THE EXACT DOT PRODUCT

ULRICH KULISCH AND VAN SNYDER

1. Introduction

The exact dot product is computed by use of complete arithmetic and a complete format associated with a floating-point format. A complete format is a signed fixed-point format which has enough digit positions such that multiplication of pairs of non-exceptional floating-point numbers and accumulation of such summands have exact results.

A complete expression is composed of operands and operations. Three kinds of operands are permitted: finite floating-point numbers, exact products of two such numbers, and data of the format *complete*. Such operands can be added or subtracted in an arbitrary order. All operations are to be exact. The result of *complete arithmetic* reflects the complete information as given by the input data and the operations in the expression. Not a single bit is lost. All possible information from complete operands and accumulation operations is represented in the result.

2. Complete Arithmetic

2.1. **Complete format.** A floating-point format $\mathbb{F}$ is characterized by four parameters, the radix or base b of the number system in use, the number l of digits in the mantissa, and the least and the greatest exponents $emin$ and $emax$; thus $\mathbb{F} = \mathbb{F}(b, l, emax, emin)$.

The product of two floating-point numbers with l digits has $2l$ digits. In order to represent all products of floating-point numbers with negative exponents exactly another $|2emin|$ digits are required after the point for the fractional part of the complete format. In order to represent all products of floating-point numbers with positive exponents exactly $2emax$ digits before the point are required for the integer part of the complete format. Thus all products of floating-point numbers are representable in a fixed-point register of length $2emax + 2l + 2|emin|$ without loss of information. Products of floating-point numbers can be added into a register of this size. Each accumulation can result in an (intermediate) overflow of the integer part by one bit. To accommodate possible overflows another k bits before the point allows b^k accumulations to be computed without loss of information due to overflow. The final result of the accumulation if rounded into a floating-point number is assumed to have an exponent between $emax$ and $emin$; otherwise the problem needs to be scaled.

Complete format is characterized by two numbers $m = k + 2emax$, which is the number of digits before the point, and $d = 2l + 2|emin|$, which is the number of digits after the point. In detail, it is composed of four parts, *viz.* a three-bit status, a one-bit sign, an m-bit integer part, and a d-bit fractional part, where k

is chosen such that $L = m + d + 4$ is a multiple of eight. For IEEE 754 binary64 format with $l = 53$, $emin = -1022$, and $emax = 1023$, the recommended value for $L = m + d + 4$ is 4288 bits (536 bytes or 67 words of 64 bits) allowing 2^{88} accumulations before overflow could occur. The status field has one of the values *exact*, *inexact*, $-\infty$, $+\infty$, *overflow*, *sNaN*, or *qNaN*.

For IEEE 754 binary32 format with $l = 24$, $emin = -126$, and $emax = 127$, the recommended value for L is 640 bits (80 bytes or 10 words of 64 bits).

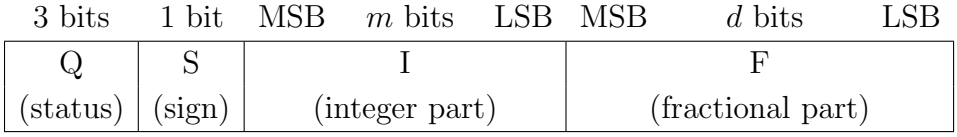

Figure 1—Binary interchange, binary complete format

For IEEE 754 decimal64 format with 20 bits of overflow, $m \geq 789$ and $d = 798$. Using the encoding specified in IEEE 754 (ten bits representing three decimal digits), and again assuming the total number of bits to be a multiple of eight, 5312 bits (664 bytes) are recommended.

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. Complete formats corresponding to other floating-point formats may be provided.

2.2. **Complete operations.**

2.2.1. ***Convert***. An implementation shall provide the following operation, producing a result of complete or floating-point format, as specified by the operation. The *source* operand may be of complete format, floating-point format, or integer format. The radix of the operand shall be the same as the radix of the result.

- *formatOf*-**convert**(*source*)
  Conversion of an exceptional floating-point operand (*sNaN*, *qNaN*, $-\infty$, $+\infty$) to complete format shall cause the corresponding status of the result to be set, and the mantissa of the operand shall be copied to the high-order l bits of the fraction part of the result. Conversion of an exceptional complete-format operand (*overflow*, *sNaN*, *qNaN*, $-\infty$, $+\infty$) to another complete format shall preserve the status; if the value of m or d of the result is less than the corresponding value for the operand, the low-order bits of the integer part and the high-order bits of the fraction part shall be preserved. Conversion of a complete operand with overflow status to floating-point format shall produce floating-point infinity with the same sign. Otherwise conversion of an exceptional complete-format operand (*sNaN*, *qNaN*, $-\infty$, $+\infty$) to floating-point format shall produce a corresponding exceptional floating-point result, with the mantissa equal to the high-order l bits of the fraction part. Conversion of a normal floating-point datum to complete format, or of a complete-format datum

to a complete format with values of m and d not less than those of the operand, shall be exact. Conversion of a normal (*exact*, *inexact*) complete format datum to a floating-point format shall round as specified in clause 4 of IEEE 754. Floating-point overflow might signal; if floating-point overflow signals, the result shall be a floating-point infinity of the same sign as the operand. The result of conversion of a normal (*exact, inexact*) complete format datum to a complete format with a value of m less than that of the operand might have overflow status. Conversion of a normal (*exact*, *inexact*) complete format datum to a complete format with a value of d less than that of the operand shall round as specified in Clause 4 of IEEE 754, and the result shall have *inexact* status, unless the result has *overflow* status. If the operand has *inexact* status the result shall have *inexact* status, unless the result has *overflow* status. Bits of the integer and fraction parts that are not derived from the operand shall be zero.

2.2.2. ***Addition and Subtraction***. In addition to the indicators of operand and result formats specified in subclause 5.1 of IEEE 754, this document specifies

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

An implementation shall provide the following operations, producing a complete-format result. The operands shall be of complete, floating-point, or integer format, and of the same radix as the result. The result shall be computed using complete arithmetic, as if any operand that is not of complete format, or of a complete format different from the result, were first converted to the same complete format as the result using the **convert** operation.

- *completeFormatOf*-**completeAddition**(*source1*, *source2*)
  The operation **completeAddition**(x, y) computes $x + y$.
- *completeFormatOf*-**completeSubtraction**(*source1*, *source2*)
  The operation **completeSubtraction**(x, y) computes $x - y$.

2.2.3. ***Multiply and Add***. An implementation shall provide the following operation, producing a complete-format result. The *source1* and *source2* operands shall be of floating-point format, and of the same radix as the result. The *source3* operand shall be a complete-format operand of the same radix as the result. The multiplication shall be computed without loss of any digits, the addition shall be computed using complete arithmetic in the complete format corresponding to the floating-point operands, and the result converted if necessary to the specified result format as if by application of the convert operation.

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

If any of the operands in this operation is an exceptional floating-point datum $(sNaN, qNaN, -\infty, +\infty)$ the status field is set appropriately.

References

[1] IEEE Floating-Point Arithmetic Standard 754, 2008.

[2] R. Lohner: *Interval Arithmetic in Staggered Correction Format.* In: E. Adams and U. Kulisch (eds.): *Scientific Computing with Automatic Result Verification*, pp. 301–321. Academic Press, (1993).

[3] F. Blomquist, W. Hofschuster, W. Krämer: *A Modified Staggered Correction Arithmetic with Enhanced Accuracy and Very Wide Exponent Range.* In: A. Cuyt et al. (eds.): *Numerical Validation in Current Hardware Architectures*, Lecture Notes in Computer Science LNCS, vol. 5492, Springer-Verlag Berlin Heidelberg, 41-67, 2009.

[4] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, A. Wiethoff: *C–XSC, A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, Berlin/Heidelberg/New York, 1993. See also: `http://www.math.uni-wuppertal.de/~xsc/` resp. `http://www.xsc.de/`.

[5] U. Kulisch: *Computer Arithmetic and Validity – Theory, Implementation, and Applications*, de Gruyter, Berlin, New York, 2008.

[6] U. Kulisch, V. Snyder: *The Exact Dot Product as Basic Tool for Long Interval Arithmetic*, Computing, Vol 91, Issue 4, pp. 307-313, Springer 2011.

[7] U. Kulisch: *Very fast and exact accumulation of products*, Computing, Vol. 91, Issue 4, pp. 397-405, Springer 2011.

[8] *IBM System/370 RPQ. High Accuracy Arithmetic.* SA 22-7093-0, IBM Deutschland GmbH (Department 3282, Schönaicher Strasse 220, D-71032 Böblingen), 1984.

[9] *ACRITH-XSC: IBM High Accuracy Arithmetic, Extended Scientific Computation.* Version 1, Release 1. IBM Deutschland GmbH (Schönaicher Strasse 220, D-71032 Böblingen), 1990.
1. General Information, GC33-6461-01.
2. Reference, SC33-6462-00.
3. Sample Programs, SC33-6463-00.
4. How To Use, SC33-6464-00.
5. Syntax Diagrams, SC33-6466-00.

[10] S. Oishi, K. Tanabe, T. Ogita and S. M. Rump: Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices, *Journal of Computational and Applied Mathematics* 205 (2007), 533–544.