NATIONAL BUREAU OF STANDARDS REPORT

10 563

FORTRAN PROGRAM FOR ARBITRARY PRECISION ARITHMETIC

NBS

U.S. DEPARTMENT OF COMMERCE

NATIONAL BUREAU OF STANDARDS

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Center for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

Applied Mathematics—Electricity—Heat—Mechanics—Optical Physics—Linac Radiation[2]—Nuclear Radiation[2]—Applied Radiation[2]—Quantum Electronics[3]—Electromagnetics[3]—Time and Frequency[3]—Laboratory Astrophysics[3]—Cryogenics[3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

Analytical Chemistry—Polymers—Metallurgy—Inorganic Materials—Reactor Radiation—Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute also monitors NBS engineering standards activities and provides liaison between NBS and national and international engineering standards bodies. The Institute consists of the following technical divisions and offices:

Engineering Standards Services—Weights and Measures—Flammable Fabrics—Invention and Innovation—Vehicle Systems Research—Product Evaluation Technology—Building Research—Electronic Technology—Technical Analysis—Measurement Engineering.

THE CENTER FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Center consists of the following offices and divisions:

Information Processing Standards—Computer Information—Computer Services—Systems Development—Information Processing Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world, and directs the public information activities of the Bureau. The Office consists of the following organizational units:

Office of Standard Reference Data—Office of Technical Information and Publications—Library—Office of Public Information—Office of International Relations.

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Part of the Center for Radiation Research.
[3] Located at Boulder, Colorado 80302.

NATIONAL BUREAU OF STANDARDS REPORT

NBS PROJECT
5000104

April 1, 1971

NBS REPORT
10 563

FORTRAN PROGRAM FOR ARBITRARY PRECISION ARITHMETIC

Leonard C. Maximon
National Bureau of Standards
Washington, D. C.

IMPORTANT NOTICE

NATIONAL BUREAU OF STA ıs accounting documents intended
for use within the Government. subjected to additional evaluation
and review. For this reason, the listing of this Report, either in
whole or in part, is not authori ː Office of the Director, National
Bureau of Standards, Washingto ıy the Government agency for which
the Report has been specitically opies for its own use.

NBS

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

1. Introduction

In this report we describe a program, written in Fortran, for performing the elementary arithmetic operations-- addition, subtraction, multiplication and division-- with real numbers having a fixed but essentially arbitrary number of significant digits, this number being determined by the user of the program. The program consists of a set of subroutines which perform the above arithmetic operations keeping a fixed number of digits throughout the calculation, as well as a number of "peripheral" subroutines which serve as a link to any outside program, taking input from it in the form of integer, real or double precision numbers and putting these numbers in a form suitable for input to the arithemtic subroutines, or taking the output from the arithmetic subroutines and either converting it to a double precision number for return to the outside program, or simply printing the arbitrary length number in a form that is convenient to read.

Although it is clear that many parts of these subroutines, or perhaps even an entire subroutine, could be written more efficiently in machine language, such a program would be applicable to only one particular computer. Since it was our desire to be able to use these subroutines on any of the currently available large computers, they have been written in ANSI Fortran, avoiding the use of symbols or operations that are peculiar to a particular computer.

Before describing some of the details of this program, let us say a bit about its use and interest from the point of view of a physicist or applied mathematician. Generally, a physics problem does not require more than a few decimals of accuracy in the final answer and rarely does it require more than the six or seven significant figures present in single precision real numbers in most computers. However, in arriving at the final answers, one often encounters cancellation of numbers which are very large relative to their difference, and in this case the number of significant figures in the result is diminished--there may in fact be none. One may in such cases use the double precision arithmetic of the computer, which generally provides about 16 significant figures, and for some computers up to 28 or 32,but that is, for practical purposes, the limit of what is generally available, and for many problems of physical interest that is not sufficient. The difficulty described here-- the cancellation of nearly equal numbers with consequent loss of significance is indeed inherent in all diffraction problems, and the mathematical difficulties encountered in computing the cross sections for the scattering of high energy electrons and high energy protons from nuclei provided much of the initial impetus for the development of this arbitrary precision arithmetic program. Although analytical methods may be developed for a particular problem, the most direct and generally applicable technique is simply to perform the pertinent part of the calculation (that part involving the cancellation with resultant loss of significant figures) with however may significant figures are needed in order that the final result have the desired number of significant figures.

In addition to its use in obtaining numerical solutions to physical problems with difficulties of the type just described, another application of an arbitrary precision arithmetic program is worth mentioning at this point, namely as a tool for checking the routines used in existing computers for the computation of elementary and special functions in double or higher precision. With an arbitrary precision arithmetic program one may compute the function values with a number of significant figures larger than that given by the fixed word length computer routine to be checked.

Our interest in the development of an arbitrary precision arithmetic program has not been solely in conjunction with the applications just mentioned, however. We do not consider an arbitrary precision arithmetic program to be an end in itself, but rather as a set of routines which would be called by subroutines, also in arbitrary precision, for the computation of the elementary functions and the special functions of mathematical physics. It is the development of arbitrary precision algorithms for these functions that is of interest from the standpoint of applied mathematics. For example, while the use of a combination of power series and asymptotic series may be sufficient for the computation of the values of a function to a relatively small number of places, they are quite unsatisfactory for much of the argument range if one is to compute function values to a very large, arbitrary number of places. In this case, non-linear recursive processes (such as Newton's method for calculating square roots, for example), have great advantages, and the mathematical analysis pertinent to these and other applicable procedures is of importance in itself. In a subsequent report we shall discuss some of the algorithms that have been developed and used for the computation of the elementary and special functions to arbitrary precision.

The basic elements of this program are, as we have said, the Fortran subroutines which perform the elementary arithmetic operations. Each of these subroutines has three arguments-- two input variables, the two numbers to be added, subtracted, multiplied or divided,-- and one output variable, the sum, difference, product or quotient of the two input variables. Each of the arguments is a linear array containing the digits of the fraction part[1], the exponent part and the sign of the number in question. We therefore also provide in the program a number of peripheral subroutines which serve an input-output function with respect to the subroutines performing the elementary arithmetic operations. Of these peripheral subroutines, those which serve an input function take an integer, real number or double precision number, either from the main program or from some other program, and put them in a form (the linear array just referred to) suitable for input variables to the elementary arithmetic routines. Those peripheral subroutines which perform an output function take the linear array that is the output (the sum, difference, product or dividend) of one of the elementary arithmetic routines and either convert it into a double precision number to be used in the main program or returned to some other program, or print out the linear array in a form that is easy to read. In addition there are two subroutines, PRM(NDG) and WRIT(NSG,NCH,NC1), which serve both as depositories for all the parameters and format statements (needed for the print-out just mentioned) that one might wish to change at one time or another, and to calculate and send to all the other subroutines via labeled COMMON any parameters which they need for their operation. The subroutine PRM(NDG), with a single argument specifying the number of digits with which the multilength calculation is to be performed, must be called before calling any other subroutines of the program.

In the following pages of this report we shall discuss in some detail the linear array in which numbers are stored as well as the components of the program just mentioned: The elementary arithmetic subroutines, the input-output subroutines and the executor subroutine PRM(NDG). Since there are clearly many ways to write and structure an arbitrary precision arithmetic program, our discussion will be primarily concerned with the detailed reasons and over-all point of view which led us to the features and structure specific to this program, and which we believe must, in any event, at least be considered in writing an arbitrary or multiple precision arithmetic program. The choice in writing or structuring a specific part of the program was often influenced by considerations of mathematical simplicity and over-all flexibility of the program as well as questions of computer storage requirements and computer time required for a given operation. One would of course like to minimize storage and time requirements and have nonetheless a program with flexibility, simple in its details and over-all structure. Since these attributes are sometimes mutually exclusive, one must, throughout the writing of the program, choose some compromise position between them, and the exact choice of this position is of course a subjective matter. We shall try to point out the reasons for our choices. Clearly one might write a quite different program if, for example, the question either of storage or time of operation were of critical importance, and flexibility of the program as a whole of relatively minor interest. In the present case, the desire to have a "portable" program, i.e., one which could be run with essentially no modifications on any of the modern computers which accept ANSI Fortran, determined a conscious choice in which flexibility was chosen over speed of operation. A program written in assembly language, applicable to one specific computer, could have a much shorter running time, and, depending on the circumstances in which the program is to be used, such a program might be preferable.

2. Numbers, their representation and storage

Since it is basic to all of the subroutines, we discuss first the manner in which a number is represented in this program, and the linear array in which it is stored. In this, and indeed in any computer program, we are dealing with numbers with a finite though possibly large number of digits, and there are many ways of representing such numbers. The representation of a real number, a, which corresponds most closely to the numbers which are the input to and output from each of the elementary arithmetic subroutines, is

$$a = \pm\, d \cdot M^k \qquad (1)$$

where M is a positive integer $\geqq$ 2,

k is an integer, positive, negative or zero

and $\frac{1}{M} \leq d < 1$.

Any real number, with the exception of zero, can be written in this form. With this exception, d has a decimal[2] representation

$$.i_1 i_2 i_3 \ldots$$

where the i_n are integers such that

$$0 < i_1 < M$$

and $$0 \leq i_n < M, \qquad n > 1.$$

This form of the representation of a, in which the first digit of the fraction part is non-zero, we call a normalized number. The number zero is denoted throughout this program by setting all the digits of d equal to zero, k=0, and a plus sign in front of the representation. It is thus the only number in this program for which the first digit is zero. The linear array corresponding to this representation contains, successively, the digits i_1, i_2, ... of d, the exponent, k, and the sign (stored as a +1 or a -1). The elements of the linear array are thus all integers, and all manipulations with the elements of the arrays are performed in integer arithmetic. The base, M, is denoted in the program by* NBRT, specified in the subroutine PRM(NDG) and shared with the other subroutines via the labeled COMMON/DGS/. While the value of NBRT may be changed from one calculation to the next (see discussion of PRM), it is not changed in the course of a calculation and hence need not be carried in the linear array NA(I) denoting the number a. With regard to the digits i_1, i_2, ... one could of course store them successively in the first, second, ... cells of the array NA(I). This would, however, not only be wasteful of computer storage, but, more seriously, result in excessively long computation times: The subroutines for addition, subtraction

* At the end of this report we give a list of the important parameters and their acronymic origins.

and multiplication perform these operations in much the same fashion as one does in grade school, the basic operations being addition, subtraction, multiplication, shifting and carrying. The time required for the addition of two numbers thus increases linearly with the number of digits in the array and the time required for the multiplication of two numbers increases quadratically[3] with the number of digits in the array.* The computing time required by these programs is thus reduced very considerably by storing the digits i_1, i_2, ... in groups--the first n digits in NA(1), the next n digits in NA(2), and so forth. The operations of addition, subtraction and multiplication are then between pairs of blocks of n digits. The number n is denoted by NDW in the program and the number of cells of NA(I) needed to hold all the digits is denoted by ND. The exponent k (in Eq.(1)) is stored in cell ND1=ND+1 and the sign in front of that equation is stored in cell ND2=ND+2. As in the case of the base NBRT, the parameters NDW, ND, ND1 and ND2, along with the parameters $\text{NBASE} = \text{NBRT}^{\text{NDW}}$ and NBASEP = NBASE/NBRT are given in the subroutine PRM(NDG) and shared with the other subroutines via labeled COMMON,

* There are in fact two subroutines for division in the program; for one the computation time increases linearly with the number of digits, for the other quadratically. These are discussed in some detail later in this report.

listed in each subroutine as

```
COMMON/DGS/ND,ND1,ND2,NDW,NBRT,NBASE,NBASEP          (2)
```

The parameter NBASE is thus effectively the base in which the calculation is performed: the addition of a number in a cell of NA(I) with a number in a cell of the linear array NB(J) corresponding to a number b will require a carry if their sum is equal to or greater than NBASE. The parameter NBASEP is the smallest number that can appear in NA(1) unless a is zero, in which case the first ND1 cells of NA(I) will have zeros and cell ND2 will have +1. All the parameters just listed in the COMMON/DGS/ are used frequently by all the subroutines. As we have mentioned, in order to save on memory requirements and, most especially, to reduce computation time, it would be desireable to make ND, the number of cells needed to store the digits of a number, as small as possible. This means that one should make NDW, the number of digits in each cell, as large as possible. However, we are finally limited by the fact that on a given computer there is a limit to the magnitude of an integer which may be stored in one cell. Thus, generally, for a computer operating in the binary mode and having a word length of N bits, the largest integer that can be stored is $2^{N-1} - 1$. Further, since in the multiplication subroutine in particular, and in other subroutines in general, we will want to multiply the integer in a cell of one array by the integer in some cell of another array, their product must not exceed the largest integer that can be stored. As we have already noted, the largest integer that will appear in one of the first ND cells is NBASE - 1 = $\mathrm{NBRT}^{\mathrm{NDW}} - 1$. Thus the number of digits per cell, NDW, is restricted by the condition

$$(\mathrm{NBRT}^{\mathrm{NDW}} - 1)^2 \leq 2^{N-1} - 1 .$$

The largest possible value of NDW for a few values of NBRT is indicated in the table below, for word lengths of 32, 36, 48 and 60 bits.

Table I

	Word length in bits			
NBRT	32	36	48	60
2	15	17	23	29
8	5	5	7	9
10	4	5	7	8
16	3	4	5	7

From the table above it is clear, given our previous remarks, that it is greatly advantageous to use the present program on a computer with a longer word length. Not only are the storage requirements reduced, but the running time on a machine with a 60 bit word length will be approximately half of that on a machine with a 32 or 36 bit word length for the addition and subtraction subroutines, and approximately a quarter for the multiplication subroutine.

Two further items concerning the linear array should be mentioned. As we have stated, the exponent part of a number (k in Eq.(1)) is stored in cell ND1 = ND+1. The magnitude of the exponent k is thus limited in this program to the value of the largest integer that can be stored in the computer, which, as we have noted, is generally $2^{N-1} - 1$ for a computer with an N bit word length operating in the binary mode. The numbers that can be handled by this program must therefore (see Eq.(1)) be less, in magnitude, than

$$\mathrm{NBRT}^{2^{N-1} - 1} .$$

If we choose NBRT = 10 and have a word length of 36 bits, for example, this number is greater than

$$10^{10^{10}} .$$

Even if NBRT = 2 and we consider a word length of 32 bits, we find

$$2^{2^{31} - 1} > 10^{10^{8}} .$$

Considering the size of these numbers, we provide neither overflow nor underflow checks in this program.

Lastly, as mentioned earlier, the sign of a number is stored in cell ND2 = ND+2 of the linear array as a +1 or a -1. One could clearly save one cell of memory by incorporating the sign, for example with the number in the first cell, NA(1), which would then be either positive or negative depending on whether the number itself was positive or negative. Alternatively, one could incorporate the sign with each of the numbers in the first ND cells, NA(1), NA(2),... NA(ND), which would then all be either positive or negative, depending on whether the number itself is positive or negative. We considered that this saving in memory was, however, of little importance and therefore opted for the conceptual simplicity that is gained by having non-negative integers in all of the ND cells containing the digits, and being able to manipulate the sign independently of the digits.

3. The parameter repository, PRM(NDG)

So much then for the question of the manner in which we represent a number and the description of the linear array in which it is stored. We now proceed to a discussion of the subroutine PRM(NDG). The intention in establishing this subroutine was manifold. The first concerned the manner in which the arbitrary precision routines presented here might be used in conjunction with an already existing program. We might envisage, for example, a program, very possibly with a fairly complicated logical structure, in the course of which certain quantities are calculated, but have an insufficient number of significant digits due to cancellations such as we discussed earlier.

One might then desire to perform that part of the calculation in which the loss of significance occurs with more significant digits than are provided by double precision, utilizing the arbitrary precision arithmetic routines. Since these routines will inevitably increase the computation time, they would not be used when extended precision is not required. After performing part of the calculation with the arbitrary precision routines, the calculation would return to the already existing program. As we have mentioned, this link is performed by a number of peripheral subroutines which take integer, real or double precision numbers and convert them into linear arrays, and to this end a number of parameters which one may wish to vary must be established: The base, NBRT, used in the calculation, the number of digits per cell, NDW, the total number of digits, NDG, and several others determined from them,--- ND, ND1, ND2, NBASE, NBASEP. All of these parameters are in fact required by both the peripheral and the arithmetic subroutines, and hence must be established before calling any of those subroutines. There are, to be sure, many ways of accomplishing this. The guiding principle employed here was that of keeping to an absolute minimum the modifications made to an already existing program, necessitated by its incorporation of the arbitrary precision subroutines. Thus we decided not to introduce any paramters or other data via READ statements, which would introduce data into the program from external data cards. The possible disruption of the sequence of data cards read in a program of some complexity, due to the introduction of the arbitrary precision subroutines at one or a number of points within the program, was considered an excessive modification. We therefore confine to the subroutine PRM(NDG) the initial values of the parameters NBRT and NDW and the subsequent calculation of the other parameters shared by the various subroutines via COMMON/DGS/ (see (2) and (3)). Even this common statement need not appear in the external calling program. The entire task of setting up the parameters needed for the peripheral and arithmetic subroutines comprising this program is accomplished by introducing into the external calling program the single card CALL PRM(NDG), the argument NDG being the number of significant digits desired in the course of the arbitrary precision computation. The subroutine PRM(NDG) then computes ND = [NDG/NDW] (ND = largest integer less than or equal to NDG/NDW), so that the number of significant digits actually used in the computation is ND•NDW. In the event that one wishes to change any of the parameters NDG, NDW or NBRT during the calculation, this can be accomplished by specifying the values of the three parameters and introducing the single statement CALL PRM2(NDG,NDW,NBRT). Except for its argument list, PRM2 is similar to PRM. It should be noted, however, that the program which calls the subroutines of this arbitrary precision program must, in addition to the statement CALL PRM(NDG), have dimension statements for the arrays which appear as arguments in the subroutines of this program. The dimension of these arrays must be at least ND2 = [NDG/NDW] + 2. In the event that one may wish to change NDG or NDW from time to time, it is generally convenient to assign at the outset a dimension size sufficiently large to cover all reasonable eventualities.

By containing within it all statements and calculations pertinent to the parameters of the program, the subroutine PRM(NDG) not only frees the external program of any unneeded modifications, but also frees the arithmetic subroutines of any mention of specific parameter values. Once called, PRM sets up the parameters which are then transmitted to the arithmetic subroutines and determine the mode in which they operate.

4. The peripheral subroutines, input and output

We next discuss briefly the peripheral subroutines. There are three peripheral input subroutines, INT(I,NI), DFLT(DX,NX) and AFLT(A,NA). The inputs to these three subroutines are, respectively, integer, I, double precision, DX, and real, A. The subroutines return, in each case, a linear array-- NI, NX and NA, respectively-- of length ND2, corresponding to the input constant, which may then be used as input to the arithmetic subroutines. It is obviously assumed that the inputs to these subroutines, I, DX and A, do not exceed the capacity of the computer and are constants that can be handled legitimately by it. From the standpoint of the subroutines INT, DFLT and AFLT there are no other restrictions on their magnitude, except that in the case of the subroutine INT(I,NI) the number of digits, ND•NDW, alloted in the array NI must be sufficient to allow for storage of all of the digits of the input integer I. The first argument of each of these subroutines must, however, be of the type indicated: integer, double precision or real, respectively.

There are two peripheral output subroutines, RET(NX,DX) and PRINT(NC). The input argument to the subroutine RET(NX,DX) is the array NX and the subroutine returns DX, the double precision number corresponding to this array, to the calling routine. The double precision number, DX, must not exceed the overflow or underflow capabilities of the computer. The subroutine PRINT(NC) prints out the linear array NC in a form that is simple to read. The format in which this print-out appears permits a number of options to the user. Basically, the array appears as a string of digits, interspersed with blanks, preceded by a + or a - sign and a decimal point, and followed by parentheses in between which there is the exponent part of the number. Although this exponent, k in Eq.(1), refers to the base M, its value is printed as a number in the base 10. Thus the number π may be printed in the form

```
+.31415 92653 58979 32384   (     1)
```

However, it can also be printed out in the form

```
+3.14159 26535 89793 23846   (    0)
```

or with any other number of digits before the decimal, denoted by the parameter NDBD. Further, the number of digits in each block following the decimal, after which there appears a blank, need not be five, but may be any other number of the user's choice, this number being denoted by NDWP. In addition, the user may determine the over-all format for the print-out, viz., the indentation before printing the sign and the placement of the parentheses and the exponent; (in the event that more than one line is required to print out all the digits, one may wish to print the parentheses and exponent on the last line, for example). All these options-- the parameters NDBD and NDWP and the over-all format, are specified, not in the subroutine PRINT(NC) itself, but in the subroutines PRM(NDG) and WRIT(NSG,NCH,NC1). This latter subroutine is called by the subroutine PRINT(NC), which provides all the arguments in the argument list of the subroutine WRIT(NSG,NCH,NC1): Here NSG is the sign (a Hollerith + or -) of the number, NCH is an array of Hollerith characters (stored with but one character in each cell of the array) which are the digits of the number, and NC1 is an integer, the exponent part of the number. The parameters NDWP and NDBD are specified in the subroutine PRM(NDG) and the over-all format is specified in the subroutine WRIT(NSG,NCH,NC1), as is also the number of characters, NL, of the array NCH that one actually prints out. Although generally one would print out all the digits calculated, it is at times convenient to be able to print out fewer. Since the array NCH is filled with Hollerith characters, one character per cell with the blanks between the digits considered as characters in this array, the total number of characters in this array is ND•NDW + [(ND•NDW-NDBD)/NDWP]. Further, since the array NCH is formed in the subroutine PRINT(NC), this subroutine also requires the values of the parameters NDWP and NDBD. They are listed in the subroutine PRM(NDG) and passed to the subroutines WRIT(NSG,NCH,NC1) and PRINT(NC) via the COMMON/DGS/, which for these three subroutines reads

```
     COMMON/DGS/ND,ND1,ND2,NDW,NBRT,NBASE,NBASEP,NDWP,NDBD         (3)
```

rather than as shown in (2).

The reasons for putting the parameters NDWP, NDBD and NL as well as the over-all format statements in subroutines separate from PRINT(NC) were both conceptual and practical. First, as discussed earlier in describing the subroutine PRM(NDG), we wished to separate those elements of the subroutine which one might wish to vary, such as specific parameter values or specific format statements, from the structure of the subroutine performing a given function-- arithmetic or input-output in the case of the peripheral subroutines, which is fixed. Then, from the practical point of view, while this program has been written in ANSI Fortran, someone working predominantly with a specific computer might quite possibly prefer to use the already compiled (machine dependent) object decks rather than the Fortran source decks. It is nonetheless most convenient to leave in Fortran statements the items that may be varied frequently, such as the over-all format for the print-out. Wishing to keep these Fortran statements to a minimum, we therefore created the separate subroutine WRIT(NSG,NCH,NC1) which contains nothing other than the over-all format statements and the parameter NL. Thus in actual practice one might have the entire program other than this subroutine in the form of object decks, leaving only this subroutine as a Fortran source deck.

5. Arithmetic

Finally we come to the arbitrary precision arithmetic[4] subroutines themselves. The program has five subroutines of this kind-- one each for addition, subtraction and multiplication, and two for division: ADD(NA,NB,NC), SUB(NA,NB,NC), MULT(NA,NB,NC), DIV(NA,NB,NC) and SDIV(NA,NB,NC). In all of these, the first two arguments, the arrays NA and NB, are the input to the subroutine, and the subroutine returns the array NC which represents, respectively, the sum, difference, product or quotient of the numbers represented by the arrays NA and NB. Omitting the first letter, N, to represent the number itself, the subroutine and the corresponding operations they perform are given below:

ADD(NA,NB.NC)	$C = A + B$
SUB(NA,NB,NC)	$C = A - B$
MULT(NA,NB,NC)	$C = A \cdot B$

DIV(NA,NB,NC)	C = A/B
SDIV(NA,NB,NC)	C = A/NB(1)

In the calling of any of the above five subroutines, the first two argument names may be identical if they correspond to the same number. Thus, for example, one may have the call statement

```
CALL ADD(NX,NX,NZ)
```

However, in calling the subroutines ADD, SUB, MULT or DIV, the last argument name must never be identical to either of the first two argument names, even if one does not desire to keep either of those arguments, since throughout the operation performed by the particular subroutine the array listed as the third argument is used as an intermediate storage area. Thus, for example, one must not write either

```
CALL MULT(NX,NY,NX)
```

or

```
CALL MULT(NX,NY,NY)
```

even if one is not interested in preserving either the first or the second argument array. In the case of the subroutine SDIV, this intermediate storage area is not required, and this subroutine has been written so that one may write the statement

```
CALL SDIV(NX,NY,NX)
```

or

```
CALL SDIV(NX,NY,NY)
```

In the former case, the result of the division is stored in the array NX, in the second case in the array NY. While the option of being able to write such call statements also for the subroutines ADD, SUB, MULT and DIV would clearly have had certain advantages, we considered them to be outweighed by the fact that it would have necessitated adding an array, dimensioned within the subroutine, to each of these subroutines. In the case of the subroutines ADD, SUB, MULT and SDIV we chose to write them so that there were no arrays other than the three named in the argument list. With all of the arithmetic subroutines, the first two argument arrays are either unaltered in the course of the subroutine, or, if there are altered, then they are returned in their original form by the end of the subroutine.

The difference between the subroutines DIV and SDIV will be explained shortly, Of the first three of the arithmetic subroutines there is relatively little to be said. The essence of the operations in each case follows very closely the procedure used in elementary school arithmetic. The basic operations involve comparison, shifting, carrying, adding, subtracting and multiplying. The subroutine for multiplication has one refinement. We are involved here with multiplying the number in a given cell of NA by the number in some other cell of NB, say NA(J)·NB(K), where $1 \leqq J \leqq ND$, $1 \leqq K \leqq ND$. Now it might at first sight appear consistent to neglect, in the product, all terms for which $J + K > ND$, since an individual product NA(J)·NB(K) should be entered in the cell NC(J+K). There may be many such terms, however-- specifically, ND of them such that $J + K = ND + 1$, $ND - 1$ of them such that $J + K = ND + 2$, and so forth. And for large values of ND the sum of all products such that $J + K > ND$ would, after carrying, make an important contribution to the number that should appear in NC(ND). For this reason we use the cells ND1 and ND2 of the array NC for products of the form NA(J)·NB(K) where $J + K = ND1$ and $J + K = ND2$, respectively. Only at the end of the calculation, when all carrying and shifting operations have been performed, do we load the proper exponent in NC(ND1) and the proper sign in NC(ND2).

The subroutine SDIV follows very closely the operation known familiarly as short division. It may be used only if the numbers stored in the cells NB(J), J = 2,3, ... ND are all zero, since it uses only the number stored in NB(1), and of course the exponent NB(ND1) and the sign NB(ND2). It is an extremely fast routine, compared with the subroutine DIV, and therefore is to be preferred whenever the number corresponding to the array NB is such that all digits after the first NDW are zero. It is thus particularly useful for dividing by an integer having not more than NDW digits, such as occurs, for example, in the programming of power series in which successive terms must be divided by integers or the product of integers. Provided these integers are less than NBASE = $\mathrm{NBRT}^{\mathrm{NDW}}$ (if NBRT = 10 and NDW = 5 these integers must be less than 100,000) one should use SDIV(NA,NB,NC) for division. In the event that the number corresponding to the array NB is zero (in which case NB(1) = 0), the subroutine returns the array NC with the elements of NA in it, i.e., one gets the same answer as upon division by unity. However, an error message DIVISION BY ZERO is printed out to warn the user.

The subroutine DIV(NA,NB,NC) must be used for division if the number B,corresponding to the array NB, is such that any digits after the first NDW are non-zero. This subroutine first computes the reciprocal of B and then multiplies (by calling the subroutine MULT) the result by NA. The reciprocal is computed by the non-linear iterative procedure defined by

$$x_{n+1} = x_n(2 - Bx_n)$$

which involves two multiplications and one subtraction. This recursion is equivalent to the Newton-Raphson method[5] for the function $f(x) = \frac{1}{x} - B$. The initial value for this recursion, x_0, which should be chosen to be as close as possible to the reciprocal of B for rapid convergence, is determined within the subroutine using the contents of the first two cells of the array NB. The number of iterations required is also determined within the subroutine, by the requirement that the final iteration should give the reciprocal of B correct to all ND·NDW places. In the event that B = 0, the response of this subroutine is identical to that just described in connection with the subroutine SDIV(NA,NB,NC).

Concerning the memory requirements of the arithmetic subroutines, we note that with the exception of the subroutine DIV(NA,NB,NC), there are no arrays internal to these subroutines--the only arrays are those appearing in the argument list. The subroutine DIV(NA,NB,NC) has, in addition to the three arrays in the argument list, three linear arrays internal to the subroutine.

With regard to speed of operation, we have used these subroutines on a number of different high-speed computers and found, running the program in base ten (NBRT = 10), with a 32 or 36 bit word-length computer, that for arrays containing 60 digits, the operations of addition, subtraction and multiplication are of the order of 1000 times as long as for the same operation in double precision on the same computer. (These operations are thus of the order of milliseconds rather than microseconds, as is the case for the basic arithmetic operations performed by the computer.) A division performed by SDIV takes about one-fourth the time of an addition performed by ADD, and a division performed by DIV takes about 8 times as long as a multiplication performed by MULT. In using this arbitrary precision program for the calculation of the elementary functions to 60 digits, we have again found this rough factor of 1000 times the time required by the computer for the computation of the same function in double precision.

Parameters and their names

NDG	Number of DiGits
ND	Number of Digits
ND1	ND + 1
ND2	ND + 2
NDW	Number of Digits per Word
NBRT	Number Base RooT
NBASE	Number BASE
NBASEP	Number BASE, Provisional
NDWP	Number of Digits per Word in the Print=out
NDBD	Number of Digits Before the Decimal

Subroutines and their names

ADD(NA,NB,NC)	ADDition
SUB(NA,NB,NC)	SUBtraction
MULT(NA,NB,NC)	MULTiplication
DIV(NA,NB,NC)	DIVision
SDIV(NA,NB,NC)	Short DIVision
PRM(NDG)	PaRaMeters
PRM2(NDG)	PaRaMeters, 2nd version
WRIT(NSG,NCH,NC1)	WRITe
INT(I,NI)	INTeger
DFLT(DX,NX)	Double precision FLoaTing point number
AFLT(A,NA)	reAl single precision FLoaTing point number
RET(NX,DX)	RETurn double precision floating point number
PRINT(NC)	PRINT

REFERENCES

1. D.E. Knuth- Seminumerical Algorithms, Vol. 2 of his series of books on "The Art of Computer Programming (Addison-Wesley Pub. Co., 1969). On p. 181 of this volume there is a fine etymological discussion of the merits of the terms exponent part and fraction part as opposed to characteristic and mantissa. We side with his choice of nomenclature.

2. We in no way imply, by our use of the word "decimal", that the base, M, is equal to ten. Indeed, in "An Introduction to the Theory of Numbers" (Oxford University Press, 4th ed. 1960), Hardy and Wright speak of 'decimals in the scale of r', with the appended footnote: "We ignore the verbal contradiction involved in the use of 'decimal'; there is no other convenient word." (Sec. 9.3 on Representation of numbers in other scales, pp. 111, 112.)

3. For an extremely interesting discussion of the fact that the execution time for a computer algorithm for multiplying two numbers does not have to increase quadratically with the number of digits, see pp. 258-278 of ref. 1.

4. By far the most interesting reference on this subject is the volume by Knuth cited above in ref. 1. One should note, in particular, Sec. 4.3 entitled "Multiple-precision arithmetic", pp. 229-280. Specific articles of interest, some of which are

mentioned in the bibliography on pp. 243-245 of that volume, are

a) D.A. Pope and M.L. Stein, Multiple Precision Arithmetic, Comm. ACM 3, 652 (1960).

b) P. Rabinowitz, Multiple Precision Division, Comm. ACM 4, 98 (1961).

c) A.G. Cox and H.A. Luther, A Note on Multiple Precision Arithmetic, Comm. ACM 4, 353 (1961).

d) M.L. Stein, Divide-and-Correct Methods for Multiple Precision Division, Comm. ACM 7, 472 (1964).

e) E.V. Krishnamurthy, On a Divide-and-Correct Method for Variable Precision Division, Comm. ACM 8, 179 (1965).

f) B.I. Blum, An Extended Arithmetic Package, Comm. ACM 8, 318 (1965).

g) S.K. Nandi and E.V. Krishnamurthy, A Simple Technique for Digital Division, Comm. ACM 10, 299 (1967).

h) E.V. Krishnamurthy and S.K. Nandi, On the Normalization Requirements of Divisor in Divide-and-Correct Methods, Comm. ACM 10, 809 (1967).

5. This method is also called, simply, Newton's method. See "Survey of Numerical Analysis", edited by John Todd (McGraw-Hill Book Co., Inc. New York, N.Y. 1962). Note in particular pp. 30-35 in the article on classical numerical analysis by John Todd, pp. 223,224 in the article on matrix computations by Morris Newman, and p. 259 in the article on numerical methods for finding solutions of nonlinear equations by Urs Hochstrasser. Also of interest in this regard are p. 244 of ref. 1 and the book by J.F. Traub-- "Iterative Methods for the Solution of

Equations (Prentice Hall, Inc., Englewood Cliffs, N.J., 1964).

ACKNOWLEDGEMENTS

We gratefully acknowledge several very helpful discussions with Morris Newman of the Applied Mathematics Division of the National Bureau of Standards in the formative stages of this work. His ideas were particularly influential in determining the manner in which numbers were dealt with in this program and in the use of the non-linear recursion for the division subroutine DIV.

Almost all of this program was developed and tested during the spring and summer of 1969, when the author was at the Physics Department of the State University of New York at Stony Brook, L.I. He wishes to thank the members of that department for their warm hospitality.

That this endeavor was more fun than work was due primarily to the constant encouragement, technical assistance, warmth and sense of humor of John Milazzo, head of the Instruction and Research Support Group of the Computing Center at SUNY, Stony Brook. A very particular debt is also due two other members of his group, Gary DiPillo and Domenic Seraphin, for their friendly and patient willingness to explain the subtleties of computer operation which were essential to the working out of the details of this program.

DEPARTMENT OF COMMERCE ★ NATIONAL BUREAU OF STANDARDS ★