293
10/17/68
LA-3979

488

MASTER

LOS ALAMOS SCIENTIFIC LABORATORY
of the
University of California
LOS ALAMOS • NEW MEXICO

The Simulation of Two Arithmetic Structures

UNITED STATES
ATOMIC ENERGY COMMISSION
CONTRACT W-7405-ENG. 36

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

This report expresses the opinions of the author or authors and does not necessarily reflect the opinions or views of the Los Alamos Scientific Laboratory.

Printed in the United States of America. Available from
Clearinghouse for Federal Scientific and Technical Information
National Bureau of Standards, U. S. Department of Commerce
Springfield, Virginia 22151

Price: Printed Copy \$3.00; Microfiche \$0.65

LA-3979
UC-32, MATHEMATICS
AND COMPUTERS
TID-4500

LOS ALAMOS SCIENTIFIC LABORATORY
of the
University of California
LOS ALAMOS • NEW MEXICO

Report written: March 1968

Report distributed: October 8, 1968

The Simulation of Two Arithmetic Structures

by

R. C. Blandford
N. Metropolis

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liability with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

THE SIMULATION OF TWO ARITHMETIC STRUCTURES

by

R. C. Blandford and N. Metropolis

ABSTRACT

Two classes of arithmetic are incorporated, by programming, into a computer. The first, significant-digit arithmetic, provides for the monitoring of precision in a computation. The second, specified-point arithmetic, is a generalization of the conventional fixed-point type. The implementation is discussed. As an application of the first type, a new technique is described for function evaluation based on polynomial approximation.

1. INTRODUCTION

Conventional, normalized floating-point arithmetic is not very convenient for studies in error analysis; more specifically, for monitoring the effects due to either inherent errors or cancellation. More suitable for these purposes is a type known as "significant-digit" arithmetic; this form has been described earlier [1]. The second type is a generalization of conventional fixed-point arithmetic called specified-point arithmetic. Both of these arithmetic structures have been incorporated by one of us in the logical design of MANIAC III at the University of Chicago. The purpose of the present effort has been to simulate these arithmetic forms on an existing computer, namely, MANIAC II in our Laboratory. Moreover, a new number type has been introduced into our programming language MADCAP [2] so that arithmetic on numbers thus declared in a program is performed in significant-digit form. For specified-point operations, recourse is made to procedural statements (to be described below); the representation of numbers, however, is the same in the two cases.

A modest number of input and output routines, together with a library of elementary functions, has been completed; this has made it convenient to

pursue some problems in error analysis [3]. Finally, we report on a new result utilizing these facilities that is concerned with function evaluation based on polynomial approximation.

2. SIGNIFICANT-DIGIT ARITHMETIC

Notation and adjustment rules.

Let a number be represented by $x = 2^e \cdot f = (e, f)$, where e is an integer exponent and f a fractional part such that $|f| < 1$. The transformation $(e, f) \rightarrow (e+a, 2^{-a}f)$ for integer a is called adjustment by a ; for $a > 0$, $2^{-a}f$ is rounded; for $a < 0$, $2^{-a}|f| < 1$ must, of course, be satisfied. Define m by $\frac{1}{2} \leq 2^m |f| < 1$; i.e., m is a nonnegative integer specifying the number of leading zeros in $|f|$.

The adjustment rules for the four arithmetic operations (+, -, x, /) for significant-digit arithmetic may be summarized as follows: let

$$(e_3, f_3) = (e_1, f_1) * (e_2, f_2), \quad (1)$$

where $*$ indicates one of the four operations; an m_i is associated with each (e_i, f_i) . For +, -, the rule is simply

$$e_3 = \max(e_1, e_2); \quad (2)$$

for x , $+$, the adjustment is such that

$$m_3 = \max(m_1, m_2). \quad (3)$$

Eq. (2) corresponds to the usual scheme for so-called normalized addition and subtraction except that the usual postnormalization step is omitted; Eq. (3) states that the number of meaningful bits in the result is equal to the smaller of the corresponding numbers in the two input fractions. A more complete discussion may be found in [4].

Implementation.

On MANIAC II, the exponent e_M is represented with four bits, and is such that the correspondence with e defined above is $e = e_M \cdot 16$. This unit is too coarse for the representation of the meaningful bits of f . A single word for each number might still have been used--but with a different allocation of bits for exponent and fractional part to accommodate the smaller, binary base. It was, however, more convenient to use separate words for each part. One consideration was that a new (single-word) format for the representation of numbers implied in any case a separation of the two parts in the course of simulating the desired arithmetic operations. With the two-word allocation, the range of exponents could be very large indeed; however, it seemed reasonable to restrict the range to $-255 < e < 255$. Actually it was useful to bias the exponent representation by 2^8 and to reserve $e = 0$ for a special symbol discussed below. A very convenient instruction, used frequently in the present implementation of the new arithmetic operations, normalizes a number, i.e., adjusts it by negative m , and places m in a prescribed index register.

Addition (subtraction) is effected by first comparing the relevant exponents and then adjusting by the (nonnegative) difference, the operand with the smaller exponent; adding (subtracting) with roundoff of the fractional parts is then performed. It is useful to examine whether the exponent difference is in range, and if not, to bypass the adjustment of the one number and take the other directly as the result.

Multiplication (division) is accomplished by first normalizing the two operands utilizing the instruction mentioned above; this produces m_1 and m_2 ; multiplication (division) is then performed by fixed-point instructions and the normalized result is adjusted by $\max(m_1, m_2)$ and then rounded.

As mentioned above, we use $e = 0$ to represent "true zero," symbol $0 = (0, 0)$, with the following properties:

$$\left. \begin{aligned} x \pm 0 &= 0 \pm x = x \\ x \cdot 0 &= 0 \cdot x = 0 \\ 0 + x &= 0; \quad x \div 0 \text{ undefined}^\dagger \end{aligned} \right\} \quad (4)$$

Thus, it is useful to detect (by simple inspection of e) whether either operand is 0 and accordingly proceed directly to the appropriate result specified by Eq. (4). Finally, inspection for exponent overflow or underflow (if possible) is made in the non-zero case for all four operations, and interrupt is caused if detected.

Characteristics.

The programs for these arithmetic operations have been prepared as subroutines. Characteristics, such as the number of instructions in a typical, nonzero case, are listed in Table II. Speed ratios are somewhat arbitrary in nature but are included for the sake of completeness. Such ratios depend upon the detailed nature of the vocabulary of the computer at hand, and on the precise description of the set of operations being compared. We have taken the following as a basis for comparison: two numbers in the appropriate representation are in the main store, the given operation is performed, and the result is stored. The comparison is made with the standard floating-point operations. It is evident that the speed ratios are such that research studies utilizing this type of arithmetic can be pursued conveniently; for daily, routine computations, the simulated form should be replaced by actual hardware in order to achieve speeds comparable to those for more conventional types of arithmetic.

[†]Computer is interrupted.

TABLE I

Characteristics of the programs simulating the two types of arithmetic structures. The values of the various entries depend, of course, on the detailed logical structure of the computer at hand, but this dependence is probably not a sensitive one. That the ratios for addition and multiplication are comparable reflects the fact that MANIAC II has a completely parallel multiplier with operation times similar to that of the adder. On the other hand, division is relatively slow, thus a low ratio is to be expected.

	Significant-digit arithmetic	Specified-point arithmetic
Addition (subtraction)		
N	44	27
T	68	68
S	11.6	7.2
Multiplication		
N	54	27
T	66	44
S	16.9	10.7
Division		
N	74	45
T	95	67
S	3.6	2.7

N is the number of computer operations executed in a typical example.

T is the total number of instructions in the respective programs.

S is the ratio of execution time of program compared with that of conventional floating-point instructions.

Systems consideration.

Our compiler, MADCAP, permits several number types to be a part of its language; e.g., provision is made for large integers, double-precision real numbers, and matrices, in addition to the standard real numbers. Once a number type is introduced, the corresponding arithmetic is implied in any subsequent algebraic operation. We are indebted to our colleagues, M. Wells and D. Bradford, for the inclusion of the new type--significant-digit numbers--in the MADCAP system [2]. It has enabled several investigators to pursue numerical experiments with great simplicity of preparation.

3. SPECIFIED-POINT ARITHMETIC

Description.

Modern computers, possessing conventional floating-point operations, usually have included some version of a fixed-point type of arithmetic. The latter form may be described in the notation

of Sec. 2 by stating $e_1 = 0$ for all computer numbers x_i . Frequently, this zero value is by implication; i.e., in this number representation, all bits are used for the fractional part. The resulting dichotomy of representations has some awkwardness for the designer, the engineer, the systems programmer, and the user. Some time ago, it occurred to one of us that a single representation would be a useful simplification; moreover, the exponent could be used in a new type of arithmetic that is a generalization of the conventional fixed-point arithmetic in the following sense. Instead of being restricted to having the binary point on the extreme left of all numbers, i.e., $|x_i| < 1$, the programmer may select at his option the position of the binary point for the representation of his numbers and have the arithmetic performed in a manner that maintains this position. For example, the problem at hand may be one of (i) integers, (ii) fractions exclusively, or (iii) a mixed type.

Clearly, it would be convenient if the binary point was fixed at the extreme right of the word in (i), at the extreme left for (ii), and at some appropriate intermediate position for (iii). (It will be seen that the choice is an even wider one.)

Adjustment rule.

Consider the ordered pair of operands in Eq.

(1). A single adjustment obtains for all four operations in specified-point arithmetic; namely,

$$e_3 = e_1. \quad (5)$$

Usually, $e_1 = e_2$; if not, it is the exponent of the anterior operand that dictates the exponent of the result. Thus if $e_1 \neq e_2$, the representation of $(x_1 + x_2)$ differs from that of $(x_2 + x_1)$; similarly, for multiplication.

As with conventional fixed-point arithmetic, the fractional part of the result f_3 may, under addition and division, overflow here; moreover it should be noted that multiplication can also lead to an overflow condition. Two reactions are available when overflow occurs: either ignore if anticipated or interrupt.

Implementation.

For addition or subtraction, adjust x_2 by $(e_1 - e_2)$, then combine the fractional parts. Symbolically,

$$\begin{aligned} e_1 &\rightarrow e_3 \\ f_1 \pm 2^{-(e_1 - e_2)} f_2 &\rightarrow f_3. \end{aligned} \quad (6)$$

Care must be taken if one of the terms is "true zero" = (0,0). Strict adherence to Eq. (6) requires that we adopt

$$\begin{aligned} (e_1, f_1) \pm (0, 0) &\rightarrow (e_1, f_1) \\ (0, 0) \pm (e_2, f_2) &\rightarrow \text{fractional overflow.} \end{aligned} \quad (7)$$

The latter stems from the fact that adjusting any nontrue zero posterior operand to the exponent of "true zero" implies an "infinite" left shift of f_2 .

For multiplication, the appropriate steps are

$$\begin{aligned} e_1 &\rightarrow e_3 \\ 2^{e_2} f_1 \cdot f_2 &\rightarrow f_3; \end{aligned} \quad (8)$$

where $f_1 \cdot f_2$ is a double-precision product, which is left- or right-shifted by $|e_2|$ according as $e_2 > 0$ or $e_2 < 0$, unchanged if $e_2 = 0$. Again care must be exercised if one factor is (0,0): the consistent interpretation is

$$\begin{aligned} (0, 0) \cdot (e_2, f_2) &= (0, 0) \\ (e_1, f_1) \cdot (0, 0) &= (e_1, 0). \end{aligned} \quad (9)$$

For division, it turns out to be expedient to first normalize the divisor; call it (e_{2N}, f_{2N}) . Then adjust the dividend by e_{2N} as a double-precision quantity; divide it by f_{2N} , to obtain f_3 . Symbolically,

$$\begin{aligned} e_1 &\rightarrow e_3 \\ 2^{-e_{2N}} f_1 + f_{2N} &\rightarrow f_3. \end{aligned} \quad (10)$$

The special cases are

$$\begin{aligned} (e_1, f_1) + (0, 0) &\rightarrow \text{undefined,} \\ (0, 0) + (e_2, f_2) &\rightarrow (0, 0). \end{aligned} \quad (11)$$

Table I includes the operational characteristics of these arithmetic instructions. As before, speed comparisons are made with conventional floating-point operations.

It is advantageous to have the representation of numbers for this arithmetic identical with that for significant-digit arithmetic; among other reasons, it is useful to be able to use the same numbers for either of the two sets of arithmetic. On the other hand, in the MADCAP system, the number type dictates the class of arithmetic to be used in an operation. Rather than introduce a separate number type for specified-point operations, it seemed expedient to effect the assimilation of these operations by MADCAP as procedures, rather than as subroutines coded in machine language. This essentially means that the programmer cannot use the conventional forms for algebraic equations, but must make

explicit reference to the individual procedures, but this is fairly straightforward. If a computer has these specified-point instructions as part of its vocabulary, it would, of course, be expedient to have a separate set of symbols for them in the associated language.

4. LIBRARY OF ROUTINES

Convenient use of significance arithmetic relies on a set of routines that is consistent with this approach. Firstly, input numbers must be represented within the computer in a way that reflects their precision; similarly, output numbers should also convey their associated precision. Secondly, the subroutines for computing simple functions must, of course, take account of the magnitude and precision of the input variables (and parameters) in order to establish the precision of the function values. Moreover, the precision achieved should be maximal, in the sense that another formulation of subroutine would not lead to a greater number of meaningful bits.

Input-output.

For input-output routines, we adopt the method of radix conversion of variable-precision numbers suggested by Metropolis and Ashenurst [5]. Briefly, input decimal numbers are indicated with or without an associated exponent, with the decimal point always indicated (even for imprecise integer values) such that every decimal digit of a number is interpreted as meaningful; e.g., $47. + 4$, $-.85192 - 3$, 12.76 , where, as usual, the second of the pair (if present) is the associated exponent base ten.

Similarly, for output decimal numbers only meaningful digits are present, thus such numbers do not usually have a uniform length. Moreover, since a decimal representation has a coarser structure compared to that of binary, it is useful to append to each output (decimal) number an integer that indicates the error in the last digit of the number.

One further remark about input numbers. Integers and ratios of integers are, of course, known to maximally representable precision in a computer. It would be annoying to include a string of zeros in order to indicate this full precision; consequently we have adopted the convention that numbers

without decimal point, and ratios of such numbers, are to be converted to binary form with maximal precision.

Unless otherwise indicated, numbers converted to binary form are adjusted so that the least significant bit of their fractions resides in the rightmost stage of the word. However, the programmer may specify a preceding stage for this residence; this has the effect of reducing the effect of accumulated rounding error. (Usually this specification is consistent with the most precise input number, apart from integers or their ratios.) This notion of "guard bits" has been discussed in earlier work [4].

For specified-point arithmetic, the programmer specifies the desired position of the fixed binary point, and input numbers are adjusted to achieve this. We stress the fact that the formal representation of numbers is the same for the two kinds of arithmetic; it is only the adjustment rules for representing the results of arithmetic operations that differ.

Simple functions.

A small set of subroutines is available for both types of arithmetic. The simplest of these is square root; it exists as an explicit instruction in the present computer, and being a unary operation, the adjustment rules are straightforward; they are not discussed here.

At present the library includes logarithmic, exponential, trigonometric, and inverse trigonometric functions. These have been evaluated by means of power series; the number of terms used depends upon the magnitude and precision of the dependent variable. Moreover, the function value is represented in the computer with a precision that is consistent with that of its argument. In the preparation of these subroutines, a technique was developed that is perhaps of more general interest; we discuss this in the next paragraph.

Series evaluation.

Whenever a function is represented by a simple

power-series approximation $f(x) = \sum_{i=0}^n a_i x^i$, the

question arises as to how many terms should be used in a particular application. Often a conservative

estimate is made, based on worst case of argument and for fully representable precision. Frequently, many fewer terms would suffice--not only because of the particular value of the argument, but also because of low precision in that value. We determine, as a first step, the highest term consistent with a particular argument; with this information we can then use efficient recursive techniques, such as Horner's scheme [6], instead of computing successive terms and examining their magnitudes.

In the preparation of our library of subroutines, it became evident that a good estimate of the highest term in a series expansion that need be considered is readily available if that part, call it x , of the argument that participates directly in the expansion satisfies $|x| < \frac{1}{2}$, i.e., $|x|$ has at least one leading zero in binary form. We discuss below how this condition is achieved; for the present, assume it is satisfied, and let

p = maximum number of available bits for fraction

g = number of guard bits (programmer's option)

z be determined by $\frac{1}{2} \leq 2^z |x| < 1$.

Note that $z \geq 1$ because $|x| < \frac{1}{2}$, and is given by the negative of the exponent of normalized $|x|$.

As a first approximation, say n_1 , to the highest power that need be considered, we can write

$$n_1 = \left\lceil \frac{p-g}{z} \right\rceil, \quad (12)$$

since $p-g$ is the number of meaningful bits, and each power of x introduces at least z zeros. However, x may have been adjusted to reflect its precision.

Since $|x| < 1$ (necessarily), its exponent e_x in adjusted form is a direct measure of that precision. (Note that the exponent may be negative.) Thus, a second approximation n_2 that takes into account the precision of x is given by

$$n_2 = \left\lceil \frac{p-q-e_x}{z} \right\rceil. \quad (13)$$

A final estimate n considers the effect of the coefficients in the power series. Since they are usually less than unity, n_2 tends to be an overestimate. In any case, a simple approach is to once prepare an easily derived table of n vs n_2 for each series.

Techniques for polynomial evaluation that provide a measure of precision in the result have been studied recently [2]. Application shows that, for all the series that occur in our set of subroutines, the simple Horner scheme is adequate. This follows primarily from the fact that the coefficients in the series decrease monotonically and have maximal precision; in addition the domain of x is quite small, and the magnitude of the first derivative--a measure of correlation error--in this domain is also small and monotone.

We return to the question of the transformation of the independent variable, say y , namely, $x = T(y)$, such that $|x| < \frac{1}{2}$ is satisfied. We consider first the natural exponent function, $\exp y$, (in binary form): let $y/\ln 2 = w = I + f$, where I, f are the integer and fractional parts respectively of w , such that $0 \leq f < 1$. If $f < \frac{1}{2}$, then define $x' = f$; if $f \geq \frac{1}{2}$, define $x' = f - 1$ and increase I by unity. Thus, $\exp y = 2^w = 2^I \cdot 2^{x'} = 2^I \cdot \exp(x' \ln 2) = 2^I \cdot \exp x$, where $\exp x$ is approximated by $\sum_{k=0}^n x^k/k!$. It is clear that the condition $|x| < \frac{1}{2}$ is satisfied.

For $\ln y$: one first writes $y = w \cdot 2^e$, where $\frac{1}{2} \leq w < 1$ and e is the corresponding integer exponent. Hence $\ln y = \ln w + e \ln 2$ and $\ln w =$

$\ln(1+x) = \sum_{i=1}^n (-1)^{i-1} x^i/i$. It is however more effi-

cient to consider the following: if $w \geq \frac{2}{3}$, proceed as above; if $w < \frac{2}{3}$, then $2w - 1 \rightarrow x$ and $e - 1 \rightarrow e$. Again $|x|$ has at least one leading zero.

For the trigonometric functions, the angle, say y , is easily reduced so that $0 < y \leq \frac{\pi}{4}$. A further reduction is achieved by $y = \frac{k}{8} \rightarrow x$ for $k = 0, 1, \dots, 6$ so that $x < \frac{1}{8}$. Thus z is at least 3 in Eq. (12).

After the polynomial in x has been evaluated, trigonometric relations for the sum of two angles are used. (The square root operations that are implied in some of these relations are directly executed as machine instructions in MANIAC II.)

For the inverse trigonometric functions: again, preliminary transformations can be found to reduce the magnitude of the argument. We discuss only $\eta = \sin^{-1} y$. Let $\alpha = \sin^{-1} a$ where a is to be speci-

fied below. Then from elementary considerations,

$$\eta = \sin^{-1} \left\{ y\sqrt{1-a^2} - a\sqrt{1-y^2} \right\} + \alpha = \sin^{-1} x + \alpha. \quad (14)$$

y is easily restricted to nonnegative values; if

$y > \sqrt{2}/2$, then $\sqrt{1-y^2} \rightarrow y$; thus $y \leq \sqrt{2}/2$ can be

assumed. By inspection, if $a = \frac{1}{2}$, then for

$.3 \leq y \leq \sqrt{2}/2$, $|x| \leq \frac{1}{4}$; if $y < .3$, then

$y \rightarrow x$ ($a = \alpha = 0$). Finally, use Taylor expansion

for $\sin^{-1}x$ and evaluate Eq. (14). If $\sqrt{1-y^2} \rightarrow y$ has been used, then form the complement of η .

It should be remarked that the subroutines in the library have been formulated so that the precision of the function value reflects that of the argument and is immediately discernible by the computer representation of the function value [3].

REFERENCES

1. Metropolis, N., and Ashenhurst, R. L., "Significant Digit Computer Arithmetic," IRE Trans. Electron. Computers EC-7 (1958) 265.
2. Wells, M., MADCAP Manual. Los Alamos Scientific Laboratory, Group T-7, 1964.
3. Menzel, M., and Metropolis, N., "Algorithms in Unnormalized Arithmetic. II. Unrestricted Polynomial Evaluation," Num. Math. 10 (1967) 451-462.
Metropolis, N., "Algorithms in Unnormalized Arithmetic," Proc. Colloq. Int. Centre Nat. Rech. Sci. Besançon, France (1967).
Fraser, M., and Metropolis, N., Matrix Inversion, to be published.
4. Ashenhurst, R. L., and Metropolis, N., "Unnormalized Floating Point Arithmetic," J. Assoc. Comp. Mach. 6 (1959) 415.
5. Metropolis, N., and Ashenhurst, R. L., "Radix Conversion in an Unnormalized Arithmetic System," Math. of Comp. XIX (1965) 435.
6. Henrici, P., Elements of Numerical Analysis, Wiley and Sons, Inc., New York, 1964.