

To: Floating-point working group, IEEE Computer Society Microprocessor Standards Committee
From: IFIP Working Group 2.5 – Numerical Software
Subject: Interval arithmetic and complete arithmetic support in IEEE standard P754
Date: 17 December 2007
Reference: Draft Standard for Floating-Point Arithmetic P754, Draft 1.5.0

1 Introduction

This paper proposes editorial changes to draft 1.5.0 of the IEEE standard for floating-point arithmetic, to specify optional support for interval arithmetic and complete arithmetic. The intent is to specify how these arithmetics should be implemented if a system chooses to do so, not to require that they be provided.

2 Editorial changes for interval arithmetic

Positions within draft 1.5.0 are denoted here in the form page:line.

14:7 Replace the introduction to the list:

“Formats defined by this standard are classified by two independent criteria. The first is whether they are interchange or non-interchange formats. The second is whether they are point or interval formats.”

14:23+ Insert a new paragraph after the itemized list:

“Point formats consist of a single element. Interval formats consist of two elements, having the same encoding. It is recommended but not required that languages provide interval formats and the operations specified on intervals in subclause 9.5. 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 9.5.2, and should provide the interval comparison and lattice operations specified in subclause 9.5.3. Unless otherwise specified, definitions in this standard apply to point formats.”

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

“Point formats consist of a single element. Interval formats consist of two elements, having the same encoding. It is recommended but not required that languages provide interval formats and the operations specified on intervals in Clause 9.5. 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 9.5.2 for all interval formats it provides, and should provide the interval comparison and lattice operations specified in subclause 9.5.3. Unless otherwise specified, definitions in this standard apply to point formats.”]

56:44+ Insert a new subclause:

“9.5 Interval operations

“9.5.1 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.”

“9.5.2 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. 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, 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.

“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 11a.

Table 11a – Result of **intervalMultiplication**

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

Table 11a – Result of **intervalMultiplication** (cont.)

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

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

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 11b if the interval *source*₂ does not contain zero, and Table 11c if the interval *source*₂ contains zero.

Table 11b – Result of **intervalDivision** when *source*₂ does not contain zero

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

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

Table 11c – 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	$[x_2 / y_1, +\infty)$	$[x_2 / y_1, x_2 / y_2]^\dagger$	$(-\infty, x_2 / y_2]$	$[0, +\infty)$
$[x_1, x_2], x_1 \leq 0 \leq x_2$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, x_2], x_1 > 0$	\emptyset	$(-\infty, x_1 / y_1]$	$[x_1 / y_2, x_1 / y_1]^\dagger$	$[x_1 / y_2, +\infty)$	$(-\infty, 0]$
$(-\infty, x_2], x_2 < 0$	\emptyset	$[x_2 / y_1, +\infty)$	$[x_2 / y_1, x_2 / y_2]^\dagger$	$(-\infty, x_2 / y_2]$	$[0, +\infty)$
$(-\infty, x_2], x_2 > 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 < 0$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$
$[x_1, +\infty), x_1 > 0$	\emptyset	$(-\infty, x_1 / y_1]$	$[x_1 / y_2, x_1 / y_1]^\dagger$	$[x_1 / y_2, +\infty)$	$(-\infty, 0]$
$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$	$(-\infty, +\infty)$

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

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

\dagger $[r_1, r_2]$ 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 to committee: 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. If IEEE permits non-normative notes in its standards, you may wish to insert such a note to this effect at this point.]

[Note to committee: Subclause 9.5.3 could be omitted without great loss of functionality. Its advantage is that if an implementation supports interval arithmetic operations with parallel circuits (for higher performance), the comparison and lattice 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.]

“9.5.3 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. 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. 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. If x is the empty interval the result is true. If y is the empty interval the result is false.
- *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, 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. 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.”

61:31+ Add references to the bibliography:

“R. Kirchner and U.W. Kulisch, *Hardware support for interval arithmetic*, **Reliable Computing** **12**, 3 (2006) 225-237.”

“U.W. Kulisch, **Computer Arithmetic and Validity — Theory, Implementation and Applications**, de Gruyter (2008).”

3 Editorial changes for complete arithmetic

14:21 After “extendable precision formats” insert “, and complete formats,”

22:11+ Insert two new subclauses:

“3.8 Complete formats

“Complete formats are signed fixed-point formats such that multiplication of pairs of normal floating-point numbers 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, $2\text{ }emax$ 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 7a specifies the values of v , m and d for floating-point operands. The digits in each format have the same radix as the digits in the associated operands.

Table 7a – 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
$m + d$	≥ 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.

“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 but the remainder of the format is implementation defined. If the value is *NaN* the remainder of the format is implementation defined.

“This standard does not require an implementation to provide any complete format. If it provides complete arithmetic, it 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, 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 supported radix.

“3.9 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 3.3:

- 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.

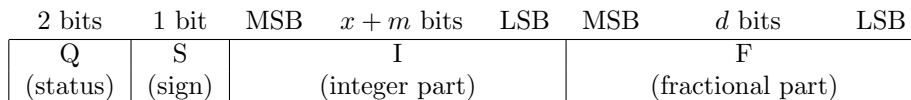


Figure 3.3—Binary interchange, binary complete format

“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 3.4:

- 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.

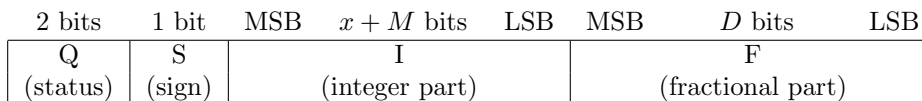


Figure 3.4—Binary interchange, decimal complete format

24:1 Replace “, and then rounded that result” by “. 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”.

24:2 Add a sentence at the end of the paragraph:

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

24:10+ (the first line 10, immediately before **4.3.1 Rounding-direction attributes to nearest**) insert a new paragraph:

“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.”

25:6 Replace “, and then rounded that intermediate result” by “. 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 intermediate result is then rounded”.

25:12 Replace “four” by “five”.

25:13+ Insert a new first item in the list:

“— complete operations produce a fixed-point result in complete format.”

25:22 Replace “three” by “four”.

26:12+ Insert a new paragraph:

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

30:12+ Insert a new paragraph and list:

“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 $(x \times y) + z$.”

30:15 Insert “and complete” before the first “formats”.

30:19-21 Replace the paragraph:

“If the conversion is to a floating-point format that has a different radix from the source, or the same radix as the source but a narrower precision, or from a complete format, the result shall be rounded as specified in clause 4. Conversion from a complete format to a complete format with a smaller value of *d* shall round as specified in clause 4. Conversion from a floating-point format to a floating-point format with the same radix but wider precision and range, or 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) to signal.”

30:19 Replace “or” by a comma, insert “, or from a complete format to a floating-point format” after “same radix”.

30:21 Insert “, or from a floating-point format to a complete format,” after “range”.

30:30+ Insert a new paragraph and list

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

— *exactFormatOf*-**convert**(*source*)

30:33 Insert “and binary complete” before “formats”.

31:4 Insert “and complete” before “formats”.

32:4 Insert “and complete” before “operands”.

33:42+ Insert a new paragraph:

“The **isSigned** operation shall be provided for all supported complete formats.”

38:4+ Insert a new paragraph:

“For every supported complete format available for arithmetic, it shall be possible to compare one complete datum to another in that format. Additionally, complete data represented in different formats shall be comparable as long as the operands’ formats have the same radix.”

44:3 Insert “or complete” after “floating-point”.

44:16 Insert “or complete” after “floating-point”.

44:38 Insert “floating-point” before “formats”.

44:36 Insert “or complete” before “result”.

45:5 Insert “floating-point” before “formats”.

45:10+ Insert a new subclause:

“6.2.2a NaN encodings in complete formats

“Other than the value of the status field that indicates a value in complete format is a NaN, the encoding of NaNs in complete formats is implementation defined.”

47:10 Replace the first line of the item:

“c) fusedMultiplyAdd or exactFusedMultiplyAdd: fusedMultiplyAdd $(0, \infty, c)$, exactFusedMultiplyAdd $(0, \infty, c)$, fusedMultiplyAdd $(\infty, 0, c)$, or exactFusedMultiplyAdd $(\infty, 0, c)$ unless c is a quiet”

47:13 Replace “addition or subtraction or fusedMultiplyAdd” by “addition, subtraction, fusedMultiplyAdd, or exactFusedMultiplyAdd”

47:21 Insert “or complete” after “integer”.

47:20 Insert “or complete” before “format”.

47:34-37 Replace the paragraph:

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) 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 4) 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 follows:

48:5+ Insert a new paragraph:

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