OAK RIDGE NATIONAL LABORATORY
operated by
UNION CARBIDE CORPORATION
NUCLEAR DIVISION
for the
U.S. ATOMIC ENERGY COMMISSION

UNION CARBIDE

ORNL - TM - 1249

COPY NO. 140

DATE - Nov. 4, 1966

Neutron Physics Division

SIGNIFICANCE ARITHMETIC FOR FORTRAN

W. R. Burrus and B. W. Rust*

Abstract

The FORTRAN statement X # 193./71. - 2721./1001. has a true value of 0.0000281409..., but a single-precision FORTRAN computation (27 bit fraction) yields 0.28133392E-04 (using an E14.8 output FORMAT) due to loss of significance when subtracting. When the number of significant digits is questionable, the same problem may be run in both single and double precision. This is inconvenient because existing programs must be modified and because double precision requires twice as much storage space. We have implemented a simple alternative due to Max Goldstein for keeping track of the amount of significance in floating-point numbers by representing them in unnormalized form. Programs may be written in ordinary FORTRAN-IV, and the significance operations implemented by use of the ALTER feature of the loader. Some problems and applications of significance arithmetic are discussed.

Author

*Computer Science Center, Union Carbide Nuclear Division, Oak Ridge.

NOTE:
This Work Partially Supported by
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Under Order R-104 (01)

INTRODUCTION

Meaningless digits come into a numerical calculation from four sources:

1. errors in the input data,
2. numbers going out of range (or spilling),
3. roundoff in the arithmetic operations,
4. loss of significance when subtracting numbers with similar value or adding numbers of similar magnitude and opposite sign.

The presence of meaningless digits in the final results is often not obvious from a superficial inspection, and many a spurious digit has found its way from a numerical calculation into the literature. Many schemes have been proposed to keep track of the number of significant digits in the results. (For explanation, we will refer to the number of accurate digits, even though the calculation is performed on a binary computer. It will be understood that one digit of accuracy implies about 3.4 bits of accuracy.) Some of these methods are mentioned briefly, but the scheme that was used with the IBM 7090/7094 FORTRAN-IV is considered in detail. A few results using a significance tracing arithmetic are given.

The simplest significance tracing scheme, in concept, is "interval arithmetic," in which each number is represented by a pair of numbers equal to the number's smallest and largest possible value. Every arithmetic operation or input/output operation transforms the lower and upper bound so that the resulting interval is sufficiently wide to include the true result. Interval arithmetic guarantees that all the digits which agree in the lower and upper bound are accurate. The primary disadvantage of interval arithmetic is that it requires two floating-point words to represent

one number, which necessitates more storage space; also, its use tends to give pessimistic results in a long computation, since for each arithmetic operation the assumption is made that the worst possible lower and upper bound will result.

Other schemes have been suggested to offset the pessimistic feature of interval arithmetic by considering tendencies for the accumulation of errors over many operations. Richtmeyer[1] suggests an arithmetic scheme in which some digits (or bits) explicitly carry the number of significant digits. Another type of arithmetic, discussed by Wadey,[2] computes the error at each step as if the errors of the operands were independently statistically distributed. This last method involves a square-root operation at each step.

The method that we selected to implement is based on an unnormalized floating-point number representation suggested by Ashenhurst and Metropolis[3] in which numbers are carried as fractions that contain only significant digits preceded by leading zero digits. These fractions and a corresponding exponent are carried in a single IBM 7090/94 word.

Although significance arithmetics can handle all four types of error mentioned above, our emphasis is only on the loss of significance when two numbers of nearly equal size are subtracted or when two numbers of nearly equal magnitude and opposite sign are added. Our philosophy is to treat known errors in the input data explicitly in the program. Errors due to input truncation can similarly be taken into account by augmenting the input errors by a suitable amount. But we are still left with the not-so-easy-to-predict arithmetical errors which are inherent in a finite computer.

GOLDSTEIN'S RULES

Goldstein[4] has described a set of rules for performing unnormalized arithmetic on the IBM 7090/94. These rules have been incorporated as a special modification into the NYU Computing Center machine. A special operating mode was provided in which the ordinary floating-point arithmetic instructions were interpreted by the hardware as unnormalized instructions. Standard floating-point representation was used, except that each number had just enough leading zeros in the fraction that all the remaining bits were alleged to be significant. Addition and subtraction simply omitted the usual normalization step. Multiplication and division gave the same number of leading zeros as the less accurate operand, after which a floating-point round operation took place.

In unnormalized arithmetic the result cited in the abstract is obtained as

```
 0.27183098E 01
-0.27182817E 01
---------------
 0.00000281E 01
```

whereas the correctly normalized results is 0.28140873E-04. Zero must be treated as a special case. The ordinary 7090/94 zero is an all-zero word, but it is possible to obtain a significance zero as the result of subtracting two equal numbers:

```
 0.34567890E-13
-0.34567890E-13
---------------
 0.00000000E-13
```

This "significance zero" implies that the true result is less in magnitude than 0.00000001E-13, or 10^{-21}. Multiplication or division which involves one or two "significance zero" operands must return an appropriate result. Thus (0.00000000E-13)**2 should have a value which implies that the magnitude is less than 10^{-42}.

The price that one pays for using an unnormalized arithmetic is that more significance is lost than when using comparable normalized arithmetic. However, Goldstein has demonstrated that in most cases the loss of accuracy is not significantly worse than the loss that occurs in ordinary 7090/94 FORTRAN arithmetic, which ignores the rounding operation. Compared with interval arithmetic, unnormalized arithmetic does not guarantee that a certain number of digits are correct, but the results will usually be correct to within a few units in the last significant digit.

FORTRAN IMPLEMENTATION

The IBM 7090/94 FORTRAN's have built-in provision for three types of floating-point arithmetic: REAL, DOUBLE PRECISION, and COMPLEX. The normal FORTRAN floating-point arithmetic is biased or truncated, since a floating-point round operation is not called for. It occurred to us that it would be convenient to have many different types of arithmetic available. In addition to the three above, one might wish the following:

ROUNDED single precision arithmetic

RANDOM ROUNDED arithmetic (using pseudo-randomly generated bits) in order to check statistical models of error propagation

INTERVAL arithmetic, etc.

as well as unnormalized SIGNIFICANCE arithmetic. It would be convenient if the arithmetic were implemented in a package of subroutines so that the type of arithmetic could be changed without modification to the source FORTRAN program.

An easy solution for single-word arithmetic is to hand-code FORTRAN FUNCTION routines ADD, SUB, FMP, and DIV to perform the elementary operations. But this is somewhat inconvenient for the programmer, since an

expression of the form

$$X = A*(B-C)/D$$

becomes

$$X = DIV(MPY(A,SUB(B,C)),D) \quad .$$

It would be much more preferable to use the ordinary symbols and have them interpreted in terms of the desired arithmetic.

Some FORTRAN compilers, such as the CDC FORTRAN-63, provide a TYPE OTHER declaration, so that transfers to a special subroutine are automatically provided. One may achieve the same effect for double-word arithmetic in FORTRAN by declaring the variables to be TYPE COMPLEX and by supplying hand-coded routines which have the same name as the COMPLEX routines but which implement the desired arithmetic. An additional problem in the implementation of SIGNIFICANCE arithmetic is that some way has to be provided for the number of significant digits to be ascertained from the output.

It has turned out that single-word auxiliary arithmetic can be implemented very simply in FORTRAN-IV for the 7090/94. In order to use the ordinary arithmetic operations and have them correctly reinterpreted, one may redefine the floating-point operations FAD, FSB, FDP, and FMP by means of MACRO definitions. It is first necessary to obtain an ASSEMBLY language listing of the compiled program (or a PREST deck) and to reassemble with the MACRO definitions included. In our version, we obtain a PREST deck from the FORTRAN compiler (IBFTC, version 9) and utilize the ALTER feature of the input editor (IEDIT) to insert the necessary MACRO's.

In the MACRO definition, we included a switch which could be tested. If SWTCH .EQ. 0.0, then the ordinary FORTRAN arithmetic was performed,

but if SWTCH .NE. 0.0, a transfer was made to a special routine. With the switch off, the execution time of the ordinary arithmetic was increased by seven cycles per operation on the 7090. To obtain easy access to SWTCH by all programs, it was placed in labeled COMMON. Thus for ordinary arithmetic

```
      CØMMØN  /SCØM/  SWTCH
                .
                .
                .
      SWTCH # 0.0
        C # A-B
                .
                .
```

and for significance arithmetic

```
                .
                .
      SWTCH # 1.0
        C # A-B
                .
                .
```

One could use additional values of SWTCH, if desired, to switch back and forth between several different types of arithmetic.

The following form was used for the redefinition MACRO's so that they could be used in conjunction with indexing and indirect addressing: Assume that the FORTRAN statement is A # B + C:

```
ZFAD   ØPSYN   FAD       SØ THAT ØRDINARY FAD CAN BE USED
 FAD   MACRØ   X,Y
       NØP     X,Y       REPLACES ØRIGINAL FAD IN THE TEXT
                         SØ THAT IF X WERE ** CØRRECT ADDRESS
                         WØULD HAVE BEEN STØRED IN PRØPER PLACE
       NZT     SWTCH
       TRA     *+5       TRANSFER FØR ØRDINARY FAD
       SXA     *+2,4     PREPARE FØR TRANSFER TØ RØUTINE
       TSX     SGADD,4
    R  AXT     **,4      RESTØRE INDEX REGISTER
       TRA     *+2
      ZFAD*    *-7       NØRMAL FAD PERFØRMED WHEN SWTCH IS ØFF
      ENDM     FAD
```

The SGADD routine had the following form:

```
        ENTRY   SGADD
SGADD   STØ     B
        SXA     RTN,4     STØRES RETURN INFØRMATIØN
        CLA     -4,4      PICKS UP NØP X,Y
        STA     FETCH     STØRES ADDRESS
        STT     FETCH     AND STØRES TAG
        XEC     1,4       REMØTELY EXECUTES INSTRUCTIØN AT RZ
                          TØ RESTØRE XR4 IN CASE Y # 4
FETCH   CLA     **
        STØ     C
          .               PERFØRMS REQUIRED SIGNIFICANCE
          .               ADDITIØN; LEAVES ANSWER IN ACC.
          .
 RTN    AXT     **,4
        TRA     1,4
   B
   C
        END
```

A MAP listing of the ALTER deck and of the SIGNIFICANCE routines is given in the appendix. These routines follow Goldstein's rules except that we did not bother with a correct round operation after division. True zero (denoted by 0) and "significance zero" (denoted by 0.0) were treated as special cases. Figure 1 shows the results of the basic arithmetical operations when either operand or both operands are zeros. We make no claim for efficiency, and would appreciate being informed of faster methods.

Fortuitously, the outputing of the number of significant digits presents no problem in 7090/94 FORTRAN (II or IV). The input/output routines make the BCD conversion so that the number of leading zeros in the fraction is preserved if an "E" type FORMAT is used. For example, an unnormalized 1.0 might print out as 0.00100000E 03 with an E14.8 FORMAT.

EXAMPLES

Some of the features of unnormalized significance arithmetic are nicely brought out by attempting to invert the notoriously poorly conditioned Hilbert matrices of various orders. Table 1 shows a 3 x 3 Hilbert

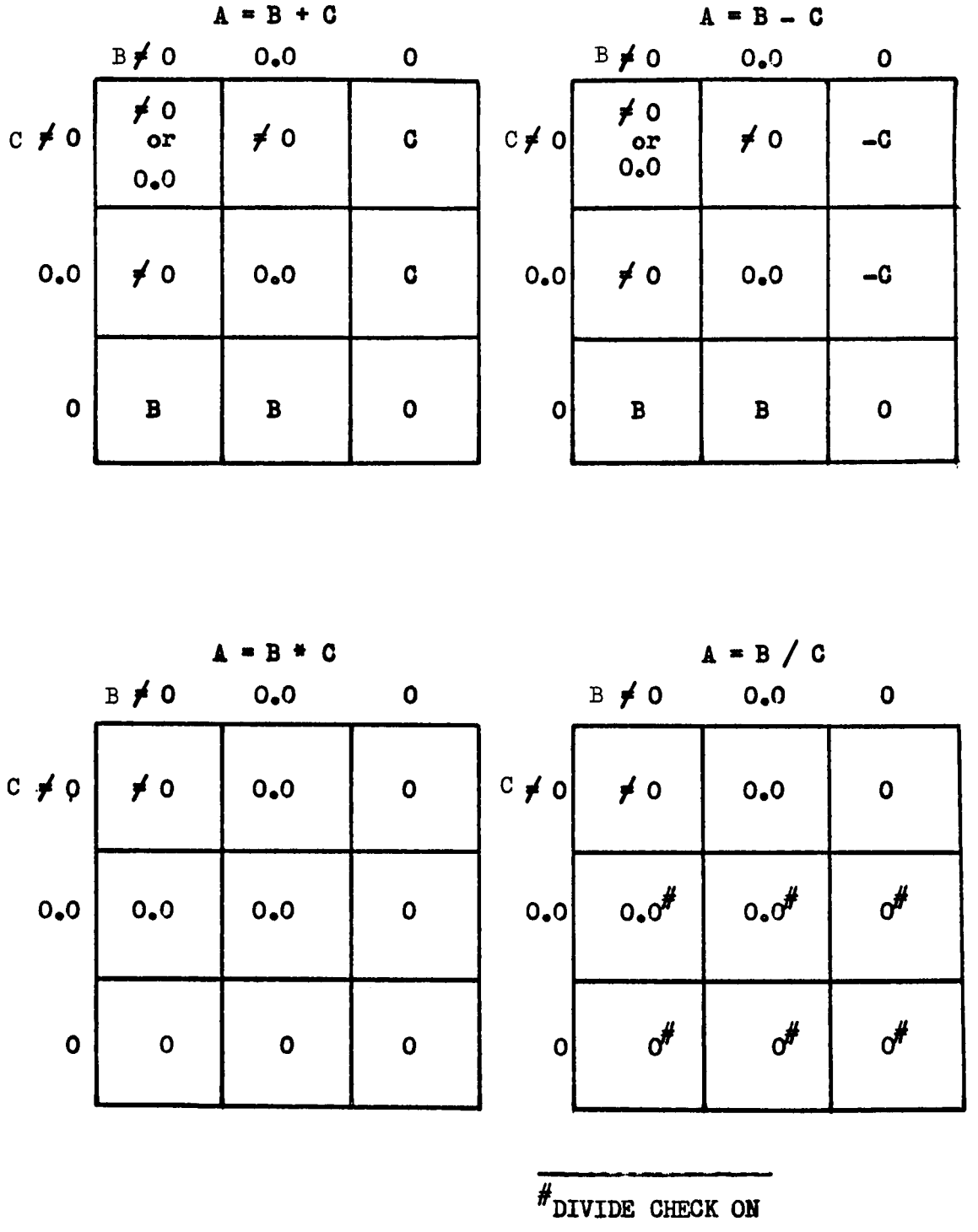

Fig. 1. The Results of Significance Arithmetic for Various Combinations of Zero and Nonzero Arguments. 0.0 denotes a significance zero and 0 denotes a true zero.

Table 1. Three-by-Three Hilbert Matrix Example

MATRIX A

0.09999999E 01	0.50000000E 00	0.33333333E-00
0.50000000E 00	0.33333333E-00	0.25000000E-00
0.33333333E-00	0.25000000E-00	0.20000000E-00

INVERSE OF A

0.00900000E 04	-0.00360000E 04	0.00300001E 04
-0.00360000E 04	0.01920001E 04	-0.00180000E 05
0.00300000E 04	-0.01800001E 04	0.00180001E 04

PRODUCT

0.00099999E 04	-0.00000000E 04	0.00000000E 04
0.00000001E 03	0.00010000E 04	0.00000000E 04
-0.00000001E 03	-0.00000000E 04	0.00010000E 04

matrix, its computed inverse (using a Gaussian elimination method), and the product of the computed inverse times the original matrix. Note that the elements of the inverse matrix have lost several digits of significance and that the off-diagonal elements of the product are not all zero as they should be. It is clear from the significance arithmetic, however, that the off-diagonal elements of the product have no significance to speak of. The 4 x 4 case is shown in Table 2. About half the significance has been lost in the computed inverse. Because the elements of the correct Hilbert inverse have simple floating-point representation, the spurious digits are easy to recognize. The diagonal elements of the product have lost nearly all significance, so that we should expect the worse for the 5 x 5 case. Table 3 shows that this is about as far as we can go in single precision on the 7090/94.

Table 4 shows some elementary FORTRAN statements and a comparison of the true value with the values obtained by ordinary arithmetic and significance arithmetic. The first result, R1, is the example given in the abstract. The second is of interest because ordinary arithmetic yields a small negative number instead of zero. R4 and R6 show how this false zero can propagate into other results. The third result is a true zero. R5 illustrates the effect of attempting to divide by true zero. The eighth result, R8, is contrived to show an extremely adverse case of significance arithmetic, where three significant digits are indicated but none are correct. In this case, "0.1" had a small conversion error when converted to binary, and combined with the roundoff, the sum picked up a spurious bit once in about every five summations. One can see from this example that significance arithmetic is more reliable if only a few numbers enter into the result. The last result, R9, shows how the error in R8 propagates.

Table 2. Four-by-Four Hilbert Matrix Example

MATRIX A

C.09999999E 01	0.50000000E 00	0.33333333E-00	0.25000000E-00
C.50000000E 00	0.33333333E-00	0.25000000E-00	0.20000000E-00
C.33333333E-00	0.25000000E-00	0.20000000E-00	0.16666666E-00
C.25000000E-00	0.20000000E-00	0.16666666E-00	C.14285714E-00

INVERSE OF A

0.00001600E 06	-0.00001200E 07	0.00002400E 07	-0.00001400E 07
-0.00012000E 06	0.00012000E 07	-0.00027000E 08	0.00016800E 07
C.00024000E 06	-0.00027000E 07	0.00064801E 07	-0.00042001E 07
-C.00014000E 06	0.00016800E 07	-0.00042001E 07	C.00028000E 07

PRODUCT

C.00000100E 06	0.00000000E 07	-0.00000000E 07	-0.00000000E 07
C.00000000E 06	0.00000010E 07	-0.00000000E 07	0.00000000E 07
-C.00000000E 05	0.00000000E 07	0.00000001E 07	-C.00000000E 07
C.00000001E 05	0.00000001E 06	-0.00000000E 07	C.00000101E 07

Table 3. Five-by-Five Hilbert Matrix Example

MATRIX A

0.99999999E 01	0.50000000E 00	0.33333333E-00	0.25000000E-00	0.20000000E-00
0.50000000E 00	0.33333333E-00	0.25000000E-00	0.20000000E-00	0.16666666E-00
0.33333333E-00	0.25000000E-00	0.20000000E-00	0.16666666E-00	0.14285714E-00
0.25000000E-00	0.20000000E-00	0.16666666E-00	0.14285714E-00	0.12500000E-00
0.20000000E-00	0.16666666E-00	0.14285714E-00	0.12500000E-00	0.11111110E-00

INVERSE OF A

0.00000025E 08	-0.00000003E 10	0.00000009E 10	-0.00000013E 10	0.00000005E 10
-0.00000030E 09	0.00000048E 10	-0.00000019E 11	0.00000266E 10	-0.00000124E 10
0.00000104E 09	-0.00000189E 10	0.00000079E 11	-0.00001175E 10	0.00000567E 10
-0.00001400E 08	0.00000268E 10	-0.00001176E 10	0.00001792E 10	-0.00000881E 10
0.00000630E 08	-0.00000126E 10	0.00000567E 10	-0.00000881E 10	0.00000440E 10

PRODUCT

0.00000001E 08	0.00000000E 10	-0.00000000E 10	-0.00000000E 10	-0.00000000E 10
0.00000000E 08	0.00000000E 10	-0.00000000E 10	0.00000000E 10	-0.00000000E 10
0.00000000E 08	0.00000000E 10	-0.00000000E 10	0.00000000E 10	-0.00000000E 10
0.00000000E 08	0.00000001E 09	-0.00000000E 10	-0.00000000E 10	-0.00000000E 10
0.00000000E 08	0.00000001E 09	0.00000000E 10	-0.00000000E 10	-0.00000000E 10

Table 4. Comparison of True Value with Computed Value Using Ordinary and Significance Arithmetic for Some Simple FORTRAN Expressions.

FORTRAN Statement	Results to 8 Figures with E14.8		
	True Values	Computed Value	
		Ordinary Arithmetic	Significance Arithmetic
R1 # 193./71.-2721./1001.	0.28140873E-04	0.28133392E-04	0.0000281E 01
R2 # 0.1*10. - 1.0	0.	-0.75405806E-08	0.00000000E 00
R3 # 0.0	0.	0.	0.
R4 # 1.0/R2	∞	-0.13421772E 09	0.00000000E 17*
R5 # 1.0/R3	∞	0.*	0.*
R6 # 100. + R4	∞	-0.13421763E 09	0.00000000E 17
R7 # 1.0 + 100. - 100.	0.10000000E 01	0.09999999E 01	0.01000000E 02
DO 2 I # 1,10000 2 R8 # R8 + 0.1	0.10000000E 04	.99997382E 03	0.10000094E 04
R9 # R8 - 1000.	0.	-0.26130676E-01	0.00000947E 03

*Divide check.

PRECAUTIONS

It is possible that system MACRO's such as an explicit or implied FLOAT will fail to operate properly because of the redefined floating-point operations. It is safer to switch the significance arithmetic off while performing a mode conversion, etc., as shown below:

```
      .
      .
SWTCH # 0.0
   X # I
SWTCH # 1.0
      .
      .
```

If SWTCH is not set and is undefined, the FORTRAN-IV loader will store an STR instruction in its location, and significance arithmetic will result.

It is conceivable that one might want to perform normal floating-point operations on unnormalized numbers and obtain normalized results. The present method of implementing significance arithmetic will not allow this because a floating-point multiplication of two unnormalized numbers may or may not give a normalized result and a floating point division of two unnormalized numbers will give the wrong answer if the divisor has more leading zeros than the dividend. For this reason care should be taken not to attempt to perform these normal floating-point operations on unnormalized numbers. One way of avoiding this difficulty would be to normalize the operands before carrying out the operations. Instead of testing the status of the switch in the MACRO, one could just transfer to the subroutine that carries out the arithmetic. Then in the arithmetic subroutine the switch could be checked. If significance arithmetic were desired, the operation could proceed in the same manner as it does now,

but if normal arithmetic were desired the operands could be normalized and the normal floating-point operation carried out. In such a scheme the floating divide MACRO might have the form

```
FDP     MACRØ   X,Y
        NØP     X,Y       REPLACES ØRIGINAL FDP IN THE TEXT SØ
                          THAT IF X WERE ** CØRRECT ADDRESS
                          WØULD HAVE BEEN STØRED IN PRØPER PLACE
        SXA     *+2,4     PREPARE FØR TRANSFER TØ RØUTINE
        TSX     SGDVP,4
  R     AXT     **,4      RESTØRE INDEX REGISTER
```

The subroutine itself might have the form

```
        ENTRY   SGDVP
SGDVP   STØ     DVND
        SXA     RTN,4     STØRE RETURN INFØRMATIØN
        CLA     -2,4      PICKS UP NØP X,Y
        STA     FETCH     STØRE ADDRESS
        STT     FETCH     AND TAG
        XEC     2,4       REMØTELY EXECUTES INSTRUCTIØN AT
                          R TØ RESTØRE XR4 IN CASE Y = 4
FETCH   CLA     **
        STØ     DVSR
        CLA     SWTCH     CHECK THE SWITCH AND TRANSFER
        TNZ     SG        TØ SG IF SIGNIFICANCE ARITHMETIC
                          DESIRED
        CLA     DVSR      ØTHERWISE,
        FAD     ZERØ      NØRMALIZE THE DIVISØR
        STØ     DVSR
        CLA     DVND
        FAD     ZERØ      THEN NØRMALIZE THE DIVIDEND
        FDP     DVSR      AND CARRY ØUT THE DIVISIØN
        TRA     RTN
SG       .                PERFØRM THE
         .                DIVISIØN IN
         .                SIGNIFICANCE ARITHMETIC
RTN     AXT     **,4
        TRA     2 4
SCØM CØNTRL SWTCH,DVND    LABELED CØMMØN BLØCK SCØM
SWTCH   BSS     1         SWTCH IN LABELED CØMMØN
DVND    HTR     0
DVSR    HTR     0
ZERØ    DEC     0.0
```

Note that such an implementation has a disadvantage in that it requires a transfer to a subroutine and the execution of several instructions to carry out ordinary arithmetic. This would increase the running time considerably

if the program used mostly ordinary arithmetic performed on numbers that would never have leading zeros. But it has an advantage in that it decreases the number of instructions in the MACRO itself and hence the number of instructions that would replace each floating-point operation in the main program. This might be advantageous for large programs which almost fill up the memory.

Since FORTRAN library routines return normalized results, the amount of significance will be lost if they are called with unnormalized arguments unless special precautions are taken. Goldstein[4] and Ashenhurst[5] discuss the evaluation of FUNCTIONS which preserve significance. One possible scheme to determine the amount of significance in the function value corresponding to an unnormalized argument is to evaluate the function at both ends of the significance interval corresponding to the argument. For example, if it were required to find the SIN of the number 0.00012321E+4, then the number 0.00012321E+4 could be replaced by two numbers — 0.00012321E+4 itself and the number obtained by adding a binary one to the last bit of it. The result would be the significance interval corresponding to 0.00012321E+4. Then the SIN could be called for both of these values, getting a normalized result each time. The amount of significance that the answer should have would be the number of digits that are identical in the two results, and the correct answer could be obtained by denormalizing one of them until it had just this many significant digits.

CONCLUSIONS

Unnormalized floating-point significance arithmetic is easy to implement in FORTRAN-IV; however, some precautions must be exercised by the programmer. It is easy to extend the method used to obtain a variety

of different types of arithmetic, as well as significance arithmetic. We have found significance arithmetic to be extremely valuable when running a new problem for the first time or when running the problem to obtain final results to be published.

In addition, there are some instances where significance arithmetic can be used routinely to good advantage. A common problem is testing a floating-point number against zero to branch in an algorithm. Testing against zero is hazardous in ordinary arithmetic because of the accumulation of roundoff errors and the loss of significance. But significance arithmetic provides a simple solution by ignoring the bits of lowest significance, as in

```
IF (AND(CRIT,MASK) .EQ. 0.0) .... ,
```

with MASK defined by a DATA statement to be an octal constant which masks out the last few bits of the word.

ACKNOWLEDGMENT

We wish to express our appreciation to Jack Zeigler and Buford Carter of the Computer Sciences Center, Union Carbide Nuclear Division, Oak Ridge, for their advice and encouragement in trying to "beat the system."

REFERENCES

1. R. D. Richtmeyer, The Estimation of Significance, NYO-9083 (1960).

2. W. G. Wadey, J. Assoc. Computing Mach. 7, 129 (1960).

3. R. L. Ashenhurst and N. Metropolis, J. Assoc. Computing Mach. 6, 415 (1959).

4. M. Goldstein, Commun. Assoc. Computing Mach. 6, 111 (1963).

5. R. L. Ashenhurst, J. Assoc. Computing Mach. 11, 168 (1964).

APPENDIX

Listing of ALTER DECK

and

SIGNIFICANCE Routines

LISTING OF ALTER DECK.

```
        *ALTER  1
        NOCRS
MAKE    MACRO   A,B
Z%A     OPSYN   A
A       MACRO   X,Y
        PMC     ON
        NOP     X,Y
        NZT     SWTCH
        TRA     *+5
        SXA     *+2,4
        TSX     B,4
        AXT     **,4
        TRA     *+2
        Z%A%*   *-7
        PMC     OFF
        ENDM    A
        ENDM    MAKE
        MAKE    FAD,SGADD
        MAKE    FSB,SGSUB
        MAKE    FMP,SGMPY
        MAKE    FDP,SGDVP
        *ENDAL
```

SIQADD
ASSEMBLED TEXT.

$TEXT SIQADD

						ENTRY	SGADD
BINARY CARD ID. SIQA0002							
00000	0601 00 0 00045	10001	SGADD	STO	B		
00001	0634 00 4 00043	10001		SXA	RTN,4		
00002	0500 00 4 77774	10000		CLA	-4,4		
00003	0621 00 0 00006	10001		STA	FETCH		
00004	0625 00 0 00006	10001		STT	FETCH		
00005	0522 00 4 00001	10000		XEC	1,4		
00006	0500 00 0 00000	10000	FETCH	CLA	**		
00007	0601 00 0 00046	10001		STO	C		
00010	0500 00 0 00045	10001		CLA	B		
00011	0100 00 0 00042	10001		TZE	USEC		
00012	4320 00 0 00047	10001		ANA	MASK1		
00013	0100 00 0 00023	10001		TZE	TESTC		
00014	0500 00 0 00045	10001		CLA	B		
00015	4300 00 0 00046	10001		UFA	C		
00016	0760 00 0 00011	10000		FRN			
00017	4100 00 0 00043	10001		TNZ	RTN		
00020	0500 00 0 00046	10001		CLA	C		
00021	4320 00 0 00050	10001		ANA	#0777000000000		
00022	0020 00 0 00043	10001		TRA	RTN		
BINARY CARD ID. SIQA0003							
00023	0500 00 0 00046	10001	TESTC	CLA	C		
00024	0100 00 0 00040	10001		TZE	USEB		
00025	4320 00 0 00047	10001		ANA	MASK1		
00026	0100 00 0 00035	10001		TZE	BIGXP		
00027	0500 00 0 00046	10001		CLA	C		
00030	4300 00 0 00045	10001		UFA	B		
00031	0760 00 0 00011	10000		FRN			
00032	4100 00 0 00043	10001		TNZ	RTN		
00033	0500 00 0 00045	10001		CLA	B		
00034	0020 00 0 00043	10001		TRA	RTN		
00035	4500 00 0 00045	10001	BIGXP	CAL	B		
00036	4400 00 0 00046	10001		SBM	C		
00037	4120 00 0 00042	10001		TMI	USEC		
00040	0500 00 0 00045	10001	USEB	CLA	B		
00041	0020 00 0 00043	10001		TRA	RTN		
00042	0500 00 0 00046	10001	USEC	CLA	C		
00043	0774 00 4 00000	10000	RTN	AXT	**,4		
00044	0020 00 4 00001	10000		TRA	1,4		
00045	0000 00 0 00000	10000	B	HTR	0		
BINARY CARD ID. SIQA0004							
00046	0000 00 0 00000	10000	C	HTR	0		
00047	000777777777	10000	MASK1	OCT	000777777777		
00050	777000000000	10000		*LORG			
	00000	01111		END			

SIGSUB
ASSEMBLED TEXT.

```
                    $TEXT  SIGSUB

                                                        ENTRY    SGSUB

BINARY CARD ID. SIGS0002
     00000  0631 00 0 00046  10001   SGSUB  STO      B
     00001  0634 00 4 00044  10001          SXA      RTN,4
     00002  0500 00 4 77774  10000          CLA      -4,4
     00003  0621 00 0 00006  10001          STA      FETCH
     00004  0625 00 0 00006  10001          STT      FETCH
     00005  0522 00 4 00001  10000          XEC      1,4
     00006  0500 00 0 00000  10000  FETCH   CLA      **
     00007  0750 00 0 00002  10000          CHS
     00010  0631 00 0 00047  10001          STO      C
     00011  0500 00 0 00046  10001          CLA      B
     00012  0100 00 0 00043  10001          TZE      USEC
     00013  4320 00 0 00050  10001          ANA      MASK1
     00014  0100 00 0 00024  10001          TZE      TESTC
     00015  0500 00 0 00046  10001          CLA      B
     00016  4300 00 0 00047  10001          UFA      C
     00017  0750 00 0 00011  10000          FRN
     00020  4100 00 0 00044  10001          TNZ      RTN
     00021  0500 00 0 00047  10001          CLA      C
     00022  4320 00 0 00051  10001          ANA      #0777000000000

BINARY CARD ID. SIGS0003
     00023  0020 00 0 00044  10001          TRA      RTN
     00024  0500 00 0 00047  10001  TESTC   CLA      C
     00025  0100 00 0 00041  10001          TZE      USE8
     00026  4320 00 0 00050  10001          ANA      MASK1
     00027  0100 00 0 00036  10001          TZE      BIGXP
     00030  0500 00 0 00047  10001          CLA      C
     00031  4300 00 0 00046  10001          UFA      B
     00032  0750 00 0 00011  10000          FRN
     00033  4100 00 0 00044  10001          TNZ      RTN
     00034  0500 00 0 00046  10001          CLA      B
     00035  0020 00 0 00044  10001          TRA      RTN
     00036  4500 00 0 00046  10001  BIGXP   CAL      B
     00037  4400 00 0 00047  10001          SBM      C
     00040  4120 00 0 00043  10001          TMI      USEC
     00041  0500 00 0 00046  10001  USE8    CLA      B
     00042  0020 00 0 00044  10001          TRA      RTN
     00043  0500 00 0 00047  10001  USEC    CLA      C
     00044  0774 00 4 00000  10000  RTN     AXT      **,4
     00045  0020 00 4 00001  10000          TRA      1,4

BINARY CARD ID. SIGS0004
     00046  0000 00 0 00000  10000  B       HTR      0
     00047  0000 00 0 00000  10000  C       HTR      0
     00050  000777777777     10000  MASK1   OCT      000777777777
     00051  777000000000     10000          *LORG
                      00000  01111          END
```

SIGMPY
ASSEMBLED TEXT.

$TEXT SIGMPY

				ENTRY	SGMPY	
BINARY CARD ID. SIGM0002						
00000	0634 00 4 00062	10001	SGMPY	SXA	RTN,4	
00001	0500 00 4 77774	10000		CLA	-4,4	
00002	0621 00 0 00005	10001		STA	INST	
00003	0625 00 0 00005	10001		STT	INST	
00004	0522 00 4 00001	10000		XEC	1,4	
00005	0500 00 0 00000	10000	INST	CLA	**	
00006	0601 00 0 00064	10001		STO	YSTO	
00007	4600 00 0 00065	10001		STQ	QSTO	
00010	0600 00 0 00070	10001		STZ	FLAG	
00011	0500 00 0 00065	10001		CLA	QSTO	
00012	0100 00 0 00035	10001		TZE	BLAP	
00013	4320 00 0 00071	10001		ANA	MASK1	
00014	4100 00 0 00022	10001		TNZ	TESTY	
00015	4500 00 0 00065	10001		CAL	QSTO	
00016	0402 00 0 00074	10001		SUB	ONEX	
00017	0400 00 0 00073	10001		ADD	ONE	
00020	0602 00 0 00065	10001		SLW	QSTO	
00021	0601 00 0 00070	10001		STO	FLAG	
00022	0500 00 0 00064	10001	TESTY	CLA	YSTO	
BINARY CARD ID. SIGM0003						
00023	0100 00 0 00034	10001		TZE	BLAP-1	
00024	4320 00 0 00071	10001		ANA	MASK1	
00025	4100 00 0 00035	10001		TNZ	BLAP	
00026	4500 00 0 00064	10001		CAL	YSTO	
00027	0402 00 0 00074	10001		SUB	ONEX	
00030	0400 00 0 00073	10001		ADD	ONE	
00031	0602 00 0 00064	10001		SLW	YSTO	
00032	0601 00 0 00070	10001		STO	FLAG	
00033	0020 00 0 00035	10001		TRA	BLAP	
00034	0600 00 0 00070	10001		STZ	FLAG	
00035	0074 00 4 03000	10011	BLAP	TSX	NRMCT,4	
00036	0 00000 0 00065	10001		PZE	QSTO	
00037	0 00000 0 00067	10001		PZE	QNO	
00040	0074 00 4 03000	10011		TSX	NRMCT,4	
00041	0 00000 0 00064	10001		PZE	YSTO	
00042	0 00000 0 00066	10001		PZE	YNO	
00043	0560 00 0 00065	10001		LDQ	QSTO	
00044	0260 00 0 00064	10001		FMP	YSTO	
00045	0760 00 0 00011	10000		FRN		
BINARY CARD ID. SIGM0004						
00046	0601 00 0 00064	10001		STO	YSTO	
00047	0074 00 4 04000	10011		TSX	DNORM,4	
00050	0 00000 0 00064	10001		PZE	YSTO	
00051	0 00000 0 00067	10001		PZE	QNO	
00052	0 00000 0 00066	10001		PZE	YNO	
00053	0500 00 0 00070	10001		CLA	FLAG	
00054	0100 00 0 00061	10001		TZE	ANS	
00055	4500 00 0 00064	10001		CAL	YSTO	
00056	4320 00 0 00072	10001		ANA	MASK2	

SIGMPY
ASSEMBLED TEXT.

```
00057  0430 00 0 00074  10001          ADD     ONEX
00060  0632 00 0 00064  10001          SLW     YSTO
00061  0500 00 0 00064  10001  ANS     CLA     YSTO
00062  0774 00 4 00000  10000  RTN     AXT     **,4
00063  0020 00 4 00001  10000          TRA     1,4
00064  0000 00 0 00000  10000  YSTO    HTR     0
00065  0000 00 0 00000  10000  QSTO    HTR     0
00066  0000 00 0 00000  10000  YNO     HTR     0
00067  0000 00 0 00000  10000  QNO     HTR     0
00070  0000 00 0 00000  10000  FLAG    HTR     0

BINARY CARD ID. SIGM0005
00071  000777777777     10000  MASK1   OCT     000777777777
00072  777000000000     10000  MASK2   OCT     777000000000
00073  000000000001     10000  ONE     OCT     000000000001
00074  001000000000     10000  ONEX    OCT     001000000000
                  00000  01111          END
```

SIGDVP
ASSEMBLED TEXT.

```
                    $TEXT  SIGDVP

                                                          ENTRY    SGDVP

BINARY CARD ID. SIGDJ002
        00000  J631 00 0 00363  10001   SGDVP  STO      DVND
        00001  J634 00 4 00361  10001          SXA      RTN,4
        00002  J530 00 4 77774  10000          CLA      -4,4
        00003  J621 00 0 00306  10001          STA      INST
        00004  J625 00 0 00006  10001          STT      INST
        00005  J522 00 4 00001  10000          XEC      1,4
        00006  J530 00 0 00000  10000   INST   CLA      **
        00007  J631 00 0 00364  10001          STO      DVSR
        00010  J630 00 0 00367  10001          STZ      FLAG
        00011  J530 00 0 00363  10001          CLA      DVND
        00012  J130 00 0 00335  10001          TZE      BLAP
        00013  4320 00 0 00370  10001          ANA      MASK1
        00014  4130 00 0 00322  10001          TNZ      TESTY
        00015  4530 00 0 00363  10001          CAL      DVND
        00016  J432 00 0 00373  10001          SUB      ONEX
        00017  J430 00 0 00372  10001          ADD      ONE
        00020  J632 00 0 00363  10001          SLW      DVND
        00021  J631 00 0 00367  10001          STO      FLAG
        00022  J530 00 0 00364  10001   TESTY  CLA      DVSR

BINARY CARD ID. SIGDJ003
        00023  J130 00 0 00334  10001          TZE      BLAP-1
        00024  4320 00 0 00370  10001          ANA      MASK1
        00025  4130 00 0 00335  10001          TNZ      BLAP
        00026  4530 00 0 00364  10001          CAL      DVSR
        00027  J432 00 0 00373  10001          SUB      ONEX
        00030  J430 00 0 00372  10001          ADD      ONE
        00031  J632 00 0 00364  10001          SLW      DVSR
        00032  J631 00 0 00367  10001          STO      FLAG
        00033  J020 00 0 00335  10001          TRA      BLAP
        00034  J630 00 0 00367  10001          STZ      FLAG
        00035  J074 00 4 03000  10011   BLAP   TSX      NRMCT,4
        00036  J 00000 0 00363  10001          PZE      DVND
        00037  J 00000 0 00365  10001          PZE      DVNNO
        00040  J074 00 4 03000  10011          TSX      NRMCT,4
        00041  J 00000 0 00364  10001          PZE      DVSR
        00042  J 00000 0 00366  10001          PZE      DVSNO
        00043  J530 00 0 00363  10001          CLA      DVND
        00044  J241 00 0 00364  10001          FDP      DVSR
        00045  4630 00 0 00364  10001          STQ      DVSR

BINARY CARD ID. SIGDJ004
        00046  J074 00 4 04000  10011          TSX      DNORM,4
        00047  J 00000 0 00364  10001          PZE      DVSR
        00050  J 00000 0 00365  10001          PZE      DVNNO
        00051  J 00000 0 00366  10001          PZE      DVSNO
        00052  J530 00 0 00367  10001          CLA      FLAG
        00053  J130 00 0 00360  10001          TZE      ANS
        00054  4530 00 0 00364  10001          CAL      DVSR
        00055  4320 00 0 00371  10001          ANA      MASK2
        00056  J430 00 0 00373  10001          ADD      ONEX
```

SIGDVP
ASSEMBLED TEXT.

```
00057  3632 00 0 00364  10001          SLW     DVSR
00060  3560 00 0 00364  10001  ANS     LDQ     DVSR
00061  3774 00 4 00300  10000  RTN     AXT     **,4
00062  3020 00 4 00001  10000          TRA     1,4
00063  0000 00 0 00000  10000  DVND    HTR     0
00064  0000 00 0 00000  10000  DVSR    HTR     0
00065  0000 00 0 00000  10000  DVNNO   HTR     0
00066  0000 00 0 00000  10000  DVSNO   HTR     0
00067  0000 00 0 00000  10000  FLAG    HTR     0
00070  000777777777     10000  MASK1   OCT     000777777777

BINARY CARD ID. SIGD0005
00071  777000000000     10000  MASK2   OCT     777000000000
00072  000000000001     10000  ONE     OCT     000000000001
00073  001000000000     10000  ONEX    OCT     001000000000
                  00000 01111          END
```

DNORML
ASSEMBLED TEXT.

```
                    $TEXT  DNORML

                                                          ENTRY    DNORM

BINARY CARD ID. DNOR0002
      00000  0500 60 4 00002  10000   DNORM  CLA*     2,4
      00001  0560 60 4 00003  10000          LDQ*     3,4
      00002  0040 00 0 01002  10011          TLQ      *+2
      00003  0131 00 0 00000  10000          XCA
      00004  0771 00 0 00033  10000          ARS      27
      00005  0621 00 0 00015  10001          STA      BACK
      00006  0601 00 0 00023  10001          STO      NCT
      00007  0500 00 0 00024  10001          CLA      #27
      00010  0402 00 0 00023  10001          SUB      NCT
      00011  0621 00 0 00016  10001          STA      BAKMO
      00012  0500 60 4 00001  10000          CLA*     1,4
      00013  4765 00 0 00033  10000          LGR      27
      00014  0361 00 0 00023  10001          ACL      NCT
      00015  0757 00 0 00000  10000   BACK   ALS      **
      00016  4763 00 0 00000  10000   BAKMO  LGL      **
      00017  4773 00 0 00033  10000          RQL      27
      00020  0760 00 0 00011  10000          FRN
      00021  0601 60 4 00001  10000          STO*     1,4
      00022  0020 00 4 00004  10000          TRA      4,4

BINARY CARD ID. DNOR0003
      00023  0000 00 0 00000  10000   NCT    HTR      0
      00024  000000000033     10000          *LORG
                         00000  01111         END
```

NRMCNT
ASSEMBLED TEXT.

```
                    $TEXT  NRMCNT

                                                        ENTRY    NRMCT

BINARY CARD ID. NRMCJ002
      00000  3530 60 4 00001   10000   NRMCT  CLA*     1,4
      00001  3631 00 0 00015   10001          STO      DSTO
      00002  3330 00 0 00017   10001          FAD      ZRO
      00003  3631 00 0 00016   10001          STO      DNSTO
      00004  3631 60 4 00001   10000          STO*     1,4
      00005  4530 00 0 00016   10001          CAL      DNSTO
      00006  4320 00 0 00020   10001          ANA      MASK
      00007  3632 00 0 00016   10001          SLW      DNSTO
      00010  4530 00 0 00015   10001          CAL      DSTO
      00011  4320 00 0 00020   10001          ANA      MASK
      00012  3432 00 0 00016   10001          SUB      DNSTO
      00013  3632 60 4 00002   10000          SLW*     2,4
      00014  3320 00 4 00003   10000          TRA      3,4
      00015  3030 00 0 00000   10000   DSTO   HTR      0
      00016  3030 00 0 00000   10000   DNSTO  HTR      0
      00017  3030303000000     10000   ZRO    DEC      0.0
      00020  3770303000000     10000   MASK   OCT      377000000000
                        00000  01111          END
```

Internal Distribution

1-3. L. S. Abbott
4-13. W. R. Burrus
14. V. R. Cain
15. G. T. Chapman
16. C. E. Clifford
17. D. C. Irving
18. F. C. Maienschein
19. R. W. Peelle
20-30. B. W. Rust
31. R. T. Santoro
32. D. K. Trubey
33. J. W. Wachter
34. W. Zobel
35. G. Dessauer (Consultant)
36. B. C. Diven (Consultant)
37. M. L. Goldberger (Consultant)
38. M. H. Kalos (Consultant)
39. L. V. Spencer (Consultant)
40-41. Central Research Library
42. Document Reference Section
43-223. Laboratory Records Department
224. Laboratory Records ORNL R.C.
225. ORNL Patent Office

External Distribution

226-240. Division of Technical Information Extension (DTIE)
241. Division of Research and Development (ORO)