COMPUTER ARITHMETIC IN THEORY AND PRACTICE

Ulrich W. Kulisch *Willard I. Miranker*

COMPUTER ARITHMETIC IN THEORY AND PRACTICE

Ulrich W. Kulisch

FAKULTÄT FÜR MATHEMATIK
UNIVERSITÄT KARLSRUHE
KARLSRUHE, WEST GERMANY

Willard L. Miranker

MATHEMATICAL SCIENCES DEPARTMENT
IBM THOMAS J. WATSON RESEARCH CENTER
YORKTOWN HEIGHTS, NEW YORK

This is a volume in
COMPUTER SCIENCE AND APPLIED MATHEMATICS
A Series of Monographs and Textbooks
Editor: WERNER RHEINBOLDT
A complete list of titles in this series appears at the end of this volume.

1981

ACADEMIC PRESS
A Subsidiary of Harcourt Brace Jovanovich, Publishers
New York London Toronto Sydney San Francisco

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 Oval Road, London NW1 7DX

Library of Congress Cataloging in Publication Data

Kulisch, Ulrich.
Computer arithmetic in theory and practice.

(Computer science and applied mathematics)
Bibliography: p.
Includes index.
1. Algebra, Abstract. 2. Interval analysis (Mathematics) 3. Floating-point arithmetic.
I. Miranker, Willard L., joint author. II. Title.
QA162.K84 512'.02 80-765
ISBN 0-12-428650-X

PRINTED IN THE UNITED STATES OF AMERICA

81 82 83 84 9 8 7 6 5 4 3 2 1

PREFACE

This book deals with computer arithmetic and treats it in a more general sense than usual. This results in more far-reaching concepts and a more uniform theory. These in turn serve to simplify even the restricted traditional notions of computer arithmetic. They also clarify and otherwise facilitate the process of implementation of computer arithmetic. The text summarizes both an extensive research activity that went on during the past fifteen years and the experience gained through various implementations of the entire arithmetic package on diverse processors, including even microprocessors. These implementations require an extension of existing higher programming languages to accommodate the entire arithmetic package. The text is also based on lectures held at the Universität Karlsruhe during the preceding decade.

While the collection of research articles that contribute to this text is not too large in number, we refrain from a detailed review of them and refer to the list of references. Since our text synthesizes and organizes diverse contributions from these sources into a coherent and global setting, we found it best not to interrupt the continuous flow of exposition with detailed citations to original sources.

The text consists of two parts. Part 1, comprising the first four chapters, deals with the theory of computer arithmetic, while Part 2, comprising the last three chapters, treats the implementation of arithmetic on computers. Our development shows that a sound implementation of arithmetic on computers depends critically on the theoretical development. Such an implementation requires the establishment of various isomorphisms between different definitions of arithmetic operations. These isomorphisms are to be established for the spaces in which the actual computer operations reside. Therefore we begin with a development of these spaces.

Central to this treatise is the concept of semimorphism. Properties of this concept are directly derivable from the diverse models that we study here. The concept of semimorphism supplies an ordering principle that permits the whole treatise to be understood and developed in close analogy to principles of geometry, whereby the mapping properties of a semimorphism replace those of a group of transformations in a geometry. Following this idea, we define the structures of the spaces occurring in computations on computers as invariants with respect to semimorphisms. The algorithms for the arithmetic operations in various spaces, which are given in Part 2, describe the implementation of these semimorphisms on computers. The result is an arithmetic with many sound and desirable properties (optimal accuracy, theoretical describability, closedness of the theory, applicability, etc.). The similarity to geometric principles just referred to is the guideline that leads—as we see it—to this closed and well-rounded presentation of computer arithmetic.

Implementation of the algorithms is made along natural lines with which the nonspecialist views computers. This principle helps to avoid those misunderstandings of the computer by its users that are caused by tricky implementations.

We mention incidentally that a sound implementation of traditional computer arithmetic is captured by what we call the vertical definition of arithmetic.

This book, of course, can be used as a textbook for lectures on the subject of computer arithmetic. If one is interested only in the more practical aspects of implementation of arithmetic on computers, Part 2 along with acceptance, a priori, of some results of Part 1, also suffices as a textbook for lectures.

We regard the availability of the directed roundings and interval operations as essential tools on the computer. Apart from the great impact on the insight and theoretical understanding of computer arithmetic exerted by the development of interval arithmetic, these tools are necessary if a computer is to be used to control rounding errors. Employment of such techniques provides computation with guarantees, and thus permits use of the computer for verification and *decidability*. Such questions cannot be studied by using arithmetic which rounds in the traditional sense. By taking appropriate measures, interval operations can be made as fast as the corresponding ordinary floating-point operations.

The theory developed in Part 1 permits additional applications to numerical analysis, applications that are beyond the scope of this book. We mention a few of them:

A clear and precise definition of all arithmetic operations on a computer is a fundamental and necessary condition for reliable error analysis of numerical algorithms. The definition of the arithmetic operations on a computer by semimorphisms is also sufficient for complete control of the computational error due to the approximate arithmetic performed by the computer for many standard problems of numerical analysis. Examples of such problems are linear systems of equations, inversion of matrices, eigenvalue problems, zeros of polynomials, nonlinear systems of equations, evaluation of arithmetic expressions and mathematical functions, linear programming problems, numerical quadrature, and initial and boundary value problems for ordinary differential equations.

The concept of semimorphism can be used directly for an axiomatic definition of arithmetic within the syntax and semantics of programming languages.

This in turn should be useful for correctness proofs of numerical programs.

The similarity of the concept of a semimorphism to the Kuratowski axioms of a topology is clear. This has been used to redefine the concept of stability of algorithms in numerical mathematics [1, 76, 82, 83].

The concept of semimorphism can be applied to study and describe the approximation of function spaces by certain subspaces.

The spaces that we later call ringoids and vectoids and in which computations on computers are performed provide a framework for the study of the cyclic termination of iterative methods on computers [28, 29, 67, 86].

Last, but hardly least, we note that the spaces of interval arithmetic can be developed much further. It is well known that the intervals over $\boldsymbol{R}$ and C form regular semigroups with respect to addition and multiplication. Such semigroups can be imbedded into groups by a well-known algebraic procedure. The extended spaces then permit the definition of concepts of metric and norm in a manner quite similar to their definition in a field or a linear space. This greatly simplifies the analysis of interval algorithms [1, 24, 25, 26, 80].

Many of these additional applications are subjects of current research, and we defer their exposition to a follow-up volume.

We are grateful to all those who have contributed through their research to this treatment. We once more refer to the bibliography at the end of this text. We especially owe thanks to Dr. P. Schweitzer of IBM Germany and to Dr. R. A. Toupin of the IBM Research Center. Their support and their interest, so congenial in its nature, have been critical for the completion of this text. We are also grateful to J. Genzano. Her virtuosity and devotion to the physical preparation of the text provided a constant force accelerating the work.

CONTENTS

INTRODUCTION AND PRELIMINARY DEFINITION OF COMPUTER ARITHMETIC

In this treatise we present a study of computer arithmetic. Central to this is the specification of the spaces that form the natural setting for arithmetic when it is performed in a computer. In earlier days of computer development the properties of computer arithmetic were influenced by such features as simplification of computer architecture, hardware and software considerations, and even the cost of the technology employed. Experience has shown that these influences led to economies that frequently resulted in added costs and burdens to the user. A typical example of the latter is the frequent production of an inexplicable, even incorrect computation. The arithmetical properties of computer operations should be precisely specified by mathematical methods. These properties should be simple to describe and, we hope, easily understood. This would enable both designers and users of computers to work with a more complete knowledge of the computational process.

To L. Kronecker we owe the remark, "*God created the natural numbers, all else is man made.*" This difference is reflected in computers, where arithmetic on the natural numbers may be performed exactly, while for all else, only approximately. In fact, there are some interesting mathematical questions involved even in the implementation of integer arithmetic on computers. However, we postpone discussing these and assume that the hardware designer has satisfactorily solved this particular problem.

In addition to the integers, numerical algorithms are usually defined in the space $\boldsymbol{R}$ of real numbers and the vectors $\boldsymbol{VR}$ or matrices $\boldsymbol{MR}$ over the real numbers. Additionally, the corresponding complex spaces $\boldsymbol{C}$, $\boldsymbol{VC}$, and $\boldsymbol{MC}$

also occur ocassionally. All these spaces are ordered with respect to the order relation $\leq$. (In all product sets the order relation $\leq$ is defined componentwise.) Recently, numerical analysts have begun to define and study algorithms for intervals defined over these spaces. If we denote the set of intervals over an ordered set $\{M, \leq\}$ by IM, we obtain the spaces $\boldsymbol{IR}$, $\boldsymbol{IVR}$, $\boldsymbol{IMR}$, and $\boldsymbol{IC}$, $\boldsymbol{IVC}$, $\boldsymbol{IMC}$.

In Fig. 1, to which we shall repeatedly refer, we present a table of spaces and operations. For example, in the second column of this figure we list the various spaces in which arithmetic is performed and which we have introduced in the previous paragraph.

	I		II		III		IV	V
1			$\boldsymbol{R}$	$\supset$	D	$\supset$	S	$+ - \cdot /$
								$\times$
2			$\boldsymbol{VR}$	$\supset$	VD	$\supset$	VS	$+ -$
								$\times$
3			$\boldsymbol{MR}$	$\supset$	MD	$\supset$	MS	$+ - \cdot$
4	$\boldsymbol{PR}$	$\supset$	$\boldsymbol{IR}$	$\supset$	ID	$\supset$	IS	$+ - \cdot /$
								$\times$
5	$\boldsymbol{PVR}$	$\supset$	$\boldsymbol{IVR}$	$\supset$	IVD	$\supset$	IVS	$+ -$
								$\times$
6	$\boldsymbol{PMR}$	$\supset$	$\boldsymbol{IMR}$	$\supset$	IMD	$\supset$	IMS	$+ - \cdot$
7			$\boldsymbol{C}$	$\supset$	CD	$\supset$	CS	$+ - \cdot /$
								$\times$
8			VC	$\supset$	VCD	$\supset$	VCS	$+ -$
								$\times$
9			MC	$\supset$	MCD	$\supset$	MCS	$+ - \cdot$
10	PC	$\supset$	IC	$\supset$	ICD	$\supset$	ICS	$+ - \cdot /$
								$\times$
11	PVC	$\supset$	IVC	$\supset$	$IVCD$	$\supset$	$IVCS$	$+ -$
								$\times$
12	PMC	$\supset$	IMC	$\supset$	$IMCD$	$\supset$	$IMCS$	$+ - \cdot$

FIGURE 1. Table of the spaces occurring in numerical computations.

For arithmetic purposes, a real number is usually represented by an infinite b-adic expansion, with operations performed on these expansions defined as the limit of the sequence of results obtained by operating on finite portions of the expansions. In principle, a computer could approximate this limiting process, but the apparent inefficiency of such an approach eliminates its serious implementation even on the fastest computers. In fact, for arithmetic purposes, the real numbers are approximated by a subset S in which all operations are simple and rapidly performable. The most common choice for this subset S is the so-called floating-point system with a fixed number of digits in the mantissa. If a prescribed accuracy for the computation cannot be achieved by operating within S, we use a larger subset D of $\boldsymbol{R}$ with the property $\boldsymbol{R} \supset D \supset S$. For arithmetic purposes we define vectors, matrices, intervals, and so forth as well as the corresponding complexifications over S and D. So doing, we obtain the spaces VS, MS, IS, IVS, IMS, CS, VCS, MCS, ICS, $IVCS$, $IMCS$ and the corresponding spaces over D. These two collections of spaces are listed in the third and fourth columns of Fig. 1. For example, CS is the set of all pairs of elements of S, VCS the set of all n-tuples of such pairs, ICS the set of all intervals over the ordered set $\{CS, \leq\}$, and so forth.

Characteristically, S and D are chosen as the sets of floating-point numbers of single and double length, respectively. However, in Fig. 1, S and D are generic symbols for a whole system of subsets of $\boldsymbol{R}$ with arithmetic properties that we shall subsequently define.

Having defined the sets listed in the third and fourth columns of Fig. 1, we turn to the question of defining operations within these sets. These operations are supposed to approximate operations that are defined on the corresponding sets listed in the second column of Fig. 1. Figure 1 has four blocks, and in every set in each of the first lines of each block, we are to define an addition, a subtraction, a multiplication, and a division. For the sets in the last line of each such block, for instance, we need to define an addition, a subtraction, and a multiplication. These required operations are listed in the fifth column of Fig. 1. Furthermore, the lines in Fig. 1 are not mutually independent arithmetically. By this we mean, for instance, that a vector can be multiplied by a scalar as well as by a matrix; an interval vector can be multiplied by an interval as well as by an interval matrix. These latter multiplication types are indicated in Fig. 1 by means of a $\times$ sign between lines in the fifth column therein.

As a preliminary and informal definition of computer arithmetic, we say the following: *By computer arithmetic, we understand all operations that have to be defined in all of the sets listed in the third and fourth columns of Fig. 1 as well as in certain combinations of these sets.* The sets S and D may, for instance, be thought of as floating-point numbers of single and double mantissa length. In a good programming system, these operations should be available as operators for all admissible combinations of data types.

We interpret this definition in somewhat more detail. First we make a count of the number of multiplications that occur in the computer arithmetic as defined above. Later on we make the analogous count for the other operations. But the multiplication count itself is enough to show that it is too much to expect the average computer user to define the system of operations by himself.

If Z denotes the set of integers on the computer, we have the *five basic data types*: Z, S, CS, IS, ICS, which in an appropriate programming language may be called integer, real, complex, real interval, and complex interval. If a and b are operands, each a possible one of these five data types, the table in Fig. 2 shows the resulting type of the product $a * b$.

$a * b$	Z	S	CS	IS	ICS
Z	Z	S	CS	IS	ICS
S	S	S	CS	—	—
CS	CS	CS	CS	—	—
IS	IS	—	—	IS	ICS
ICS	ICS	—	—	ICS	ICS

FIGURE 2. Multiplication table for the basic numerical data types $a * b$.

A dash in the table of Fig. 2 means that the product $a * b$ is not defined a priori. Indeed a floating-point number is an approximate representation of a real, while an interval is a precisely defined object. The product of the two, which ought to be an interval, may then not be precisely specified. If the user is obliged to multiply a floating-point number and a floating-point interval, he has to employ a transfer function that transforms this floating-point operand into a floating-point interval. (This implicitly endows a precision to the floating-point number.) At this point ordinary multiplication of floating-point intervals may be employed.

If the programming language used has a so-called strong typing concept, the table in Fig. 2 shows that for multiplication among pairs of the five basic data types, 17 multiplication routines are required. We shall see that the multiplication corresponding to the entries in the framed part of the table of Fig. 2 must be supplied with three different roundings if the multiplications corresponding to interval types are to be correctly defined. This requires 16 additional multiplications or a total of 33 multiplication routines for the five basic data types.

Now let T_1, T_2, and T_3 denote one of the basic data types Z, S, CS, IS, ICS. We consider the sets of matrices MT_i, vectors VT_i, and transposed vectors $V^T T_i$, $i = 1(1)3$, whose components are chosen from among the basic data types. Elements of these sets can also be multiplied. Figure 3 displays the types of such products. In the figure, T_3 is to be replaced by

$a \cdot b$	T_2	MT_2	VT_2	V^TT_2
T_1	T_3	MT_3	VT_3	—
MT_1	—	MT_3	VT_3	—
VT_1	—	—	—	MT_3
V^TT_1	—	V^TT_3	T_3	—

FIGURE 3. Table for the multiplication $a \cdot b$ of matrices and vectors over the basic data types.

the resulting type of Fig. 2 if the components of the operands are of the type T_1 and T_2, respectively. The products with operands of type T_1 and T_2 were already counted (in relation to Fig. 2). In addition to these products, we see seven essential {matrix, vector} multiplications[†] in Fig. 3. This leads in principle to $33 \times 8 = 264$ different multiplications.

The table for the addition and subtraction of the basic data types is identical to that of Fig. 2. The division table is likewise identical except for the single entry that corresponds to the quotient of two operands of type Z. Thus for each of the three cases of addition, subtraction, and division of the five basic data types, we require, as before, 33 different routines.

The table for matrix and vector addition and subtraction is given in Fig. 4.

$+\ -$	T_2	MT_2	VT_2	V^TT_2
T_1	T_3	—	—	—
MT_1	—	MT_3	—	—
VT_1	—	—	VT_3	—
V^TT_1	—	—	—	—

FIGURE 4. Addition and subtraction table of matrices and vectors over the basic data types.

Summarizing, we can say that we have 99 ($= 3 \times 33$) different additions and subtractions, 264 different multiplications, and 33 different divisions.

When we deal with interval spaces, we are obliged to append to the arithmetic operations the operations of intersection ($\cap$) and taking the

[†] A subtle point concerns the occurrence of the set IMS in Fig. 1, while for $T_i = IS$ in Fig. 3 the set MIS is listed. Later on we shall prove that they are isomorphic with respect to the order relation $\leq$ and all arithmetic operations.

convex hull ($\bar{\cup}$) of pairs of intervals. Figure 5, which lists the different possibilities, shows that there are $12 \, (= 3 \times 4)$ different intersections and $12 \, (= 3 \times 4)$ different takings of the convex hull. In Fig. 5, T_1 resp. T_2 denote *IS* or *ICS*. The intersection and convex hull are to be taken componentwise.

$\cap \; \bar{\cup}$	*IS*	*ICS*
IS	*IS*	*ICS*
ICS	*ICS*	*ICS*

$\cap \; \bar{\cup}$	T_2	MT_2	VT_2	$V^{\mathrm{T}}T_2$
T_1	T_3	—	—	—
MT_1	—	MT_3	—	—
VT_1	—	—	VT_3	—
$V^{\mathrm{T}}T_1$	—	—	—	—

FIGURE 5. Tables for intersection and convex hull.

The subject of the first part of this text concerns the development of definitions for these many operations and also specifies simple natural structures that form the settings of these operations. The second part then deals with the implementation of these operations on computers. In particular, we shall see that the large number of operations that we counted above can be reduced and built up from a relatively small number of fundamental algorithms and routines.

We shall see that there are in principle two different basic methods for defining computer arithmetic. These are called the vertical method and the horizontal method. For the horizontal method, the arithmetic is assumed to be known in the leftmost set of each row of Fig. 1. On the other hand, for the vertical method, the arithmetic is assumed to be defined by some means in each set of the first row of Fig. 1. Both methods then define the arithmetic in all of the sets in Fig. 1 by appropriate extension procedures relevant to each method. By way of illustration of these two possibilities, we consider a simple detail of Fig. 1 that concerns the sets ***R***, *D*, and *S* as well as the spaces of matrices ***MR***, *MD*, and *MS*. We assume that an arithmetic in *D* and *S* is defined. By the vertical definition of the arithmetic in *MD* and *MS*, we mean that the operations in *MD* and *MS* are defined by the operations in *D* and *S* and the usual formulas for the addition and multiplication of real matrices (see the following figure). On most computers this is precisely the method of definition of addition and multiplication for floating-point matrices. While the horizontal method to be developed has many advantages, it turns out that both methods lead to the same abstract structures as settings for the arithmetics. It will develop that the