

Contents

Preface	xv
List of Figures	xvii
List of Tables	xxi
I Introduction, Basic Definitions, and Standards	1
1 Introduction	3
1.1 Some History	3
1.2 Desirable Properties	6
1.3 Some Strange Behaviors	7
1.3.1 Some famous bugs	7
1.3.2 Difficult problems	8
2 Definitions and Basic Notions	13
2.1 Floating-Point Numbers	13
2.2 Rounding	20
2.2.1 Rounding modes	20
2.2.2 Useful properties	22
2.2.3 Relative error due to rounding	23
2.3 Exceptions	25
2.4 Lost or Preserved Properties of the Arithmetic on the Real Numbers	27
2.5 Note on the Choice of the Radix	29
2.5.1 Representation errors	29
2.5.2 A case for radix 10	30
2.6 Tools for Manipulating Floating-Point Errors	32
2.6.1 The ulp function	32
2.6.2 Errors in ulps and relative errors	37
2.6.3 An example: iterated products	37
2.6.4 Unit roundoff	39
2.7 Note on Radix Conversion	40

2.7.1	Conditions on the formats	40
2.7.2	Conversion algorithms	43
2.8	The Fused Multiply-Add (FMA) Instruction	51
2.9	Interval Arithmetic	51
2.9.1	Intervals with floating-point bounds	52
2.9.2	Optimized rounding	52
3	Floating-Point Formats and Environment	55
3.1	The IEEE 754-1985 Standard	56
3.1.1	Formats specified by IEEE 754-1985	56
3.1.2	Little-endian, big-endian	60
3.1.3	Rounding modes specified by IEEE 754-1985	61
3.1.4	Operations specified by IEEE 754-1985	62
3.1.5	Exceptions specified by IEEE 754-1985	66
3.1.6	Special values	69
3.2	The IEEE 854-1987 Standard	70
3.2.1	Constraints internal to a format	70
3.2.2	Various formats and the constraints between them	71
3.2.3	Conversions between floating-point numbers and decimal strings	72
3.2.4	Rounding	73
3.2.5	Operations	73
3.2.6	Comparisons	74
3.2.7	Exceptions	74
3.3	The Need for a Revision	74
3.3.1	A typical problem: “double rounding”	75
3.3.2	Various ambiguities	77
3.4	The New IEEE 754-2008 Standard	79
3.4.1	Formats specified by the revised standard	80
3.4.2	Binary interchange format encodings	81
3.4.3	Decimal interchange format encodings	82
3.4.4	Larger formats	92
3.4.5	Extended and extendable precisions	92
3.4.6	Attributes	93
3.4.7	Operations specified by the standard	97
3.4.8	Comparisons	99
3.4.9	Conversions	99
3.4.10	Default exception handling	100
3.4.11	Recommended transcendental functions	103
3.5	Floating-Point Hardware in Current Processors	104
3.5.1	The common hardware denominator	104
3.5.2	Fused multiply-add	104
3.5.3	Extended precision	104
3.5.4	Rounding and precision control	105

3.5.5	SIMD instructions	106
3.5.6	Floating-point on x86 processors: SSE2 versus x87 . . .	106
3.5.7	Decimal arithmetic	107
3.6	Floating-Point Hardware in Recent Graphics Processing Units	108
3.7	Relations with Programming Languages	109
3.7.1	The Language Independent Arithmetic (LIA) standard	109
3.7.2	Programming languages	110
3.8	Checking the Environment	110
3.8.1	MACHAR	111
3.8.2	Paranoia	111
3.8.3	UCBTest	115
3.8.4	TestFloat	116
3.8.5	IeeeCC754	116
3.8.6	Miscellaneous	116
II	Cleverly Using Floating-Point Arithmetic	117
4	Basic Properties and Algorithms	119
4.1	Testing the Computational Environment	119
4.1.1	Computing the radix	119
4.1.2	Computing the precision	121
4.2	Exact Operations	122
4.2.1	Exact addition	122
4.2.2	Exact multiplications and divisions	124
4.3	Accurate Computations of Sums of Two Numbers	125
4.3.1	The Fast2Sum algorithm	126
4.3.2	The 2Sum algorithm	129
4.3.3	If we do not use rounding to nearest	131
4.4	Computation of Products	132
4.4.1	Veltkamp splitting	132
4.4.2	Dekker's multiplication algorithm	135
4.5	Complex numbers	139
4.5.1	Various error bounds	140
4.5.2	Error bound for complex multiplication	141
4.5.3	Complex division	144
4.5.4	Complex square root	149
5	The Fused Multiply-Add Instruction	151
5.1	The 2MultFMA Algorithm	152
5.2	Computation of Residuals of Division and Square Root	153
5.3	Newton–Raphson-Based Division with an FMA	155
5.3.1	Variants of the Newton–Raphson iteration	155

5.3.2	Using the Newton–Raphson iteration for correctly rounded division	160
5.4	Newton–Raphson-Based Square Root with an FMA	167
5.4.1	The basic iterations	167
5.4.2	Using the Newton–Raphson iteration for correctly rounded square roots	168
5.5	Multiplication by an Arbitrary-Precision Constant	171
5.5.1	Checking for a given constant C if Algorithm 5.2 will always work	172
5.6	Evaluation of the Error of an FMA	175
5.7	Evaluation of Integer Powers	177
6	Enhanced Floating-Point Sums, Dot Products, and Polynomial Values	181
6.1	Preliminaries	182
6.1.1	Floating-point arithmetic models	183
6.1.2	Notation for error analysis and classical error estimates	184
6.1.3	Properties for deriving running error bounds	187
6.2	Computing Validated Running Error Bounds	188
6.3	Computing Sums More Accurately	190
6.3.1	Reordering the operands, and a bit more	190
6.3.2	Compensated sums	192
6.3.3	Implementing a “long accumulator”	199
6.3.4	On the sum of three floating-point numbers	199
6.4	Compensated Dot Products	201
6.5	Compensated Polynomial Evaluation	203
7	Languages and Compilers	205
7.1	A Play with Many Actors	205
7.1.1	Floating-point evaluation in programming languages .	206
7.1.2	Processors, compilers, and operating systems	208
7.1.3	In the hands of the programmer	209
7.2	Floating Point in the C Language	209
7.2.1	Standard C99 headers and IEEE 754-1985 support . . .	209
7.2.2	Types	210
7.2.3	Expression evaluation	213
7.2.4	Code transformations	216
7.2.5	Enabling unsafe optimizations	217
7.2.6	Summary: a few horror stories	218
7.3	Floating-Point Arithmetic in the C++ Language	220
7.3.1	Semantics	220
7.3.2	Numeric limits	221
7.3.3	Overloaded functions	222
7.4	FORTTRAN Floating Point in a Nutshell	223

7.4.1	Philosophy	223
7.4.2	IEEE 754 support in FORTRAN	226
7.5	Java Floating Point in a Nutshell	227
7.5.1	Philosophy	227
7.5.2	Types and classes	228
7.5.3	Infinities, NaNs, and signed zeros	230
7.5.4	Missing features	231
7.5.5	Reproducibility	232
7.5.6	The BigDecimal package	233
7.6	Conclusion	234
III	Implementing Floating-Point Operators	237
8	Algorithms for the Five Basic Operations	239
8.1	Overview of Basic Operation Implementation	239
8.2	Implementing IEEE 754-2008 Rounding	241
8.2.1	Rounding a nonzero finite value with unbounded exponent range	241
8.2.2	Overflow	243
8.2.3	Underflow and subnormal results	244
8.2.4	The inexact exception	245
8.2.5	Rounding for actual operations	245
8.3	Floating-Point Addition and Subtraction	246
8.3.1	Decimal addition	249
8.3.2	Decimal addition using binary encoding	250
8.3.3	Subnormal inputs and outputs in binary addition	251
8.4	Floating-Point Multiplication	251
8.4.1	Normal case	252
8.4.2	Handling subnormal numbers in binary multiplication	252
8.4.3	Decimal specifics	253
8.5	Floating-Point Fused Multiply-Add	254
8.5.1	Case analysis for normal inputs	254
8.5.2	Handling subnormal inputs	258
8.5.3	Handling decimal cohorts	259
8.5.4	Overview of a binary FMA implementation	259
8.6	Floating-Point Division	262
8.6.1	Overview and special cases	262
8.6.2	Computing the significand quotient	263
8.6.3	Managing subnormal numbers	264
8.6.4	The inexact exception	265
8.6.5	Decimal specifics	265
8.7	Floating-Point Square Root	265
8.7.1	Overview and special cases	265

8.7.2	Computing the significand square root	266
8.7.3	Managing subnormal numbers	267
8.7.4	The inexact exception	267
8.7.5	Decimal specifics	267
9	Hardware Implementation of Floating-Point Arithmetic	269
9.1	Introduction and Context	269
9.1.1	Processor internal formats	269
9.1.2	Hardware handling of subnormal numbers	270
9.1.3	Full-custom VLSI versus reconfigurable circuits	271
9.1.4	Hardware decimal arithmetic	272
9.1.5	Pipelining	273
9.2	The Primitives and Their Cost	274
9.2.1	Integer adders	274
9.2.2	Digit-by-integer multiplication in hardware	280
9.2.3	Using nonstandard representations of numbers	280
9.2.4	Binary integer multiplication	281
9.2.5	Decimal integer multiplication	283
9.2.6	Shifters	284
9.2.7	Leading-zero counters	284
9.2.8	Tables and table-based methods for fixed-point function approximation	286
9.3	Binary Floating-Point Addition	288
9.3.1	Overview	288
9.3.2	A first dual-path architecture	289
9.3.3	Leading-zero anticipation	291
9.3.4	Probing further on floating-point adders	295
9.4	Binary Floating-Point Multiplication	296
9.4.1	Basic architecture	296
9.4.2	FPGA implementation	296
9.4.3	VLSI implementation optimized for delay	298
9.4.4	Managing subnormals	301
9.5	Binary Fused Multiply-Add	302
9.5.1	Classic architecture	303
9.5.2	To probe further	305
9.6	Division	305
9.6.1	Digit-recurrence division	306
9.6.2	Decimal division	309
9.7	Conclusion: Beyond the FPU	309
9.7.1	Optimization in context of standard operators	310
9.7.2	Operation with a constant operand	311
9.7.3	Block floating point	313
9.7.4	Specific architectures for accumulation	313
9.7.5	Coarser-grain operators	317

9.8	Probing Further	320
10	Software Implementation of Floating-Point Arithmetic	321
10.1	Implementation Context	322
10.1.1	Standard encoding of binary floating-point data	322
10.1.2	Available integer operators	323
10.1.3	First examples	326
10.1.4	Design choices and optimizations	328
10.2	Binary Floating-Point Addition	329
10.2.1	Handling special values	330
10.2.2	Computing the sign of the result	332
10.2.3	Swapping the operands and computing the alignment shift	333
10.2.4	Getting the correctly rounded result	335
10.3	Binary Floating-Point Multiplication	341
10.3.1	Handling special values	341
10.3.2	Sign and exponent computation	343
10.3.3	Overflow detection	345
10.3.4	Getting the correctly rounded result	346
10.4	Binary Floating-Point Division	349
10.4.1	Handling special values	350
10.4.2	Sign and exponent computation	351
10.4.3	Overflow detection	354
10.4.4	Getting the correctly rounded result	355
10.5	Binary Floating-Point Square Root	361
10.5.1	Handling special values	362
10.5.2	Exponent computation	364
10.5.3	Getting the correctly rounded result	365
IV	Elementary Functions	373
11	Evaluating Floating-Point Elementary Functions	375
11.1	Basic Range Reduction Algorithms	379
11.1.1	Cody and Waite's reduction algorithm	379
11.1.2	Payne and Hanek's algorithm	381
11.2	Bounding the Relative Error of Range Reduction	382
11.3	More Sophisticated Range Reduction Algorithms	384
11.3.1	An example of range reduction for the exponential function	386
11.3.2	An example of range reduction for the logarithm	387
11.4	Polynomial or Rational Approximations	388
11.4.1	L^2 case	389
11.4.2	L^∞ , or minimax case	390

11.4.3	"Truncated" approximations	392
11.5	Evaluating Polynomials	393
11.6	Correct Rounding of Elementary Functions to binary64	394
11.6.1	The Table Maker's Dilemma and Ziv's onion peeling strategy	394
11.6.2	When the TMD is solved	395
11.6.3	Rounding test	396
11.6.4	Accurate second step	400
11.6.5	Error analysis and the accuracy/performance tradeoff	401
11.7	Computing Error Bounds	402
11.7.1	The point with efficient code	402
11.7.2	Example: a "double-double" polynomial evaluation	403
12	Solving the Table Maker's Dilemma	405
12.1	Introduction	405
12.1.1	The Table Maker's Dilemma	406
12.1.2	Brief history of the TMD	410
12.1.3	Organization of the chapter	411
12.2	Preliminary Remarks on the Table Maker's Dilemma	412
12.2.1	Statistical arguments: what can be expected in practice	412
12.2.2	In some domains, there is no need to find worst cases	416
12.2.3	Deducing the worst cases from other functions or domains	419
12.3	The Table Maker's Dilemma for Algebraic Functions	420
12.3.1	Algebraic and transcendental numbers and functions	420
12.3.2	The elementary case of quotients	422
12.3.3	Around Liouville's theorem	424
12.3.4	Generating bad rounding cases for the square root using Hensel 2-adic lifting	425
12.4	Solving the Table Maker's Dilemma for Arbitrary Functions	429
12.4.1	Lindemann's theorem: application to some transcendental functions	429
12.4.2	A theorem of Nesterenko and Waldschmidt	430
12.4.3	A first method: tabulated differences	432
12.4.4	From the TMD to the distance between a grid and a segment	434
12.4.5	Linear approximation: Lefèvre's algorithm	436
12.4.6	The SLZ algorithm	443
12.4.7	Periodic functions on large arguments	448
12.5	Some Results	449
12.5.1	Worst cases for the exponential, logarithmic, trigonometric, and hyperbolic functions	449
12.5.2	A special case: integer powers	458
12.6	Current Limits and Perspectives	458

V Extensions	461
13 Formalisms for Certifying Floating-Point Algorithms	463
13.1 Formalizing Floating-Point Arithmetic	463
13.1.1 Defining floating-point numbers	464
13.1.2 Simplifying the definition	466
13.1.3 Defining rounding operators	467
13.1.4 Extending the set of numbers	470
13.2 Formalisms for Certifying Algorithms by Hand	471
13.2.1 Hardware units	471
13.2.2 Low-level algorithms	472
13.2.3 Advanced algorithms	473
13.3 Automating Proofs	474
13.3.1 Computing on bounds	475
13.3.2 Counting digits	477
13.3.3 Manipulating expressions	479
13.3.4 Handling the relative error	483
13.4 Using Gappa	484
13.4.1 Toy implementation of sine	484
13.4.2 Integer division on Itanium	488
14 Extending the Precision	493
14.1 Double-Words, Triple-Words.	494
14.1.1 Double-word arithmetic	495
14.1.2 Static triple-word arithmetic	498
14.1.3 Quad-word arithmetic	500
14.2 Floating-Point Expansions	503
14.3 Floating-Point Numbers with Batched Additional Exponent .	509
14.4 Large Precision Relying on Processor Integers	510
14.4.1 Using arbitrary-precision integer arithmetic for arbitrary-precision floating-point arithmetic	512
14.4.2 A brief introduction to arbitrary-precision integer arithmetic	513
VI Perspectives and Appendix	517
15 Conclusion and Perspectives	519
16 Appendix: Number Theory Tools for Floating-Point Arithmetic	521
16.1 Continued Fractions	521
16.2 The LLL Algorithm	524
Bibliography	529
Index	567