



Jean-Michel Muller • Nicolas Brunie  
Florent de Dinechin • Claude-Pierre Jeannerod  
Mioara Joldes • Vincent Lefèvre  
Guillaume Melquiond • Nathalie Revol  
Serge Torres

# Handbook of Floating-Point Arithmetic

Second Edition

Jean-Michel Muller  
CNRS - LIP  
Lyon, France

Nicolas Brunie  
Kalray  
Grenoble, France

Florent de Dinechin  
INSA-Lyon - CITI  
Villeurbanne, France

Claude-Pierre Jeannerod  
Inria - LIP  
Lyon, France

Mioara Joldes  
CNRS - LAAS  
Toulouse, France

Vincent Lefèvre  
Inria - LIP  
Lyon, France

Guillaume Melquiond  
Inria - LRI  
Orsay, France

Nathalie Revol  
Inria - LIP  
Lyon, France

Serge Torres  
ENS-Lyon - LIP  
Lyon, France

ISBN 978-3-319-76525-9      ISBN 978-3-319-76526-6 (eBook)  
<https://doi.org/10.1007/978-3-319-76526-6>

Library of Congress Control Number: 2018935254

Mathematics Subject Classification: 65Y99, 68N30

© Springer International Publishing AG, part of Springer Nature 2010, 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This book is published under the imprint Birkhäuser, [www.birkhauser-science.com](http://www.birkhauser-science.com) by the registered company Springer International Publishing AG part of Springer Nature.  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Contents

<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xix</b>
<b>Preface</b>	<b>xxiii</b>
<b>Part I Introduction, Basic Definitions, and Standards</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Some History . . . . .	4
1.2 Desirable Properties . . . . .	7
1.3 Some Strange Behaviors . . . . .	8
1.3.1 Some famous bugs . . . . .	8
1.3.2 Difficult problems . . . . .	9
<b>2 Definitions and Basic Notions</b>	<b>15</b>
2.1 Floating-Point Numbers . . . . .	15
2.1.1 Main definitions . . . . .	15
2.1.2 Normalized representations, normal and subnormal numbers . . . . .	17
2.1.3 A note on underflow . . . . .	19
2.1.4 Special floating-point data . . . . .	21
2.2 Rounding . . . . .	22
2.2.1 Rounding functions . . . . .	22
2.2.2 Useful properties . . . . .	24
2.3 Tools for Manipulating Floating-Point Errors . . . . .	25
2.3.1 Relative error due to rounding . . . . .	25
2.3.2 The ulp function . . . . .	29
2.3.3 Link between errors in ulps and relative errors . . . . .	34
2.3.4 An example: iterated products . . . . .	35

2.4	The Fused Multiply-Add (FMA) Instruction . . . . .	37
2.5	Exceptions . . . . .	37
2.6	Lost and Preserved Properties of Real Arithmetic . . . . .	40
2.7	Note on the Choice of the Radix . . . . .	41
2.7.1	Representation errors . . . . .	41
2.7.2	A case for radix 10 . . . . .	43
2.8	Reproducibility . . . . .	44
<b>3</b>	<b>Floating-Point Formats and Environment</b>	<b>47</b>
3.1	The IEEE 754-2008 Standard . . . . .	48
3.1.1	Formats . . . . .	48
3.1.2	Attributes and rounding . . . . .	66
3.1.3	Operations specified by the standard . . . . .	70
3.1.4	Comparisons . . . . .	72
3.1.5	Conversions to/from string representations . . . . .	73
3.1.6	Default exception handling . . . . .	74
3.1.7	Special values . . . . .	77
3.1.8	Recommended functions . . . . .	79
3.2	On the Possible Hidden Use of a Higher Internal Precision . . . . .	79
3.3	Revision of the IEEE 754-2008 Standard . . . . .	82
3.4	Floating-Point Hardware in Current Processors . . . . .	83
3.4.1	The common hardware denominator . . . . .	83
3.4.2	Fused multiply-add . . . . .	84
3.4.3	Extended precision and 128-bit formats . . . . .	85
3.4.4	Rounding and precision control . . . . .	85
3.4.5	SIMD instructions . . . . .	86
3.4.6	Binary16 (half-precision) support . . . . .	87
3.4.7	Decimal arithmetic . . . . .	87
3.4.8	The legacy x87 processor . . . . .	88
3.5	Floating-Point Hardware in Recent Graphics Processing Units . . . . .	89
3.6	IEEE Support in Programming Languages . . . . .	90
3.7	Checking the Environment . . . . .	91
3.7.1	MACHAR . . . . .	91
3.7.2	Paranoia . . . . .	92
3.7.3	UCBTest . . . . .	92
3.7.4	TestFloat . . . . .	92
3.7.5	Miscellaneous . . . . .	93

**Part II Cleverly Using Floating-Point Arithmetic 95**

**4 Basic Properties and Algorithms 97**

- 4.1 Testing the Computational Environment . . . . . 97
  - 4.1.1 Computing the radix . . . . . 97
  - 4.1.2 Computing the precision . . . . . 99
- 4.2 Exact Operations . . . . . 100
  - 4.2.1 Exact addition . . . . . 100
  - 4.2.2 Exact multiplications and divisions . . . . . 103
- 4.3 Accurate Computation of the Sum of Two Numbers . . . . . 103
  - 4.3.1 The Fast2Sum algorithm . . . . . 104
  - 4.3.2 The 2Sum algorithm . . . . . 107
  - 4.3.3 If we do not use rounding to nearest . . . . . 109
- 4.4 Accurate Computation of the Product of Two Numbers . . . . . 111
  - 4.4.1 The 2MultFMA Algorithm . . . . . 112
  - 4.4.2 If no FMA instruction is available: Veltkamp splitting and Dekker product . . . . . 113
- 4.5 Computation of Residuals of Division and Square Root with an FMA . . . . . 120
- 4.6 Another splitting technique: splitting around a power of 2 . . . . . 123
- 4.7 Newton–Raphson-Based Division with an FMA . . . . . 124
  - 4.7.1 Variants of the Newton–Raphson iteration . . . . . 124
  - 4.7.2 Using the Newton–Raphson iteration for correctly rounded division with an FMA . . . . . 129
  - 4.7.3 Possible double roundings in division algorithms . . . . . 136
- 4.8 Newton–Raphson-Based Square Root with an FMA . . . . . 138
  - 4.8.1 The basic iterations . . . . . 138
  - 4.8.2 Using the Newton–Raphson iteration for correctly rounded square roots . . . . . 138
- 4.9 Radix Conversion . . . . . 142
  - 4.9.1 Conditions on the formats . . . . . 142
  - 4.9.2 Conversion algorithms . . . . . 146
- 4.10 Conversion Between Integers and Floating-Point Numbers . . . . . 153
  - 4.10.1 From 32-bit integers to floating-point numbers . . . . . 153
  - 4.10.2 From 64-bit integers to floating-point numbers . . . . . 154
  - 4.10.3 From floating-point numbers to integers . . . . . 155
- 4.11 Multiplication by an Arbitrary-Precision Constant with an FMA 156
- 4.12 Evaluation of the Error of an FMA . . . . . 160

<b>5</b>	<b>Enhanced Floating-Point Sums, Dot Products, and Polynomial Values</b>	<b>163</b>
5.1	Preliminaries . . . . .	164
5.1.1	Floating-point arithmetic models . . . . .	165
5.1.2	Notation for error analysis and classical error estimates . . . . .	166
5.1.3	Some refined error estimates . . . . .	169
5.1.4	Properties for deriving validated running error bounds . . . . .	174
5.2	Computing Validated Running Error Bounds . . . . .	175
5.3	Computing Sums More Accurately . . . . .	177
5.3.1	Reordering the operands, and a bit more . . . . .	177
5.3.2	Compensated sums . . . . .	178
5.3.3	Summation algorithms that somehow imitate a fixed-point arithmetic . . . . .	184
5.3.4	On the sum of three floating-point numbers . . . . .	187
5.4	Compensated Dot Products . . . . .	189
5.5	Compensated Polynomial Evaluation . . . . .	190
<b>6</b>	<b>Languages and Compilers</b>	<b>193</b>
6.1	A Play with Many Actors . . . . .	193
6.1.1	Floating-point evaluation in programming languages . . . . .	194
6.1.2	Processors, compilers, and operating systems . . . . .	196
6.1.3	Standardization processes . . . . .	197
6.1.4	In the hands of the programmer . . . . .	199
6.2	Floating Point in the C Language . . . . .	200
6.2.1	Standard C11 headers and IEEE 754-1985 support . . . . .	201
6.2.2	Types . . . . .	202
6.2.3	Expression evaluation . . . . .	204
6.2.4	Code transformations . . . . .	209
6.2.5	Enabling unsafe optimizations . . . . .	210
6.2.6	Summary: a few horror stories . . . . .	211
6.2.7	The CompCert C compiler . . . . .	214
6.3	Floating-Point Arithmetic in the C++ Language . . . . .	215
6.3.1	Semantics . . . . .	215
6.3.2	Numeric limits . . . . .	215
6.3.3	Overloaded functions . . . . .	217
6.4	FORTRAN Floating Point in a Nutshell . . . . .	218
6.4.1	Philosophy . . . . .	218
6.4.2	IEEE 754 support in FORTRAN . . . . .	220
6.5	Java Floating Point in a Nutshell . . . . .	222
6.5.1	Philosophy . . . . .	222
6.5.2	Types and classes . . . . .	222

6.5.3	Infinities, NaNs, and signed zeros . . . . .	224
6.5.4	Missing features . . . . .	225
6.5.5	Reproducibility . . . . .	226
6.5.6	The BigDecimal package . . . . .	227
6.6	Conclusion . . . . .	229

**Part III Implementing Floating-Point Operators 231**

<b>7</b>	<b>Algorithms for the Basic Operations</b>	<b>233</b>
7.1	Overview of Basic Operation Implementation . . . . .	233
7.2	Implementing IEEE 754-2008 Rounding . . . . .	235
7.2.1	Rounding a nonzero finite value with unbounded exponent range . . . . .	235
7.2.2	Overflow . . . . .	238
7.2.3	Underflow and subnormal results . . . . .	239
7.2.4	The inexact exception . . . . .	239
7.2.5	Rounding for actual operations . . . . .	240
7.3	Floating-Point Addition and Subtraction . . . . .	240
7.3.1	Decimal addition . . . . .	244
7.3.2	Decimal addition using binary encoding . . . . .	245
7.3.3	Subnormal inputs and outputs in binary addition . . . . .	245
7.4	Floating-Point Multiplication . . . . .	245
7.4.1	Normal case . . . . .	246
7.4.2	Handling subnormal numbers in binary multiplication . . . . .	247
7.4.3	Decimal specifics . . . . .	248
7.5	Floating-Point Fused Multiply-Add . . . . .	248
7.5.1	Case analysis for normal inputs . . . . .	249
7.5.2	Handling subnormal inputs . . . . .	252
7.5.3	Handling decimal cohorts . . . . .	253
7.5.4	Overview of a binary FMA implementation . . . . .	254
7.6	Floating-Point Division . . . . .	256
7.6.1	Overview and special cases . . . . .	256
7.6.2	Computing the significand quotient . . . . .	257
7.6.3	Managing subnormal numbers . . . . .	258
7.6.4	The inexact exception . . . . .	259
7.6.5	Decimal specifics . . . . .	259
7.7	Floating-Point Square Root . . . . .	259
7.7.1	Overview and special cases . . . . .	259
7.7.2	Computing the significand square root . . . . .	260
7.7.3	Managing subnormal numbers . . . . .	260
7.7.4	The inexact exception . . . . .	261
7.7.5	Decimal specifics . . . . .	261



7.8	Nonhomogeneous Operators . . . . .	261
7.8.1	A software algorithm around double rounding . . . . .	262
7.8.2	The mixed-precision fused multiply-and-add . . . . .	264
7.8.3	Motivation . . . . .	265
7.8.4	Implementation issues . . . . .	265
<b>8</b>	<b>Hardware Implementation of Floating-Point Arithmetic</b>	<b>267</b>
8.1	Introduction and Context . . . . .	267
8.1.1	Processor internal formats . . . . .	267
8.1.2	Hardware handling of subnormal numbers . . . . .	268
8.1.3	Full-custom VLSI versus reconfigurable circuits (FPGAs) . . . . .	269
8.1.4	Hardware decimal arithmetic . . . . .	270
8.1.5	Pipelining . . . . .	271
8.2	The Primitives and Their Cost . . . . .	272
8.2.1	Integer adders . . . . .	272
8.2.2	Digit-by-integer multiplication in hardware . . . . .	278
8.2.3	Using nonstandard representations of numbers . . . . .	278
8.2.4	Binary integer multiplication . . . . .	280
8.2.5	Decimal integer multiplication . . . . .	280
8.2.6	Shifters . . . . .	282
8.2.7	Leading-zero counters . . . . .	283
8.2.8	Tables and table-based methods for fixed-point function approximation . . . . .	285
8.3	Binary Floating-Point Addition . . . . .	287
8.3.1	Overview . . . . .	287
8.3.2	A first dual-path architecture . . . . .	288
8.3.3	Leading-zero anticipation . . . . .	290
8.3.4	Probing further on floating-point adders . . . . .	294
8.4	Binary Floating-Point Multiplication . . . . .	295
8.4.1	Basic architecture . . . . .	295
8.4.2	FPGA implementation . . . . .	296
8.4.3	VLSI implementation optimized for delay . . . . .	297
8.4.4	Managing subnormals . . . . .	300
8.5	Binary Fused Multiply-Add . . . . .	301
8.5.1	Classic architecture . . . . .	301
8.5.2	To probe further . . . . .	302
8.6	Division and Square Root . . . . .	304
8.6.1	Digit-recurrence division . . . . .	305
8.6.2	Decimal division . . . . .	308
8.7	Beyond the Classical Floating-Point Unit . . . . .	309
8.7.1	More fused operators . . . . .	309
8.7.2	Exact accumulation and dot product . . . . .	309
8.7.3	Hardware-accelerated compensated algorithms . . . . .	311

8.8	Floating-Point for FPGAs . . . . .	312
8.8.1	Optimization in context of standard operators . . . . .	312
8.8.2	Operations with a constant operand . . . . .	314
8.8.3	Computing large floating-point sums . . . . .	315
8.8.4	Block floating point . . . . .	319
8.8.5	Algebraic operators . . . . .	319
8.8.6	Elementary and compound functions . . . . .	320
8.9	Probing Further . . . . .	320
<b>9</b>	<b>Software Implementation of Floating-Point Arithmetic</b>	<b>321</b>
9.1	Implementation Context . . . . .	322
9.1.1	Standard encoding of binary floating-point data . . . . .	322
9.1.2	Available integer operators . . . . .	323
9.1.3	First examples . . . . .	326
9.1.4	Design choices and optimizations . . . . .	328
9.2	Binary Floating-Point Addition . . . . .	329
9.2.1	Handling special values . . . . .	330
9.2.2	Computing the sign of the result . . . . .	332
9.2.3	Swapping the operands and computing the alignment shift . . . . .	333
9.2.4	Getting the correctly rounded result . . . . .	335
9.3	Binary Floating-Point Multiplication . . . . .	341
9.3.1	Handling special values . . . . .	341
9.3.2	Sign and exponent computation . . . . .	343
9.3.3	Overflow detection . . . . .	345
9.3.4	Getting the correctly rounded result . . . . .	346
9.4	Binary Floating-Point Division . . . . .	349
9.4.1	Handling special values . . . . .	350
9.4.2	Sign and exponent computation . . . . .	351
9.4.3	Overflow detection . . . . .	354
9.4.4	Getting the correctly rounded result . . . . .	355
9.5	Binary Floating-Point Square Root . . . . .	362
9.5.1	Handling special values . . . . .	362
9.5.2	Exponent computation . . . . .	363
9.5.3	Getting the correctly rounded result . . . . .	365
9.6	Custom Operators . . . . .	372
<b>10</b>	<b>Evaluating Floating-Point Elementary Functions</b>	<b>375</b>
10.1	Introduction . . . . .	375
10.1.1	Which accuracy? . . . . .	375
10.1.2	The various steps of function evaluation . . . . .	376
10.2	Range Reduction . . . . .	379
10.2.1	Basic range reduction algorithms . . . . .	379
10.2.2	Bounding the relative error of range reduction . . . . .	382

10.2.3	More sophisticated range reduction algorithms . . . . .	383
10.2.4	Examples . . . . .	386
10.3	Polynomial Approximations . . . . .	389
10.3.1	$L^2$ case . . . . .	390
10.3.2	$L^\infty$ , or minimax, case . . . . .	391
10.3.3	“Truncated” approximations . . . . .	394
10.3.4	In practice: using the Sollya tool to compute constrained approximations and certified error bounds . . . . .	394
10.4	Evaluating Polynomials . . . . .	396
10.4.1	Evaluation strategies . . . . .	396
10.4.2	Evaluation error . . . . .	397
10.5	The Table Maker’s Dilemma . . . . .	397
10.5.1	When there is no need to solve the TMD . . . . .	400
10.5.2	On breakpoints . . . . .	400
10.5.3	Finding the hardest-to-round points . . . . .	404
10.6	Some Implementation Tricks Used in the CRlibm Library . . . . .	427
10.6.1	Rounding test . . . . .	428
10.6.2	Accurate second step . . . . .	429
10.6.3	Error analysis and the accuracy/performance tradeoff . . . . .	429
10.6.4	The point with efficient code . . . . .	430

## **Part IV Extensions 435**

<b>11</b>	<b>Complex Numbers 437</b>
11.1	Introduction . . . . . 437
11.2	Componentwise and Normwise Errors . . . . . 439
11.3	Computing $ad \pm bc$ with an FMA . . . . . 440
11.4	Complex Multiplication . . . . . 442
11.4.1	Complex multiplication without an FMA instruction . . . . . 442
11.4.2	Complex multiplication with an FMA instruction . . . . . 442
11.5	Complex Division . . . . . 443
11.5.1	Error bounds for complex division . . . . . 443
11.5.2	Scaling methods for avoiding over-/underflow in complex division . . . . . 444
11.6	Complex Absolute Value . . . . . 447
11.6.1	Error bounds for complex absolute value . . . . . 447
11.6.2	Scaling for the computation of complex absolute value . . . . . 447
11.7	Complex Square Root . . . . . 449
11.7.1	Error bounds for complex square root . . . . . 449
11.7.2	Scaling techniques for complex square root . . . . . 450
11.8	An Alternative Solution: Exception Handling . . . . . 451

<b>12 Interval Arithmetic</b>	<b>453</b>
12.1 Introduction to Interval Arithmetic . . . . .	454
12.1.1 Definitions and the inclusion property . . . . .	454
12.1.2 Loss of algebraic properties . . . . .	456
12.2 The IEEE 1788-2015 Standard for Interval Arithmetic . . . . .	457
12.2.1 Structuration into levels . . . . .	457
12.2.2 Flavors . . . . .	458
12.2.3 Decorations . . . . .	460
12.2.4 Level 2: discretization issues . . . . .	463
12.2.5 Exact dot product . . . . .	464
12.2.6 Levels 3 and 4: implementation issues . . . . .	464
12.2.7 Libraries implementing IEEE 1788–2015 . . . . .	464
12.3 Intervals with Floating-Point Bounds . . . . .	465
12.3.1 Implementation using floating-point arithmetic . . . . .	465
12.3.2 Difficulties . . . . .	465
12.3.3 Optimized rounding . . . . .	467
12.4 Interval Arithmetic and Roundoff Error Analysis . . . . .	468
12.4.1 Influence of the computing precision . . . . .	468
12.4.2 A more efficient approach: the mid-rad representation . . . . .	471
12.4.3 Variants: affine arithmetic, Taylor models . . . . .	475
12.5 Further Readings . . . . .	476
<b>13 Verifying Floating-Point Algorithms</b>	<b>479</b>
13.1 Formalizing Floating-Point Arithmetic . . . . .	479
13.1.1 Defining floating-point numbers . . . . .	480
13.1.2 Simplifying the definition . . . . .	482
13.1.3 Defining rounding operators . . . . .	483
13.1.4 Extending the set of numbers . . . . .	486
13.2 Formalisms for Verifying Algorithms . . . . .	487
13.2.1 Hardware units . . . . .	487
13.2.2 Floating-point algorithms . . . . .	489
13.2.3 Automating proofs . . . . .	490
13.3 Roundoff Errors and the Gappa Tool . . . . .	493
13.3.1 Computing on bounds . . . . .	494
13.3.2 Counting digits . . . . .	496
13.3.3 Manipulating expressions . . . . .	498
13.3.4 Handling the relative error . . . . .	502
13.3.5 Example: toy implementation of sine . . . . .	503
13.3.6 Example: integer division on Itanium . . . . .	508

<b>14 Extending the Precision</b>	<b>513</b>
14.1 Double-Words, Triple-Words...	514
14.1.1 Double-word arithmetic	515
14.1.2 Static triple-word arithmetic	520
14.2 Floating-Point Expansions	522
14.2.1 Renormalization of floating-point expansions	525
14.2.2 Addition of floating-point expansions	526
14.2.3 Multiplication of floating-point expansions	528
14.2.4 Division of floating-point expansions	531
14.3 Floating-Point Numbers with Batched Additional Exponent	534
14.4 Large Precision Based on a High-Radix Representation	535
14.4.1 Specifications	536
14.4.2 Using arbitrary-precision integer arithmetic for arbitrary-precision floating-point arithmetic	537
14.4.3 A brief introduction to arbitrary-precision integer arithmetic	538
14.4.4 GNU MPFR	541
<b>Appendix A. Number Theory Tools for Floating-Point Arithmetic</b>	<b>553</b>
A.1 Continued Fractions	553
A.2 Euclidean Lattices	556
<b>Appendix B. Previous Floating-Point Standards</b>	<b>561</b>
B.1 The IEEE 754-1985 Standard	561
B.1.1 Formats specified by IEEE 754-1985	561
B.1.2 Rounding modes specified by IEEE 754-1985	562
B.1.3 Operations specified by IEEE 754-1985	562
B.1.4 Exceptions specified by IEEE 754-1985	563
B.2 The IEEE 854-1987 Standard	564
B.2.1 Constraints internal to a format	564
B.2.2 Various formats and the constraints between them	565
B.2.3 Rounding	566
B.2.4 Operations	566
B.2.5 Comparisons	566
B.2.6 Exceptions	566
B.3 The Need for a Revision	566
B.4 The IEEE 754-2008 Revision	567
<b>Bibliography</b>	<b>569</b>
<b>Index</b>	<b>621</b>

# List of Figures

2.1	Positive floating-point numbers for $\beta = 2$ and $p = 3$ . . . . .	20
2.2	Underflow before and after rounding. . . . .	21
2.3	The standard rounding functions. . . . .	23
2.4	Relative error introduced by rounding a real number to nearest floating-point number. . . . .	26
2.5	Values of ulp according to Harrison's definition. . . . .	30
2.6	Values of ulp according to Goldberg's definition. . . . .	31
2.7	Counterexample in radix 3 for a property of Harrison's ulp. . . . .	32
2.8	Conversion from ulps to relative errors. . . . .	36
2.9	Conversion from relative errors to ulps. . . . .	36
3.1	Binary interchange floating-point formats. . . . .	50
3.2	Decimal interchange floating-point formats. . . . .	55
4.1	Independent operations in Dekker's product. . . . .	120
4.2	Convergence of iteration (4.7). . . . .	126
4.3	The various values that should be returned in round-to-nearest mode, assuming $q$ is within one $\text{ulp}(b/a)$ from $b/a$ . . . . .	133
4.4	Converting from binary to decimal, and back. . . . .	145
4.5	Possible values of the binary ulp between two powers of 10. . . . .	146
4.6	Illustration of the conditions (4.34) in the case $b = 2^e$ . . . . .	150
4.7	Position of $Cx$ with respect to the result of Algorithm 4.13. . . . .	160
5.1	Boldo and Melquiond's algorithm for computing $\text{RN}(a + b + c)$ in radix-2 floating-point arithmetic. . . . .	192
6.1	The tangled standardization timeline of floating-point and C language. . . . .	199

7.1	Specification of the implementation of a FP operation. . . . .	234
7.2	Product-anchored FMA computation for normal inputs. . . . .	250
7.3	Addend-anchored FMA computation for normal inputs. . . . .	251
7.4	Cancellation in the FMA. . . . .	252
7.5	FMA $ab - c$ , where $a$ is the smallest subnormal, $ab$ is nevertheless in the normal range, $ c  <  ab $ , and we have an effective subtraction. . . . .	252
7.6	Significand alignment for the single-path algorithm. . . . .	255
8.1	Carry-ripple adder. . . . .	273
8.2	Decimal addition. . . . .	274
8.3	An implementation of the decimal DA box. . . . .	274
8.4	An implementation of the radix-16 DA box. . . . .	275
8.5	Binary carry-save addition. . . . .	276
8.6	Partial carry-save addition. . . . .	276
8.7	Carry-select adder. . . . .	278
8.8	Binary integer multiplication. . . . .	281
8.9	Partial product array for decimal multiplication. . . . .	281
8.10	A multipartite table architecture for the initial approximation of $1/x$ . . . . .	287
8.11	A dual-path floating-point adder. . . . .	288
8.12	Possible implementations of significand subtraction in the close path. . . . .	289
8.13	A dual-path floating-point adder with LZA. . . . .	291
8.14	Basic architecture of a floating-point multiplier without subnormal handling. . . . .	296
8.15	A floating-point multiplier using rounding by injection, without subnormal handling . . . . .	299
8.16	The classic single-path FMA architecture. . . . .	303
8.17	An unrolled SRT4 floating-point divider without subnormal handling. . . . .	306
8.18	The 2Sum and 2Mult operators. . . . .	311
8.19	Iterative accumulator. . . . .	315
8.20	Accumulator and post-normalization unit. . . . .	317
8.21	Accumulation of floating-point numbers into a large fixed-point accumulator . . . . .	318
10.1	The difference between $\ln$ and its degree-5 Taylor approximation in the interval $[1, 2]$ . . . . .	389
10.2	The difference between $\ln$ and its degree-5 minimax approximation in the interval $[1, 2]$ . . . . .	390
10.3	The $L^2$ approximation $p^*$ is obtained by projecting $f$ on the subspace generated by $B_0, B_1, \dots, B_n$ . . . . .	391

10.4	The $\exp(\cos(x))$ function and its degree-4 minimax approximation on $[0, 5]$ . . . . .	392
10.5	A situation where we can return $f(x)$ correctly rounded. . . . .	398
10.6	A situation where we cannot return $f(x)$ correctly rounded. . . . .	398
12.1	Comparison of the relative widths of matrices computed using <code>MMu13</code> and <code>MMu15</code> . . . . .	474
A.1	The lattice $\mathbb{Z}(2, 0) \oplus \mathbb{Z}(1, 2)$ . . . . .	557
A.2	Two bases of the lattice $\mathbb{Z}(2, 0) \oplus \mathbb{Z}(1, 2)$ . . . . .	557



# List of Tables

1.1	Results obtained by running Program 1.1 on a Macintosh with an Intel Core i5 processor. . . . .	11
2.1	Rounding a significand using the “round” and “sticky” bits. . . . .	24
2.2	ARRE and MRRE for various formats. . . . .	43
3.1	Main parameters of the binary interchange formats of size up to 128 bits specified by the 754-2008 standard [267]. In some articles and software libraries, 128-bit formats were sometimes called “quad precision.” However, quad precision was not specified by IEEE 754-1985. . . . .	49
3.2	Main parameters of the decimal interchange formats of size up to 128 bits specified by the 754-2008 standard [267]. . . . .	49
3.3	How to interpret the encoding of an IEEE 754 binary floating-point number. . . . .	51
3.4	Parameters of the encodings of binary interchange formats [267]. As stated above, in some articles and software libraries, 128-bit formats were called “quad precision.” However, quad precision was not specified by IEEE 754-1985. . . . .	51
3.5	Extremal values of the IEEE 754-2008 binary interchange formats. . . . .	52
3.6	Binary encoding of various floating-point data in the binary32 format. . . . .	52
3.7	Width (in bits) of the various fields in the encodings of the decimal interchange formats of size up to 128 bits [267]. . . . .	56
3.8	Decimal encoding of a decimal floating-point number (IEEE 754-2008). . . . .	58
3.9	Binary encoding of a decimal floating-point number (IEEE 754-2008). . . . .	59

3.10	Decoding the dectet $b_0b_1b_2 \cdots b_9$ of a densely packed decimal encoding to three decimal digits $d_0d_1d_2$ . . . . .	60
3.11	Encoding the three consecutive decimal digits $d_0d_1d_2$ , each of them being represented in binary by four bits, into a 10-bit dectet $b_0b_1b_2 \cdots b_9$ of a densely packed decimal encoding. . .	60
3.12	Parameters of the interchange formats. . . . .	64
3.13	Parameters of the binary256 and binary1024 interchange formats deduced from Table 3.12. . . . .	64
3.14	Parameters of the decimal256 and decimal512 interchange formats deduced from Table 3.12. . . . .	65
3.15	Extended format parameters in the IEEE 754-2008 standard. . . . .	65
3.16	Minimum number of decimal digits in the decimal external character sequence that allows for an error-free write-read cycle, for the various basic binary formats of the standard. . .	74
3.17	Results returned by Program 3.1 on a 64-bit Intel platform. . . . .	80
3.18	Execution times of decimal operations on the IBM z10. . . . .	88
4.1	Quadratic convergence of iteration (4.7). . . . .	126
4.2	Converting from binary to decimal and back without error. . . . .	146
4.3	Comparison of various methods for checking Algorithm 4.13. . . . .	161
5.1	Errors of various methods for $\sum x_i$ with $x_i = \text{RN}(\cos(i))$ . . .	184
5.2	Errors of various methods for $\sum x_i$ with $x_i = \text{RN}(1/i)$ . . . . .	185
6.1	FLT_EVAL_METHOD macro values. . . . .	205
6.2	FORTTRAN allowable alternatives. . . . .	220
6.3	FORTTRAN forbidden alternatives. . . . .	220
7.1	Specification of addition/subtraction when both $x$ and $y$ are zero. . . . .	241
7.2	Specification of addition for floating-point data of positive sign. . . . .	241
7.3	Specification of subtraction for floating-point data of positive sign. . . . .	242
7.4	Specification of multiplication for floating-point data of positive sign. . . . .	246
7.5	Special values for $ x / y $ . . . . .	257
7.6	Special values for $\text{sqrt}(x)$ . . . . .	259

8.1	Minimum size of an exact accumulator for the main IEEE formats . . . . .	310
9.1	Standard integer encoding of binary32 data. . . . .	324
9.2	Some floating-point data encoded by $X$ . . . . .	330
9.3	Speedups for some binary32 custom operators in software . . . . .	374
10.1	Some worst cases for range reduction. . . . .	385
10.2	Degrees of minimax polynomial approximations for various functions and approximation ranges. . . . .	385
10.3	Some results for small values in the binary32 format, assuming rounding to nearest. . . . .	401
10.4	Some results for small values in the binary64 format, assuming rounding to nearest. . . . .	402
10.5	Some results for small values in the binary64 format, assuming rounding toward $-\infty$ . . . . .	403
10.6	Hardest-to-round points for functions $e^x$ and $e^x - 1$ in binary32 arithmetic. . . . .	406
10.7	Hardest-to-round points for functions $2^x$ and $10^x$ in binary32 arithmetic. . . . .	407
10.8	Hardest-to-round points for functions $\ln(x)$ and $\ln(1 + x)$ in binary32 arithmetic. . . . .	408
10.9	Hardest-to-round points for functions $\log_2(x)$ and $\log_{10}(x)$ in binary32 arithmetic. . . . .	409
10.10	Hardest-to-round points for functions $\sinh(x)$ , $\cosh(x)$ and $\tanh(x)$ in binary32 arithmetic. . . . .	410
10.11	Hardest-to-round points for the inverse hyperbolic functions in binary32 arithmetic. . . . .	411
10.12	Hardest-to-round points for functions $\sin(x)$ and $\cos(x)$ in binary32 arithmetic. . . . .	412
10.13	Hardest-to-round points for functions $\tan(x)$ , $\operatorname{asin}(x)$ , $\operatorname{acos}(x)$ and $\operatorname{atan}(x)$ in binary32 arithmetic. . . . .	413
10.14	Hardest-to-round points for functions $\operatorname{erf}(x)$ and $\operatorname{erfc}(x)$ in binary32 arithmetic. . . . .	414
10.15	Hardest-to-round points for function $\Gamma(x)$ in binary32 arithmetic. . . . .	415
10.16	Hardest-to-round points for function $\ln( \Gamma(x) )$ in binary32 arithmetic. . . . .	416
10.17	Hardest-to-round points for the Bessel functions in binary32 arithmetic. . . . .	417
10.18	Hardest-to-round points for functions $e^x$ , $e^x - 1$ , $2^x$ , and $10^x$ in binary64 arithmetic. . . . .	420

10.19	Hardest-to-round points for functions $\ln(x)$ and $\ln(1 + x)$ in binary64 arithmetic. . . . .	421
10.20	Hardest-to-round points for functions $\log_2(x)$ and $\log_{10}(x)$ in binary64 arithmetic. . . . .	422
10.21	Hardest-to-round points for functions $\sinh(x)$ and $\cosh(x)$ in binary64 arithmetic. . . . .	423
10.22	Hardest-to-round points for the inverse hyperbolic functions in binary64 arithmetic. . . . .	424
10.23	Hardest-to-round points for the trigonometric functions in binary64 arithmetic. . . . .	425
10.24	Hardest-to-round points for the inverse trigonometric functions in binary64 arithmetic. . . . .	426
12.1	Difference of the interval evaluation of Machin's formula for various precisions. . . . .	470
14.1	Asymptotic complexities of some multiplication algorithms. . . . .	539
B.1	Main parameters of the formats specified by the IEEE 754-1985 standard. . . . .	562
B.2	The thresholds for conversion from and to a decimal string, as specified by the IEEE 754-1985 standard. . . . .	563
B.3	Correctly rounded decimal conversion range, as specified by the IEEE 754-1985 standard. . . . .	564

# Preface

FLOATING-POINT ARITHMETIC is by far the most widely used way of approximating real-number arithmetic for performing numerical calculations on modern computers. A rough presentation of floating-point arithmetic requires only a few words: a number  $x$  is represented in radix  $\beta$  floating-point arithmetic with a sign  $s$ , a significand  $m$ , and an exponent  $e$ , such that  $x = s \times m \times \beta^e$ . Making such arithmetic reliable, fast, and portable is however a very complex task. Although it could be argued that, to some extent, the concept of floating-point arithmetic (in radix 60) was invented by the Babylonians, or that it is the underlying arithmetic of the slide rule, its first modern implementation appeared in Konrad Zuse's Z1 and Z3 computers.

A vast quantity of very diverse arithmetics was implemented between the 1960s and the early 1980s. The radix (radices 2, 4, 8, 16, and 10, and even radix 3, were then considered), and the sizes of the significand and exponent fields were not standardized. The approaches for rounding and for handling underflows, overflows, or "forbidden operations" (such as  $5/0$  or  $\sqrt{-3}$ ) were significantly different from one machine to another. This lack of standardization made it difficult to write reliable and portable numerical software.

Pioneering scientists including Brent, Cody, Kahan, and Kuki highlighted the relevant key concepts for designing an arithmetic that could be both useful for programmers and practical for implementers. These efforts resulted in the IEEE 754-1985 standard for radix-2 floating-point arithmetic. The standardization process was expertly orchestrated by William Kahan. The IEEE 754-1985 standard was a key factor in improving the quality of the computational environment available to programmers. A new version, the IEEE 754-2008 standard, was released in August 2008. It specifies radix-2 and radix-10 floating-point arithmetic. At the time of writing these lines, a working group is preparing a new version of IEEE 754, which should be released in 2018. It will not be very different from the 2008 version.

By carefully specifying the behavior of the arithmetic operators, the IEEE 754-1985 and 754-2008 standards allowed researchers and engineers to design extremely powerful yet portable algorithms, for example, to compute very accurate sums and dot products, and to formally prove some critical parts of

programs. Unfortunately, the subtleties of the standard are little known by the nonexpert user. Even more worrying, they are sometimes overlooked by compiler designers. As a consequence, floating-point arithmetic is sometimes conceptually misunderstood and is often far from being exploited to its full potential.

This led us to the decision to compile into a book selected parts of the vast knowledge of floating-point arithmetic. This book is designed for programmers of numerical applications, compiler designers, programmers of floating-point algorithms, designers of arithmetic operators (floating-point adders, multipliers, dividers, ...), and more generally students and researchers in numerical analysis who wish to more accurately understand a tool that they manipulate on an everyday basis. During the writing, we tried, whenever possible, to illustrate the techniques described by an actual program, in order to allow a more direct practical use for coding and design.

The first part of the book presents the history and basic concepts of floating-point arithmetic (formats, exceptions, correct rounding, etc.) and various aspects of the 2008 version of the IEEE 754 standard. The second part shows how the features of the standard can be used to develop effective and nontrivial algorithms. This includes summation algorithms, and division and square root relying on a fused multiply-add. This part also discusses issues related to compilers and languages. The third part then explains how to implement floating-point arithmetic, both in software (on an integer processor) and in hardware (VLSI or reconfigurable circuits). It also surveys the implementation of elementary functions. The fourth part presents some extensions: complex numbers, interval arithmetic, verification of floating-point arithmetic, and extension of the precision. In the Appendix, the reader will find an introduction to relevant number theory tools and a brief presentation of the standards that predated IEEE 754-2008.

## Acknowledgments

Some of our colleagues around the world, in academia and industry, and several students from École normale supérieure de Lyon and Université de Lyon greatly helped us by discussing with us, giving advice and reading preliminary versions of this edition, or by giving comments on the previous one that (hopefully) helped us to improve it: Nicolas Brisebarre, Jean-Yves L'Excellent, Warren Ferguson, Christian Fleck, John Harrison, Nicholas Higham, Tim Leonard, Dan Liew, Nicolas Louvet, David Lutz, Peter Markstein, Marc Mezzarobba, Kai Torben Ohlhus, Antoine Plet, Valentina Popescu, Guillaume Revy, Siegfried Rump, Damien Stehlé, and Nick Trefethen. We thank them all for their suggestions and interest.

Our dear colleague and friend Peter Kornerup left us recently. The computer arithmetic community misses one of its pioneers, and we miss a very kind and friendly person.

Grenoble, France  
Villeurbanne, France  
Lyon, France  
Toulouse, France  
Lyon, France  
Orsay, France  
Lyon, France  
Lyon, France  
Lyon, France

Nicolas Brunie  
Florent de Dinechin  
Claude-Pierre Jeannerod  
Mioara Joldes  
Vincent Lefèvre  
Guillaume Melquiond  
Jean-Michel Muller  
Nathalie Revol  
Serge Torres