} y_2]$	$[x_1 \overset{\vee}{\times} y_1, x_2 \overset{\wedge}{\times} y_2]$	$[0, 0]$
$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
$(-\infty, x_2], x_2 \leq 0$	$[x_2 \overset{\vee}{\times} y_2, +\infty)$	$(-\infty, +\infty)$	$(-\infty, x_2 \overset{\wedge}{\times} y_1]$	$[0, 0]$
$(-\infty, x_2], x_2 \geq 0$	$[x_2 \overset{\vee}{\times} y_1, +\infty)$	$(-\infty, +\infty)$	$(-\infty, x_2 \overset{\wedge}{\times} y_2]$	$[0, 0]$
$[x_1, +\infty), x_1 \leq 0$	$(-\infty, x_1 \overset{\wedge}{\times} y_1]$	$(-\infty, +\infty)$	$[x_1 \overset{\vee}{\times} y_2, +\infty)$	$[0, 0]$
$[x_1, +\infty), x_1 \geq 0$	$(-\infty, x_1 \overset{\wedge}{\times} y_2]$	$(-\infty, +\infty)$	$[x_1 \overset{\vee}{\times} y_1, +\infty)$	$[0, 0]$
$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$[0, 0]$

Table 4.1 – Result of **intervalMultiplication** (cont.)

<i>source</i> ₁	<i>source</i> ₂				
	$(-\infty, y_2]$ $y_2 \leq 0$	$(-\infty, y_2]$ $y_2 \geq 0$	$[y_1, +\infty)$ $y_1 \leq 0$	$[y_1, +\infty)$ $y_1 \geq 0$	$(-\infty, +\infty)$
$[x_1, x_2], x_2 \leq 0$ $x_1 < 0 < x_2$	$[x_2 \overset{\vee}{\times} y_2, +\infty)$ $(-\infty, +\infty)$	$[x_1 \overset{\vee}{\times} y_2, +\infty)$ $(-\infty, +\infty)$	$(-\infty, x_1 \overset{\wedge}{\times} y_1]$ $(-\infty, +\infty)$	$(-\infty, x_2 \overset{\wedge}{\times} y_1]$ $(-\infty, +\infty)$	$(-\infty, +\infty)$ $(-\infty, +\infty)$
$[x_1, x_2], x_1 \geq 0$ $[0, 0]$	$(-\infty, x_1 \overset{\wedge}{\times} y_2]$ $[0, 0]$	$(-\infty, x_2 \overset{\wedge}{\times} y_2]$ $[0, 0]$	$[x_2 \overset{\vee}{\times} y_1, +\infty)$ $[0, 0]$	$[x_1 \overset{\vee}{\times} y_1, +\infty)$ $[0, 0]$	$(-\infty, +\infty)$ $[0, 0]$
$(-\infty, x_2], x_2 \leq 0$	$[x_2 \overset{\vee}{\times} y_2, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, x_2 \overset{\wedge}{\times} y_1]$	$(-\infty, +\infty)$
$(-\infty, x_2], x_2 \geq 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 \leq 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 \geq 0$ $(-\infty, +\infty)$	$(-\infty, x_1 \overset{\wedge}{\times} y_2]$ $(-\infty, +\infty)$	$(-\infty, +\infty)$ $(-\infty, +\infty)$	$(-\infty, +\infty)$ $(-\infty, +\infty)$	$[x_1 \overset{\vee}{\times} y_1, +\infty)$ $(-\infty, +\infty)$	$(-\infty, +\infty)$ $(-\infty, +\infty)$

Note: $-\infty$ is represented by **negativeInfinity** and $+\infty$ is represented by **positiveInfinity**.

— *formatOf* **intervalDivision** (*source*₁, *source*₂)

the operation **intervalDivision**(*x*,*y*) computes $r := x/y$, where r is defined by Table 4.2 if the interval *source*₂ does not contain zero, and by Table 4.3 if the interval *source*₂ contains zero.

Table 4.2 – Result of **intervalDivision** when *source*₂ does not contain zero

<i>source</i> ₁	<i>source</i> ₂			
	$[y_1, y_2]$ $y_2 < 0$	$[y_1, y_2]$ $y_1 > 0$	$(-\infty, y_2]$ $y_2 < 0$	$[y_1, +\infty)$ $y_1 > 0$
$[x_1, x_2], x_2 \leq 0$	$[x_2 \overset{\vee}{/} y_1, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, x_2 \overset{\wedge}{/} y_2]$	$[0, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, 0]$
$[x_1, x_2], x_1 < 0 < x_2$	$[x_2 \overset{\vee}{/} y_2, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, x_2 \overset{\wedge}{/} y_1]$	$[x_2 \overset{\vee}{/} y_2, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, x_2 \overset{\wedge}{/} y_1]$
$[x_1, x_2], x_1 \geq 0$ $[0, 0]$	$[x_2 \overset{\vee}{/} y_2, x_1 \overset{\wedge}{/} y_1]$ $[0, 0]$	$[x_1 \overset{\vee}{/} y_2, x_2 \overset{\wedge}{/} y_1]$ $[0, 0]$	$[x_2 \overset{\vee}{/} y_2, 0]$ $[0, 0]$	$[0, x_2 \overset{\wedge}{/} y_1]$ $[0, 0]$
$(-\infty, x_2], x_2 \leq 0$	$[x_2 \overset{\vee}{/} y_1, +\infty)$	$(-\infty, x_2 \overset{\wedge}{/} y_2]$	$[0, +\infty)$	$(-\infty, 0]$
$(-\infty, x_2], x_2 \geq 0$	$[x_2 \overset{\vee}{/} y_2, +\infty)$	$(-\infty, x_2 \overset{\wedge}{/} y_1]$	$[x_2 \overset{\vee}{/} y_2, +\infty)$	$(-\infty, x_2 \overset{\wedge}{/} y_1]$
$[x_1, +\infty), x_1 \leq 0$	$(-\infty, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, +\infty)$	$(-\infty, x_1 \overset{\wedge}{/} y_2]$	$[x_1 \overset{\vee}{/} y_1, +\infty)$
$[x_1, +\infty), x_1 \geq 0$ $(-\infty, +\infty)$	$(-\infty, x_1 \overset{\wedge}{/} y_1]$ $(-\infty, +\infty)$	$[x_1 \overset{\vee}{/} y_2, +\infty)$ $(-\infty, +\infty)$	$(-\infty, 0]$ $(-\infty, +\infty)$	$[0, +\infty)$ $(-\infty, +\infty)$

Note: $-\infty$ is represented by **negativeInfinity** and $+\infty$ is represented by **positiveInfinity**.

Table 4.3 – Result of **intervalDivision** when *source*₂ contains zero

<i>source</i> ₁	<i>source</i> ₂				
	[0, 0]	$[y_1, y_2]$ $y_1 < y_2 = 0$	$[y_1, y_2]$ $y_1 < 0 < y_2$	$[y_1, y_2]$ $0 = y_1 < y_2$	$(-\infty, y_2]$ $y_2 = 0$
$[x_1, x_2], x_2 < 0$	\emptyset	$[x_2 \overset{\vee}{/} y_1, +\infty)$	$[x_2 \overset{\vee}{/} y_1, x_2 \overset{\wedge}{/} y_2]^\dagger$	$(-\infty, x_2 \overset{\wedge}{/} y_2]$	$[0, +\infty)$
$[x_1, x_2], x_1 \leq 0 \leq x_2$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, x_2], x_1 > 0$	\emptyset	$(-\infty, x_1 \overset{\wedge}{/} y_1]$	$[x_1 \overset{\vee}{/} y_2, x_1 \overset{\wedge}{/} y_1]^\dagger$	$[x_1 \overset{\vee}{/} y_2, +\infty)$	$(-\infty, 0]$
$(-\infty, x_2], x_2 < 0$	\emptyset	$[x_2 \overset{\vee}{/} y_1, +\infty)$	$[x_2 \overset{\vee}{/} y_1, x_2 \overset{\wedge}{/} y_2]^\dagger$	$(-\infty, x_2 \overset{\wedge}{/} y_2]$	$[0, +\infty)$
$(-\infty, x_2], x_2 > 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 < 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 > 0$	\emptyset	$(-\infty, x_1 \overset{\wedge}{/} y_1]$	$[x_1 \overset{\vee}{/} y_2, x_1 \overset{\wedge}{/} y_1]^\dagger$	$[x_1 \overset{\vee}{/} y_2, +\infty)$	$(-\infty, 0]$
$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$

Table 4.3 – Result of **intervalDivision** when *source*₂ contains zero (cont.)

<i>source</i> ₁	<i>source</i> ₂			
	$(-\infty, y_2]$ $y_2 > 0$	$[y_1, +\infty)$ $y_1 < 0$	$[y_1, +\infty)$ $y_1 = 0$	$(-\infty, +\infty)$
$[x_1, x_2], x_2 < 0$	$[0, x_2 \overset{\wedge}{/} y_2]^\dagger$	$[x_2 \overset{\vee}{/} y_1, 0]^\dagger$	$(-\infty, 0]$	$(-\infty, +\infty)$
$[x_1, x_2], x_1 \leq 0 \leq x_2$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, x_2], x_1 > 0$	$[x_1 \overset{\vee}{/} y_2, 0]^\dagger$	$[0, x_1 \overset{\wedge}{/} y_1]^\dagger$	$[0, +\infty)$	$(-\infty, +\infty)$
$(-\infty, x_2], x_2 < 0$	$[0, x_2 \overset{\wedge}{/} y_2]^\dagger$	$[x_2 \overset{\vee}{/} y_1, 0]^\dagger$	$(-\infty, 0]$	$(-\infty, +\infty)$
$(-\infty, x_2], x_2 > 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 < 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 > 0$	$[x_1 \overset{\vee}{/} y_2, 0]^\dagger$	$[0, x_1 \overset{\wedge}{/} y_1]^\dagger$	$[0, +\infty)$	$(-\infty, +\infty)$
$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$

Notes: \emptyset The empty interval is encoded as (+NaN, -NaN).

† $[r_1, r_2]$ with $-\infty < r_2 < r_1 < +\infty$ is a special encoding of the result $(-\infty, r_2] \cup [r_1, +\infty)$. Since the result of an interval operation is supposed to be a single interval, results that consist of the union of two disjoint intervals are represented by intervals in which the left bound is greater than the right bound.

$-\infty$ is represented by **negativeInfinity** and $+\infty$ is represented by **positiveInfinity**.

After an **intervalDivision** operation, the program is expected to check whether the result is a single interval or two disjoint intervals. If the result is two disjoint intervals, the calculation should proceed along two separate paths, one for each interval. Alternatively, the program can check before each division whether *y* contains zero, divide *y* into two intervals $[y_1, 0]$ and $[0, y_2]$, perform two **intervalDivision** operations, and proceed from there with two separate paths of computation.

NOTE

Forking the calculation when divide produces two intervals is the method by which, e.g., the interval Newton method finds all zeroes within a domain.

4.3 Interval comparison and lattice operations

An implementation that provides interval formats should provide the following comparison and lattice operations, for all supported non-storage interval floating-point formats with operands of the same radix. If the result of a lattice operation is the empty interval, it is encoded (+NaN,−NaN).

- *boolean* **intervalCompareEqual** (*source*₁, *source*₂)
if neither *x* nor *y* is the empty interval the operation **intervalCompareEqual** (*x*, *y*) computes the result $(x_1 = y_1 \wedge x_2 = y_2)$, where = is defined by the **compareEqual** operation in subclause 5.6.1 of IEEE P754. If either *x* or *y* is the empty interval the result is false.
- *boolean* **intervalCompareLessEqual** (*source*₁, *source*₂)
if neither *x* nor *y* is the empty interval the operation **intervalCompareLessEqual** (*x*, *y*) computes the result $(x_1 \leq y_1 \wedge x_2 \leq y_2)$, where \leq is defined by the **compareLessEqual** operation in subclause 5.6.1 of IEEE P754. If either *x* or *y* is the empty interval the result is false.
- *formatOf* **intervalGreatestLowerBound** (*source*₁, *source*₂)
the operation **intervalGreatestLowerBound** (*x*, *y*) computes the result $r := [\min(x_1, y_1), \min(x_2, y_2)]$.
- *formatOf* **intervalLeastUpperBound** (*source*₁, *source*₂)
the operation **intervalLeastUpperBound**(*x*, *y*) computes the result $r := [\max(x_1, y_1), \max(x_2, y_2)]$.
- *boolean* **intervalInclusion** (*source*₁, *source*₂)
if neither *x* nor *y* is the empty interval, the operation **intervalInclusion** (*x*, *y*) computes the result $(y_1 \leq x_1) \wedge (x_2 \leq y_2)$ where \leq is defined by the **compareLessEqual** operation in subclause 5.6.1 of IEEE P754. If *y* is the empty interval the result is false. Otherwise if *x* is the empty interval the result is true.
- *boolean* **intervalElementOf** (*source*₁, *source*₂)
the operation **intervalElementOf** (*x*, *y*) where *x* is a point and *y* is an interval computes the result **intervalInclusion** ([*x*, *x*], *y*).
- *formatOf* **intervalHull** (*source*₁, *source*₂)
if neither *x* nor *y* is the empty interval, the operation **intervalHull** (*x*, *y*) computes the result $r := [\min(x_1, y_1), \max(x_2, y_2)]$. If one of *x* or *y* is the empty interval the result is the other operand. If both *x* and *y* are the empty interval the result is the empty interval.
- *formatOf* **intervalIntersection** (*source*₁, *source*₂)
the operation **intervalIntersection** (*x*, *y*) computes $r := [\max(x_1, y_1), \min(x_2, y_2)]$. If $r_2 < r_1$, where $<$ is defined by the **compareLess** operation in subclause 5.6.1 of IEEE P754, the final result is the empty interval. Otherwise the final result is *r*.

- *boolean* **intervalCheckProper** (*source*)
the operation **intervalCheckProper** (x) computes the result $x_1 \leq x_2$, where \leq is defined by the **compareLessEqual** operation in subclause 5.6.1 of IEEE P754. This tests whether *source* is a proper interval or the representation of a disjoint interval, usually resulting from the operation **intervalDivision**. If x is the empty interval the result is false.
- *boolean* **intervalEmpty** (*source*)
the result of the operation **intervalEmpty** (x) is true if and only if x represents the empty interval.

NOTE

The advantage of an implementation providing comparison and lattice operations is that if it supports interval arithmetic operations with parallel circuits (for higher performance), these operations can be performed mostly or entirely by using the same parallel circuits. Because they do not require changing the rounding direction attribute, they can be performed by software using point operations without significant loss of performance.

5 Complete arithmetic

5.1 Complete formats

Complete formats are signed binary fixed-point formats such that multiplication of pairs of normal floating-point numbers in the formats specified by IEEE P754 and accumulation of up to b^v summands using operations having those destination formats have exact results. They are *complete* in that all possible information from normal floating-point operands of multiplication and accumulation operations is represented.

These formats are characterized by parameters m and d . The parameter m specifies the number of digits before the point, and d specifies the number of digits after the point. The product of two floating-point numbers with p digits has $2p$ digits. In order to represent the fractional part exactly, another $|2 \text{ emin}|$ digits are required after the point. In order to represent the integer part without overflow, 2 emax digits are required before the point. Each accumulation can result in an overflow of the integer part; therefore, another v digits before the point allows b^v accumulations of products of floating-point operands or sums or differences of complete operands to be computed without overflow.

Table 5.1 specifies the values of v , m and d for floating-point operands in the formats specified by IEEE P754. The digits in each format have the same radix as the digits in the associated operands.

Table 5.1 – Complete format parameters for floating-point operands

Parameter	Complete formats associated with:				
	binary32	binary64	binary128	decimal64	decimal128
v , overflow digits	≥ 65	≥ 65	≥ 65	≥ 20	≥ 20
m , digits before point = $2 \text{ emax} + v$	≥ 319	≥ 2111	≥ 32831	≥ 789	≥ 12308
d , digits after point = $2p + 2 \text{ emin} $	300	2150	16608	798	12354

Table 5.1 – Complete format parameters for floating-point operands (cont.)

$m + d$	≥ 619	≥ 4261	≥ 49439	≥ 1587	≥ 24663
k_e , recommended value [†]	640	4288	49536	5312	82304
$\lceil k_e/k \rceil$	20	67	387	83	643

Note: [†] k_e is the width in bits of the exact interchange format encoding.
See subclause 3.9 of IEEE P754.

Complete formats include a status field that has one of the values *exact*, *inexact*, *infinity* or *NaN*, represented by bit values 00, 01, 10 and 11, respectively. If the value is *infinity* the sign is determined as for floating-point results as specified in IEEE P754, but the remainder of the format is implementation defined. If the value is *NaN* the remainder of the format is as specified in subclause 8.4.

An implementation shall at least provide complete formats corresponding to binary64 if it provides binary floating point and decimal64 if it provides decimal floating point, and operations on complete format operands or that produce complete format results shall be provided. For complete formats associated with decimal floating-point formats specified in IEEE P754, m and d shall be multiples of three. Encodings for storage and arithmetic using these formats are implementation defined, but should be fixed width.

Language standards should define mechanisms supporting complete arithmetic for each radix supported for floating-point formats.

5.2 Complete interchange format encodings

Representations of binary complete data in binary interchange formats are encoded in $k_e \geq m+d+3$ bits in the following fields as shown in Figure 5.1:

- a) 2-bit status Q,
- b) 1-bit sign S,
- c) $m + x$ -bit integer part I, and
- d) d -bit fractional part F.

The value of x is implementation defined, but should be such that $x + m + d + 3$ is a multiple of eight. It is recommended that it be a multiple of the number of bits k in the corresponding binary interchange format encoding. The values of extra overflow bits that an implementation does not use for arithmetic results shall be zero.

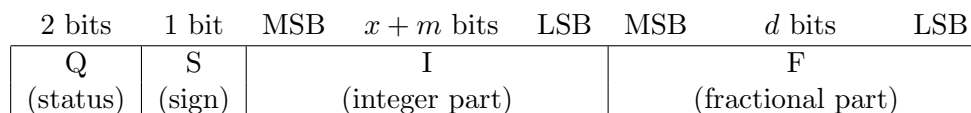


Figure 5.1—Binary interchange, binary complete format

Representations of decimal complete data in binary interchange formats are encoded in $k_e \geq 10(m+d)/3 + 3$ bits in the following fields as shown in Figure 5.2:

- a) 2-bit status Q,
- b) 1-bit sign S,
- c) $M + x$ -bit integer part I, where $M = 10 \times m/3$, and
- d) D bit fractional part F, where $D = 10 \times d/3$.

The value of x is implementation defined, but should be such that $x + M + D + 3$ is a multiple of eight. It is recommended that it be a multiple of the number of bits k in the corresponding decimal interchange format encoding. The values of extra overflow bits that an implementation does not use for arithmetic results shall be zero.

2 bits	1 bit	MSB	$x + M$ bits	LSB	MSB	D bits	LSB
Q	S	I			F		
(status)	(sign)	(integer part)			(fractional part)		

Figure 5.2—Binary interchange, decimal complete format

6 Attributes and rounding

Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the destination's format while signaling the inexact exception (see subclause 7.6 of IEEE P754), underflow, or overflow when appropriate. Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range. If the destination format is not a complete format, or is a complete format for which d is less for the result than for the operands, that result is then rounded according to one of the attributes, as specified in clause 4 of IEEE P754.

The only operations having a complete result to which rounding is applied are the **convert**, **exact-Addition**, and **exactSubtraction** operations, when the operands are complete and the value of d is less for the result than for the operands.

Implementations supporting complete formats shall provide rounding-direction attributes for complete that are separate from those for floating-point. Conversions to a complete format result shall use the rounding-direction attribute for complete formats.

7 Operations of complete arithmetic

7.1 General

Operations of complete arithmetic produce a fixed-point result in complete format.

All conforming implementations of this International Standard shall provide the operations listed in this clause for all supported complete formats available for arithmetic. Each of the computational operations that return a numeric result specified by this International Standard shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range. If the destination format is not a complete format, or is a complete format for which d is less for the result than for the operands, that result is then rounded according to one of the attributes, as specified in clauses 4 and 7 of IEEE P754. Clause 6 of IEEE P754 augments

the following specifications to cover ± 0 , $\pm\infty$, and NaN; clause 7 of IEEE P754 describes default exception handling. In this International Standard, operations are written as named functions; in a specific programming environment they might be represented by operators, or by families of format-specific functions, or by generic functions whose names might differ from those in this International Standard.

In addition to the indicators of operand and result formats specified in subclause 5.1 of IEEE P754, this International Standard specifies

- *exactFormatOf* indicates that the name of the operation specifies a complete destination format.

7.2 Operations in addition to those specified in IEEE P754

7.2.1 *exactFormatOf* arithmetic operations

Implementations shall provide the following *exactFormatOf* operations for destinations of all supported complete formats available for arithmetic, and for each destination format, for operands of all supported floating-point and complete formats available for arithmetic with the same radix as the destination format. These operations never propagate non-canonical results.

- *exactFormatOf*-**exactAddition**(*source1*, *source2*)
The operation **exactAddition**(x , y) computes $x + y$.
- *exactFormatOf*-**exactSubtraction**(*source1*, *source2*)
The operation **exactSubtraction**(x , y) computes $x - y$.

Implementations shall provide the following *exactFormatOf* operation for destinations of all supported complete formats available for arithmetic. For each destination format, the operation shall be provided for operands of all supported floating-point formats available for arithmetic with the same radix as the destination format, and for operand *source3* of all complete formats available for arithmetic with the same radix as the destination format. This operation never propagates non-canonical results.

- *exactFormatOf*-**exactFusedMultiplyAdd**(*source1*, *source2*, *source3*)
The operation **exactFusedMultiplyAdd**(x , y , z) computes $(x \times y) + z$.

7.2.2 Conversion operations for all formats

Implementations shall provide the following *formatOf* conversion operation from all supported floating-point or complete formats to all supported complete formats, and from all supported complete formats to all supported floating-point or complete formats, including storage formats, as well as conversions to and from decimal character sequences. Some format conversion operations produce results in a different radix from the operands.

- *formatOf*-**convert**(*source*)

If the conversion is from a complete format to a floating-point format that has a different radix from the source, or the same radix as the source but a narrower precision, the result shall be rounded as specified in clause 4 of IEEE P754. Conversion from a complete format to a complete format with a smaller value of d shall round as specified in clause 4 of IEEE P754. Conversion from a floating-point format to a complete format is always exact. Conversion from a complete format to a complete format with a larger value of d is always exact. Conversion from a complete format to a complete format with a smaller value of m might cause overflow (see subclause 7.4 of IEEE P754) to signal.

Implementations shall provide the following conversion operation from all supported floating-point and integer formats to all supported complete formats.

- *exactFormatOf-convert*(*source*)

7.2.3 Conversion operations for binary formats

Implementations shall provide the following *exactFormatOf* conversion operations to and from all supported binary complete formats, including storage formats; these operations never propagate non-canonical results.

- *exactFormatOf-convertFromHexCharacter*(*hexCharacterSequence*)

See subclause 5.12 of IEEE P754 for details.

- *hexCharacterSequence-convertToHexCharacter*(*source*, *conversionSpecification*)

See subclause 5.12 of IEEE P754 for details. The *conversionSpecification* specifies the precision and formatting of the *hexCharacterSequence* result.

7.3 Sign operations

Implementations shall provide the following homogeneous sign operations for all supported complete formats available for arithmetic; they only affect the sign. The operations treat complete format numbers and NaNs alike, and signal no exception. They may propagate non-canonical encodings.

- *sourceFormat-copy*(*source*)
sourceFormat-negate(*source*)
sourceFormat-abs(*source*)

copy(x) copies a complete operand x to a destination in the same format, with no change to the sign.

negate(x) copies a complete operand x to a destination in the same format, reversing the sign. $0 - x$ is not the same as $-x$ or **negate**(x).

abs(x) copies a complete operand x to a destination in the same format, changing the sign to positive.

— *sourceFormat-copySign*(*source1*, *source2*)

copySign(*x*, *y*) copies a complete operand *x* to a destination in the same format as *x*, but with the sign of *y*.

7.4 Comparisons

Implementations shall provide the following comparison operations, for operands in all supported complete formats available for arithmetic. These operations may signal exceptions.

— *boolean compareEqual*(*source1*, *source2*)
boolean compareNotEqual(*source1*, *source2*)
boolean compareGreater(*source1*, *source2*)
boolean compareGreaterEqual(*source1*, *source2*)
boolean compareLess(*source1*, *source2*)
boolean compareLessEqual(*source1*, *source2*)
boolean compareSignalingNotGreater(*source1*, *source2*)
boolean compareSignalingLessUnordered(*source1*, *source2*)
boolean compareSignalingNotLess(*source1*, *source2*)
boolean compareSignalingGreaterUnordered(*source1*, *source2*)
boolean compareQuietGreater(*source1*, *source2*)
boolean compareQuietGreaterEqual(*source1*, *source2*)
boolean compareQuietLess(*source1*, *source2*)
boolean compareQuietLessEqual(*source1*, *source2*)
boolean compareUnordered(*source1*, *source2*)
boolean compareQuietNotGreater(*source1*, *source2*)
boolean compareQuietLessUnordered(*source1*, *source2*)
boolean compareQuietNotLess(*source1*, *source2*)
boolean compareQuietGreaterUnordered(*source1*, *source2*)
boolean compareOrdered(*source1*, *source2*)

For every supported complete format available for arithmetic, it shall be possible to compare one complete datum to another complete datum, even if the data have different complete formats.

Comparison predicates for complete format data act as described in subclause 5.11 of IEEE P754.

7.5 General operation

Implementations shall provide the following non-computational operation for all supported complete formats available for arithmetic. It is never exceptional, even for signaling NaNs.

— *boolean isSigned*(*source*)

isSigned(*x*) is true if and only if *x* has negative sign. **isSigned** applies to zeros and NaNs as well.

8 Infinity, NaNs, and sign bit

8.1 General

For interval arithmetic and interval formats, infinity, NaNs and sign bits are treated as specified in IEEE P754. For complete arithmetic and complete formats, infinity, NaNs and sign bits are treated as specified in this clause.

8.2 Infinity arithmetic

As is the case for infinity in floating-point arithmetic, as specified in subclause 6.1 of IEEE P754, the behavior of infinity in complete arithmetic is derived from the limiting cases of real arithmetic with operands of arbitrarily large magnitude, when such a limit exists. Infinities shall be interpreted in the affine sense. Operations on infinite operands in complete formats act as described for operands in floating-point formats, as specified in subclause 6.1 of IEEE P754.

8.3 Operations with NaNs

Two different kinds of NaN, signaling and quiet, shall be supported in all complete arithmetic operations. Signaling NaNs afford representations for uninitialized variables and arithmetic-like enhancements (such as complex-affine infinities or extremely wide range) that are not the subject of this International Standard. Quiet NaNs should, by means left to the implementer's discretion, afford retrospective diagnostic information inherited from invalid or unavailable data and results. To facilitate propagation of diagnostic information contained in NaNs, as much of that information as possible should be preserved in NaN results of computational operations.

Under default exception handling, any operation signaling an invalid exception and for which a complete-format result is to be delivered shall deliver a quiet NaN.

Signaling NaNs shall be reserved operands that, under default exception handling, signal the invalid operation exception (see subclause 7.1 of IEEE P754) for every general-computational and signaling-computational operation except for the conversions described in subclause 5.12 of IEEE P754 (for non-default treatment see clause 8 of IEEE P754).

Every `exactFormatOf` (7.2.1), general conversion (7.2.2), binary conversion (7.2.3) and sign operation (7.3) involving one or more input NaNs, none of them signaling, shall signal no exception, except `exactFusedMultiplyAdd` might signal the invalid operation exception (see subclause 7.2 of IEEE P754). For an operation with quiet NaN inputs, other than maximum and minimum operations, if a complete-format result is to be delivered the result shall be a quiet NaN, which should be one of the input NaNs. Note that format conversions, including conversions between supported formats and external representations as character sequences, might be unable to deliver the same NaN. Quiet NaNs signal exceptions on some operations that do not deliver a complete-format result; these operations, namely comparison and conversion to a format that has no NaNs, are discussed in subclause 7.4 of this International Standard, and subclauses 5.8 and 7.2 of IEEE P754.

8.4 NaN encodings in complete formats

This subclause further specifies the encodings of NaNs as bit strings when they are the results of operations. When encoded, all NaNs have a sign bit and a pattern of other bits necessary to identify the encoding as a NaN and which determines whether they are quiet or signaling NaNs. The remaining bits, which are in the trailing field, encode the payload, which might be diagnostic information.

All complete-format NaN bit strings shall have a Q field (5.2) value of 11. A quiet NaN bit string shall be encoded with the most-significant bit of the integer part (I field) being 1. A signaling NaN bit string shall be encoded with the most-significant bit of the integer part being 0. The payload is the fractional part (F field).

8.5 Invalid operations

In addition to the invalid operations enumerated in subclause 7.2 of IEEE P754, for operations producing results in complete formats, the default result of an invalid operation shall be a quiet NaN that should provide some diagnostic information (see subclause 6.2 of IEEE P754). The following operations are invalid exception operations:

- a) `exactFusedMultiplyAdd`: `exactFusedMultiplyAdd(0,∞,c)` or `exactFusedMultiplyAdd(∞,0,c)` unless c is a quiet NaN; if c is a quiet NaN then it is implementation defined whether the invalid operation exception is signaled
- b) `exactFusedMultiplyAdd`: magnitude subtraction of infinities.

In addition to the invalid operations enumerated in subclause 7.2 of IEEE P754, for operations producing no result in a complete format, the invalid operation exceptions are:

- c) conversion of a complete-format number to an integer or complete format, when the source is NaN, infinity, or a value which would convert to a value outside the range of the result format under the applicable rounding attribute.

8.6 Overflow

The overflow exception shall be signaled if and only if

- a) the destination format is a floating-point format and the destination format's largest finite number is exceeded in magnitude by what would have been the rounded floating-point result (see clause 4 of IEEE P754) were the exponent range unbounded, or
- b) the destination format is a complete format and the destination format's largest finite number is exceeded in magnitude by what would have been the rounded exact result (see clause 6) were the value of the parameter m unbounded.

The default result shall be determined by the rounding-direction attribute and the sign of the intermediate result as specified in subclause 7.4 of IEEE P754.

8.7 Underflow

The underflow exception shall not signal if the destination format is complete.

Annex A

(informative)

Bibliography

- [1] Reinhard Kirchner and Ulrich W. Kulisch, *Hardware support for interval arithmetic*, **Reliable Computing** **12**, 3 (2006) 225–237.
- [2] Ulrich W. Kulisch, **Computer Arithmetic and Validity — Theory, Implementation and Applications**, de Gruyter, Berlin (2008) ISBN 978-3-11-020318-9.