

X-552-72-119

PREPRINT

NASA TM 65968

EXTENDED PRECISION SOFTWARE PACKAGES

(NASA-TM-X-65968)
SOFTWARE PACKAGES
Feb. 1972 36 p

EXTENDED PRECISION
E.J. Phillips (NASA)
CSCL 09B

N72-29164

G3/08

Unclas
37444

FEBRUARY 1972



GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

Reproduced by
**NATIONAL TECHNICAL
INFORMATION SERVICE**
U S Department of Commerce
Springfield VA 22151

EXTENDED PRECISION SOFTWARE PACKAGES

E. J. Phillips

**Computing and Software, Inc.
Data Systems Division**

February 1972

**GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland**

CONTENTS

	<u>Page</u>
INTRODUCTION	1
FLOP-PACKAGE	2
I. LANGUAGE	2
II. PURPOSE	2
III. METHOD	3
IV. ERROR MESSAGES	5
V. RESTRICTIONS	5
VOP-PACKAGE	5
I. LANGUAGE	5
II. PURPOSE	5
III. METHOD	6
IV. RESTRICTIONS	8
SFLOP-PACKAGE	8
I. LANGUAGE	8
II. PURPOSE	9
III. METHOD	9
IV. ERROR MESSAGES	15
V. RESTRICTIONS	16
SUBROUTINE NAME: EQUAL	16
I. LANGUAGE	16

CONTENTS (continued)

	<u>Page</u>
II. PURPOSE	16
III. METHOD	16
IV. INTERFACE INFORMATION	16
V. CALLED SUBROUTINES	17
VI. RESTRICTIONS	17
SUBROUTINE NAME: IEQUAL	17
I. LANGUAGE	17
II. PURPOSE	17
III. METHOD	17
IV. INTERFACE INFORMATION	18
V. CALLED SUBROUTINES	18
VI. RESTRICTIONS	18
SUBROUTINE NAME: DEQUAL	18
I. LANGUAGE	18
II. PURPOSE	18
III. METHOD	18
IV. INTERFACE INFORMATION	19
V. CALLED SUBROUTINES	19
VI. RESTRICTIONS	19
APPENDIX	20
REFERENCES	31

ACKNOWLEDGEMENT

The author wishes to acknowledge the contribution of the basic software by Mr. Thomas Kreifelts of the Mathematics and Data Processing Institute, Bonn, Germany and his help in implementing it on the IBM 360 system.

EXTENDED PRECISION SOFTWARE PACKAGES

INTRODUCTION

This document contains a description of three Extended Precision Packages along with three small conversion subroutines which can be used in conjunction with the Extended Precision Packages. The Extended Precision Packages were given to Goddard Space Flight Center by Mr. Thomas Kreifelts, Gesellschaft für Mathematik und Datenverarbeitung MBH Bonn, Germany, under an exchange agreement with Dr. C. E. Velez, Code 552 GSFC. The conversion programs were written by Dr. Paul Beaudet, Computer Science Corporation.

Modifications were made to the Extended Precision Packages along with benchmark tests under varying conditions by Computing and Software, Inc., personnel. These Extended Precision Packages are available in the GSFC library.

These Extended Precision Packages are software packages and make no use of Extended Precision Hardware. They are written in FORTRAN IV and they will run under 360/OS.

The three Extended Precision Packages are:

1. FLOP-PACKAGE — Normalized floating point arithmetic with symmetric rounding and arbitrary mantissa lengths.
2. VOP-PACKAGE — Unnormalized floating-point arithmetic with symmetric rounding and arbitrary mantissa lengths.
3. SFLOP-PACKAGE — Normalized floating-point interval arithmetic with appropriate rounding.

(Reference Moore for complete description of Interval Arithmetic.)⁽¹⁾

The purpose of an Extended Precision Package is to enable the user to use and manipulate numbers with large decimal places as well as those with small decimal places where precision beyond double precision is required; i. e., π (π) calculated to 96 decimal places can be used in all three packages. $\pi = 3.14159, 26535, 89793, 23846, 2643, 38327, 95028, 84197, 16939, 93751, 05820, 97494, 45923, 07816, 40628, 62089, 98628, 03482, 53421, 170.$ ⁽²⁾

Extended Precision numbers give more accurate results than double precision numbers because of the magnitude of the numbers.

The magnitude of the standard 64 bit (double precision) number is 0 or 16^{-65} (approximately 10^{-78}) through 16^{-63} (approximately 10^{-75}).⁽³⁾ The precision of a double precision number is 14 hexadecimal digits (approximately 16.8 decimal digits).

The magnitude of a 32 bit fixed point number (integer constant) is 214748 3647 (i.e., $2^{31}-1$).

The magnitude for an extended precision variable 10^{-65536} to 10^{65563} . The precision for an extended precision number is:

FLOP-PACKAGE — Infinite decimal digits
VOP-PACKAGE — 96 decimal digits
SFLOP-PACKAGE — 97 decimal digits

(In all three packages the decimal exponent cannot exceed 65536 in absolute value.)

The three conversion subroutines used with these packages are:

1. EQUAL — Converts a double precision variable into an extended precision variable array.
2. IEQUAL — Converts an integer into an extended precision variable array.
3. DEQUAL — Converts an extended precision variable array into a double precision word.

A detailed description of the three packages and the conversion subroutine follows.

FLOP-PACKAGE

I. LANGUAGE:

Fortran IV Level G or Level H

II. PURPOSE:

The objective is to perform arithmetic operations with precision numbers using normalized-floating-point arithmetic with symmetric rounding and arbitrary mantissa lengths.

III. METHOD:

- A. FLOP - numbers are represented by one dimensional arrays of short (16 bits) integers.

$$A \hat{=} IA(L + 3)$$

where L is the mantissa length of A. The representation is such that the following equation holds

$$A = IA(1) \cdot 10^{IA(3)} \cdot \sum_{j=4}^{IA(2)} IA(j) \cdot 10^{3-j}$$

IA(1) = sign(A), +1 positive, -1 negative

IA(2) = L + 3

IA(3) = decimal exponent of A (positive or negative)

IA(4) - IA(L + 3) = mantissa of A

Sign	Length	Decimal Exponent	Mantissa
IA(1)	IA(2)	IA(3)	IA(4) - IA(L + 3)

EXAMPLE:

3.141592653589793 in extended precision form is

$$\boxed{1 \ 19 \ 13 \ 14 \ 15 \ 9 \ 26 \ 53 \ 58 \ 97 \ 93} = .3141592653589793 \times 10^1$$

-65535.15778895432 is represented as

$$\boxed{-1 \ 19 \ 56 \ 55 \ 35 \ 15 \ 77 \ 88 \ 95 \ 43 \ 2} = -.655351778895432 \times 10^5$$

$$.2345 \times 10^{-3} = \boxed{1 \ 7 \ -3 \ 2 \ 3 \ 4 \ 5} = .0002345$$

$$.2345 \times 10^0 = \boxed{1 \ 7 \ 0 \ 2 \ 3 \ 4 \ 5} = .2345$$

FLOP uses an unlabelled common of length INTEGER (2L - 1) where L is the maximum mantissa length.

FLOP uses normalized floating point arithmetic. The representation of a non-zero number is normalized if the first (high order) hexadecimal

digit of its mantissa (fraction) is not zero. The process of normalization consists of shifting the mantissa (fraction) left until the high order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted.

FLOP uses what is termed post normalization. That is to say the input values may or may not be normalized but after the arithmetic operations are performed, normalization will occur. If normalization is needed, after shifting the mantissa left to obtain a nonzero digit in the high order byte, the right is padded with zeros to maintain the mantissa length.

A zero fraction (mantissa) can not be normalized. The arithmetic operations performed in the FLOP package are addition, subtraction, multiplication and division.

B. Interface Information

1. Subroutines which are available for the user: FLADD, FLSUB, FLMUL, FLDIV, FLOUT, FUMSP
2. Internal subprograms not to be called by the user: FCHECK (subroutine) IFDIA, IFLADI

Calling Sequence:

IA, IB and IC are FLOP numbers where IA and IB are inputs, IC is output.

CALL FLADD (IA, IB, IC)	IC: = IA + IB (addition)
CALL FLSUB (IA, IB, IC)	IC: = IA - IB (subtraction)
CALL FLMUL (IA, IB, IC)	IC: = IA * IB (multiplication)
CALL FLDIV (IA, IB, IC)	IC: = IA/IB (division)
CALL FUMSP (IA, IC)	IC: = IA
CALL FLOUT (IC)	IC is printed

i. e., if IC =

1	10	2	2	5	7	6	5	4	3	2
---	----	---	---	---	---	---	---	---	---	---

 it is printed out as:

(E.2) .25765432

C. Mathematics Used in FLOP Package

1. CALL FLADD (IA, IB, IC) IC: = IA + IB
2. CALL FLSUB (IA, IB, IC) IB(1) = -IB(1)
 IC: = IA + IB
3. CALL FLMUL (IA, IB, IC) IC: = IA * IB
4. CALL FLDIV (IA, IB, IC) IC: = IA/IB

IV. ERROR MESSAGES:

"UNEQUAL LENGTHS" — The mantissa lengths of two operands are not equal. They are made equal by assigning the length of the shortest mantissa to that of the largest mantissa length. Operation continue.

"ZERO DIVISOR" — Divisor is zero. No division takes place. Program continues.

V. RESTRICTIONS

1. Any combination of subroutine arguments may be identical with the exception of CALL FLSUB (IX, IX, IZ). This results in IZ: = -2 * IX.

2. There is no check for underflow or overflow. If a decimal exponent exceeds 65536 in absolute value during computations, results may be in error.

See Appendix for examples of test cases and further restrictions.

VOP-PACKAGE

I. LANGUAGE:

Fortran IV Level G and Level H

II. PURPOSE:

The purpose of the VOP-Package is to perform arithmetic operations on extended precision numbers using unnormalized floating-point arithmetic.

III. METHOD:

- A. VOP numbers are represented by one dimensional arrays of short (16 bits) integers.

$$A \hat{=} IA(L + 3)$$

where L is the mantissa (fraction) length of A. The representation is such that the following equation holds

$$A = IA(1) \cdot 10^{IA(3)} \cdot \sum_{j=4}^{IA(2)} IA(j) \cdot 10^{3-j}$$

IA(1) = sign(A), +1 = positive, -1 = negative

IA(2) = L + 3 (mantissa length plus 3; add 1 for sign field, 1 for length field, and 1 for decimal exponent field = 3)

IA(3) = decimal exponent of A

IA(4) - IA(L + 3) = mantissa of A.

Sign	Length	Decimal Exponent	Mantissa
IA(1)	IA(2)	IA(3)	IA(4) - IA(L + 3)

The arithmetic operations performed are addition, subtraction, multiplication, division, inversion.

B. Interface Information

Subroutines used in VOP-Package are: VADD, VSUB, VMULT, VDIV, VINV, VIMAL, VQTI, VUMSP, VLOUT, VCHECK, IVDIA, VNORM, VASUM

Calling Sequence:

IA, IB and IC are VOP numbers where IA and IB are inputs, IC is output.

CALL VADD (IA, IB, IC) IC: = IA + IB (addition)

CALL VSUB (IA, IB, IC) IC: = IA - IB (subtraction)

Caution: VSUB (IA, IA, IC) doesn't give zero (\emptyset) since it uses the sub-routine VADD it gives $-2 * IA$.

CALL VMULT (IA, IB, IC) IC: = IA * IB (multiplication)

CALL VDIV (IA, IB, IC) IC: = IA/IB (division)

CALL VINV (IA, IC) IC: = $\frac{1}{IA} = IA^{-1}$ (inversion)

(Done by an iterative procedure)

CALL VIMAL (K, IA, IC) IC: = K * IA, K-integer * 2
(multiplication)

CALL VQTI (M, N, L, IC) IC: = (M/N), M, N INTEGER * 2

L = Mantissa length + 3

IC is output

L - 3 is the mantissa length up to which the fraction of M/N is computed. if $10^{-1} \leq M/N$. If $10^{-1} > M/N$ normalization takes place and the resulting mantissa lengths will be smaller.

CALL VUMSP (IA, IC) IC: = IA

CALL VLOUT (IC) IC is printed

i. e. , 100.000 equals (E.3) .100000

C. Mathematics Used in VOP-Package

1. CALL VADD (IA, IB, IC) IC: = IA + IB

2. CALL VSUB (IA, IB, IC) IB(1) = -IB(1)
IC: = IA + IB

3. CALL VMULT (IA, IB, IC) IC: = IA * IB

4. CALL VINV (IB, IC) IC = $\frac{1}{IB}$ or IB^{-1}

5. CALL VDIV (IA, IB, IC) IC = IA * $\frac{1}{IB}$

6. CALL VQTI (M, N, L, IC) IC = M/N
7. CALL VIMAL (K, IA, IC) IC = K * IA

IV. RESTRICTIONS

1. Maximum mantissa length is 96.
2. Storage reservation for VOP numbers should be at least one short integer bigger than actually needed.

i. e., for computations with 10-digit-mantissa, the dimensions for VOP numbers should be at least:

INTEGER * 2 IA(14), IB(14)

3. The subroutine arguments should be different (otherwise incorrect results may occur).
4. There is no check for underflow or overflow. If a decimal exponent exceeds 65536 in absolute value during computations, results may be in error.
5. If an extended precision number is of the form .003 or .0055 where the fractional portion of the number is zero when trying to divide, an interrupt will occur because of the zero fraction portion. Therefore avoid the use of zero fractions.

i. e., instead of

1	6	0	0	0	5
---	---	---	---	---	---

 = .005

use

1	6	-2	5	00
---	---	----	---	----

 = .005 = .5 x 10⁻²

See Appendix for examples of test cases and further restrictions.

SFLOP-PACKAGE

I. LANGUAGE:

Fortran IV Level G or Level H

II. PURPOSE:

The SFLOP Package is used to perform arithmetic operations on extended precision numbers using normalized floating point interval arithmetic with symmetric rounding and arbitrary mantissa lengths.

III. METHOD:

- A. SFLOP numbers are represented by one dimensional arrays of short (16 bits) integers.

$$A = (\underline{A}, \underline{A}_j) = IA(2L + 5)$$

where L is the mantissa length of \underline{A} and \underline{A}_j .

In Detail:

IA(1) = sign (\underline{A}); ± 1 (+1 positive, -1 negative)

IA(2) = sign (\underline{A}_j); ± 1 (+1 positive, -1 negative)

IA(3) = L + 3 mantissa length of each $\underline{A}, \underline{A}_j + 3$

IA(4) = decimal exponent of \underline{A}

IA(5) = decimal exponent of \underline{A}_j

IA(6) - IA(L + 5) = mantissa of \underline{A}

IA(L + 6) - IA(2L + 5) = mantissa of \underline{A}_j

Sign \underline{A}	Sign \underline{A}_j	Length	Decimal Exponent \underline{A}	Decimal Exponent \underline{A}_j	Mantissa \underline{A}	Mantissa \underline{A}_j
IA(1)	IA(2)	IA(3)	IA(4)	IA(5)	IA(6) - IA(L + 5)	IA(L + 6) IA(2L + 5)

Example: (317.45, - 405.50) is represented by:

1	-1	8	3	3	3	1	7	4	5	4	0	5	5	0
---	----	---	---	---	---	---	---	---	---	---	---	---	---	---

SFLOP uses an unlabeled common of length 600.

SFLOP numbers work on the same order as complex numbers where the real and imaginary parts of a complex number have to occupy the same amount of storage space so does the left and right mantissas of a SFLOP number. The left mantissa of a SFLOP number say $\lfloor IA$ would be equivalent to the real part of a complex constant; the right mantissa $\lfloor IA \rfloor$ would be equivalent to the imaginary portion of the complex constant.

B. Interface Information

Operations with interval numbers are accomplished by the following subroutines: IA, IB, IC are SFLOP numbers. IA and IB are inputs, IC is output.

CALL SFLADD (IA, IB, IC)	IC: = IA + IB (addition)
CALL SFLNEG (IA, IC)	IC: = -IA (negation)
(Negates IA but also interchanges left and right mantissas)	
CALL SFLMUL (IA, IB, IC)	IC: = IA * IB (multiplication)
CALL SFLDIV (IA, IB, IC)	IC: = IA/IB (division)
CALL SFLABS (IA, IC)	IC: = $ IA $ (absolute value)
CALL SFLSEC (IA, IB, IC)	IC: = IA \cap IB (intersection) = (MAX(IA, IB), MIN(IA, IB))
CALL SFUMSP (IA, IC)	IC: = IA
CALL SFLOUT (IC)	IC is printed
	(sign) (E.XX) mantissa of $\lfloor IC$
	(sign) (E.XX) mantissa of $\lfloor IC \rfloor$

With the subroutine SUMSP one can convert the upper or lower bound of a SFLOP - interval number into a FLOP - format number and vice versa.

CALL SUMSP (ISA, IA, K, M)	1. IA: = ISA if K = 1, M = 1
ISA is a SFLOP number	2. IA: = ISA if K = 1, M = 2
IA is a FLOP number	3. ISA = IA if K = 2, M = 1
K, M INTEGER * 2	4. ISA: = IA if K = 2, M = 2

In case 1 the upper bound of a SFLOP number is converted to a FLOP number.

In case 2 the lower bound of a SFLOP number is converted to a FLOP number.

In case 3 a FLOP number is converted to the upper bound of a SFLOP number.

In case 4 a FLOP number is converted to the lower bound of a SFLOP number.

When calling the various subroutines all parameters passed must be short integers.

Example: CALL SUMSP(IA, IC, 1, 1) would produce an incorrect answer and possibly an imprecise intercept because SUMSP is expecting short (16 bits) integers and 1 is considered by default to be long (32 bits). Therefore one should use something of the following form:

L1 = 1

L2 = 2

CALL SUMSP(IA, IC, L1, L2) where IA is a SFLOP number, IC is a FLOP number.

The below subprograms are for internal use only.

SADD - Called by SFLADD to perform addition

SMUL - Called by SFLMUL to perform multiplication

SDIV - Called by SFLDIV to perform division

SCHECK - Called by SADD, SMUL, SDIV

SNULL - Called by SADD, SMUL (Return a value of 1 if mantissa zero, 2 if mantissa nonzero)

SVUMSP - Called by SADD (Assigns a FLOP or VOP number into an output array)

ISFDIA - Called by SMUL (performs actual multiplication)

ISFLAD - Called by SDIV (performs actual division)

C. Mathematics Used in SFLOP Package

1. CALL SFLADD (IA, IB, IC)

(1a) If high order byte of left mantissa ($\lfloor IA \rfloor \leq 0$) then $\lfloor IC \rfloor = \lfloor IB \rfloor$

If high order byte of left mantissa ($\lfloor IA \rfloor > 0$) and high order byte of left mantissa $\lfloor IB \rfloor \leq 0$ then $\lfloor IC \rfloor = \lfloor IA \rfloor$

(2a) If high order byte of right mantissa ($IA_j \leq 0$) then $IC_j = IB_j$

If high order byte of right mantissa ($IA_j > 0$) and high order byte of right mantissa ($IB_j \leq 0$) then $IC_j = IA_j$

(3a) If not (1a) or (2a) then

$$\lfloor IC \rfloor = \lfloor IA \rfloor + \lfloor IB \rfloor$$

$$IC_j = IA_j + IB_j$$

2. CALL SFLNEG (IA, IC)

$$\lfloor IC \rfloor = -\lfloor IA \rfloor$$

$$IC_j = \lfloor -IA \rfloor$$

3. CALL SFLDIV (IA, IB, IC)

If $IB(1) * IB(2) = -1$ no division takes place, "DIVISOR ZERO" message written.

(1) If $\lfloor IA \rfloor \leq 0$, $IA_j < 0$ and $\lfloor IB \rfloor \leq 0$ then

$$\lfloor IC \rfloor = IA_j / \lfloor IB \rfloor$$

$$IC_j = IA_j / IB_j$$

(2) If $\lfloor IA \rfloor \leq 0$, $IA_j < 0$ and $IB_j > 0$ then

$$\lfloor IC \rfloor = \lfloor IA \rfloor / \lfloor IB \rfloor$$

$$IC_j = IA_j / IB_j$$

(3) If $\lfloor IA \rfloor > 0$, $\lfloor IB \rfloor \leq 0$ then

$$\lfloor IC \rfloor = IA_j / IB_j$$

$$IC_j = \lfloor IA \rfloor / \lfloor IB \rfloor$$

(4) If $\lfloor IA \rfloor > 0$, $\lfloor IB \rfloor > 0$ then

$$\lfloor IC \rfloor = \lfloor IA \rfloor / IB_j$$

$$IC_j = IA_j / \lfloor IB \rfloor$$

(5) If $\lfloor A \leq 0, I_A \geq 0$ and $\lfloor B \leq 0$ then

$$\lfloor C = I_A / I_B$$

$$I_C = \lfloor A / \lfloor B$$

(6) If $\lfloor A \leq 0, I_A \geq 0$ and $\lfloor B > 0$ then

$$\lfloor C = \lfloor A / \lfloor B$$

$$I_C = I_A / \lfloor B$$

4. CALL SFISEC (IA, IB, IC)

$$\lfloor C = (\text{MAX}(\lfloor A, \lfloor B))$$

$$I_C = (\text{MIN}(I_A, I_B))$$

5. CALL SFLMUL (IA, IB, IC)

(1) If $\lfloor A \leq 0, I_A < 0, \lfloor B \leq 0, I_B < 0$

$$\lfloor C = I_A * I_B$$

$$I_C = \lfloor A * \lfloor B$$

(2) If $\lfloor A \leq 0, I_A < 0, \lfloor B \leq 0, I_B \geq 0$

$$\lfloor C = \lfloor A * I_B$$

$$I_C = \lfloor A * \lfloor B$$

(3) If $\lfloor A \leq 0, I_A \geq 0, \lfloor B \leq 0, I_B \leq 0$

$$\lfloor C = I_A * \lfloor B$$

$$I_C = \lfloor A * I_B$$

(4) If $\lfloor A \leq 0, I_A \geq 0, \lfloor B \leq 0, I_B > 0$

$$\lfloor C = I_A * \lfloor B$$

$$I_C = I_A * I_B$$

(5) If $\lfloor A \leq 0, I_A \geq 0, \lfloor B > 0, I_B \geq 0$

$$\lfloor C = \lfloor A * \lfloor B$$

$$I_C = I_A * I_B$$

(6) $\lfloor A \leq 0, I_A \geq 0, \lfloor B > 0, I_B < 0$

(6.1) Then if $I_A(6) \neq 0$, "ILLEGAL INTERVAL". Return to MAIN.

If $(I_B(L + 2)) \neq 0$ write above message. ELSE;

- $\underline{IC} = \underline{IA} * \underline{IB}$
 $IC_j = IA_j * IB_j$
- (7) $\underline{IA} \leq 0, IA_j < 0, \underline{IB} > 0, IB_j \geq 0$
 $\underline{IC} = \underline{IA} * \underline{IB}$
 $IC_j = IA_j * \underline{IB}$
- (8) $\underline{IA} \leq 0, IA_j < 0, \underline{IB} > 0, IB_j < 0$
 see (6.1)
- (9) $\underline{IA} > 0, IA_j \geq 0, \underline{IB} \leq 0, IB_j \leq 0$
 $\underline{IC} = IA_j * \underline{IB}$
 $IC_j = \underline{IA} * \underline{IB}$
- (10) $\underline{IA} > 0, IA_j \geq 0, \underline{IB} \leq 0, IB_j > 0$
 $\underline{IC} = IA_j * \underline{IB}$
 $IC_j = IA_j * \underline{IB}$
- (11) $\underline{IA} > 0, IA_j \geq 0, \underline{IB} > 0, IB_j \geq 0$
 $\underline{IC} = IA * IB$
 $IC_j = IA * IB$
- (12) $\underline{IA} > 0, IA_j \geq 0, \underline{IB} > 0, IB_j < 0$
 see (6.1)
- (13) $\underline{IA} > 0, IA_j < 0$
- (13.1) If $IA(6) \neq 0$ write "ILLEGAL INTERVAL." Return to MAIN.
- (13.2) If $IA(6) = 0$ then
- (13.3) If $IA(L+2) \neq 0$ then write message in (13.1).
- (13.4) ELSE; If $\underline{IB} \leq 0, IB_j \leq 0$ see (9)
 If $\underline{IB} \leq 0, IB_j > 0$ see (10)
 If $\underline{IB} > 0, IB_j \geq 0$ see (11)
 If $\underline{IB} > 0, IB_j < 0$ see (6.1)

6. CALL SFLABS (IA, IC)

- (1.0) Sign of \underline{IA} (IA(1)) is positive and sign of IA_j (IA(2)) is negative
- (1.1) If high order digit of \underline{IA} (IA(6)) $\neq \emptyset$ - "ILLEGAL INTERVAL"

- (1.2) If high order digit of $\lfloor IA$ ($IA(6)$) = \emptyset and high order digit of IA_j ($IA(L + 3)$) see (1.1)
- (1.3) If high order digit of both $\lfloor IA$ ($IA(6)$) and IA_j ($IA(L + 3)$) are zero then
- $\lfloor IC = \lfloor IA$
- $IC_j = IA_j$
- (2.0) Sign of $\lfloor IA$ is negative and sign of IA_j is positive
- (2.1) Decimal exponent of $\lfloor IA$ ($IA(4)$) greater than decimal exponent of IA_j ($IA(5)$)
- $IC(4) = IA(5)$
- $IC(5) = IA(5)$
- $IC(1) = 1$
- $\lfloor IC = \emptyset$
- $IC_j = \lfloor IA$
- (2.2) Decimal exponent of $\lfloor IA$ less than decimal exponent of IA_j .
- $IC(1) IC(2) = 1$
- $IC(4), IC(5) = IA(4)$
- $\lfloor IC = \emptyset$
- $IC_j = IA_j$
- (2.3) Decimal exponent of $\lfloor IA =$ decimal exponent of IA_j
- (2.3.1) $\lfloor IA < IA_j$ see (2.2)
- (2.3.2) $\lfloor IA > IA_j$ see (2.1)
- (2.3.3) $\lfloor IA = IA_j$ see (2.2)

IV. ERROR MESSAGES:

"UNEQUAL LENGTHS" — The mantissa length of two operands are not equal. They are made equal by assigning the lengths the shortest mantissa to that of the longest. Operation continues.

"DIVISOR CONTAINS ZERO" — Divisor interval contains zero. No division takes place. Program continues.

"ILLEGAL INTERVAL" — Invalid combination of signs in an interval number is detected during multiplication or forming of an absolute value. Operation concerned does not take place. Program continues.

V. RESTRICTIONS

1. The maximum mantissa length is 97.

2. Any combination of subroutine arguments may be identical, except for subroutine SUMSP.

3. There is no check for underflow or overflow. If a decimal exponent exceeds 65536 in absolute value during computation, results may be in error.

See Appendix for further restrictions and test cases.

SUBROUTINE NAME: EQUAL

I. LANGUAGE:

Fortran IV Level G or Level H

II. PURPOSE:

The function of EQUAL is to convert a double precision variable to an extended precision variable array.

III. METHOD:

EQUAL takes a double precision variable X and converts it into an extended precision variable array IX of dimension L. The number of significant digits is L - 3.

Ex. If X = 2.00D0 and L = 10. IX =

1	10	1	2	0	0	0	0	0	0
---	----	---	---	---	---	---	---	---	---

IV. INTERFACE INFORMATION:

Calling Sequence:

CALL EQUAL (IX, N, X)

where: X is a double precision variable. IX is an extended precision variable array. N is the dimension of IX.

V. CALLED SUBROUTINES:

None.

VI. RESTRICTIONS:

1. The converted extended precision number is of the same form as the extended precision numbers used in FLOP and VOP packages. If the SFLOP package is being used a call to SUMSP will have to be made to create a SFLOP extended precision number.

2. The decimal exponent of the extended precision numbers should not exceed 10^{-78} or 10^{75} .

SUBROUTINE NAME: IEQUAL

I. LANGUAGE:

Fortran IV Level G or Level H

II. PURPOSE:

IEQUAL converts an integer into an extended precision variable array.

III. METHOD:

IEQUAL takes an integer (16 bits) J, and converts it into an extended precision variable array IX of dimension L. The number of significant digits is L - 3.

Ex. If J = 5 and L = 10

$$\text{IX} = \boxed{1 \mid 10 \mid 1 \mid 5 \mid 0 \mid 0 \mid 0 \mid 0 \mid 0} \text{IX} = .500000 \times 10^{-1}$$

IV. INTERFACE INFORMATION:

Calling Sequence:

CALL IEQUAL (IX, N, JX)

where: JX is an integer (16 bits). IX is an extended precision variable array. N is the dimension of IX. (N = length of mantissa + 3)

V. CALLED SUBROUTINES:

None.

VI. RESTRICTIONS:

1. The converted extended precision variable array is to be used with the FLOP and VOP packages. If SFLOP package is being used with the converted numbers SUMSP will have to be called to obtain the correct form for SFLOP package.

2. Decimal exponent of extended precision number should not exceed 10^{-78} or 10^{75} .

SUBROUTINE NAME: DEQUAL

I. LANGUAGE:

Fortran IV Level G or Level H

II. PURPOSE:

The function of DEQUAL is to convert an extended precision variable array to a double precision variable.

III. METHOD:

DEQUAL takes an extended precision variable array IX (N) and converts it into a double precision variable (X).

i. e.,

1	12	5	1	2	3	4	5	7	8	9	0
---	----	---	---	---	---	---	---	---	---	---	---

 = 12345.7890 or .123457190 x 10⁵
(extended precision)

= .123457890D + 05 (double precision)

IV. INTERFACE INFORMATION:

Calling Sequence:

CALL DEQUAL (X, IX)

where: IX is an extended precision variable array. X is a double precision word.

V. CALLED SUBROUTINES:

None.

VI. RESTRICTIONS:

None.

APPENDIX

RESTRICTIONS:

1. More time is required when processing extended precision numbers as opposed to double precision or integer numbers.
2. These programs are not on any existing library and the user will have to secure desired decks from GSFC library to use them.
3. Numbers have to be either defined in the users program in the extended precision form or converted into extended precision numbers by means of the two conversion programs EQUAL and IEQUAL.
4. Only the basic arithmetic functions are performed in these packages. No trigonometric functions are available.

SAMPLE TEST CASES:

1. Test case #1 illustrates the use of all the extended precision subroutines and conversion programs available to the user. The coding and output is shown. (p. 21-28)

2. Test case #2, part of a system written by Dr. Paul Beaudet, illustrates a program coded without extended precision subroutines and with them.

SIMQ is the subroutine using only double precision numbers. (p. 29)

ESIMQ is the subroutine utilizing some of the extended precision subroutines. (p. 30)

NOT REPRODUCIBLE

LEVEL 20.1 (AUG 71)

OS/360 FORTRAN H

COMPILER OPTIONS - NAME= MAIN,OPT=02,LINECNT=58,SIZE=0000K,
SOURCE,EBCDIC,NO&IST,NODECK,LOAD,MAP,NCECIT, ID,XREF

C
C DRIVER FOR EXTENDED PRECISION PACKAGES
C
C THERE ARE THREE PACKAGES
C 1. FLOP-PACKAGE DATE 72.089/16.11.13
C 2. SFLOP-PACKAGE
C 3. VOP-PACKAGE
C

ISN 0002 IMPLICIT REAL*8 (A-H,P-Z), INTEGER*2 (I-N)
ISN 0003 INTEGER*2 IA(200),IB(200),IC(200)
ISN 0004 CCMMGN X(600)
ISN 0005 L1=1
ISN 0006 L2=2
ISN 0007 L3=3
ISN 0008 M=125
ISN 0009 N=25
ISN 0010 L10=10
ISN 0011 L12=7
ISN 0012 L17=17
ISN 0013 IA(1)=1
ISN 0014 IA(2)=80
ISN 0015 IA(3)=1
ISN 0016 IA(4)=3
ISN 0017 IA(5)=1
ISN 0018 IA(6)=4
ISN 0019 IA(7)=1
ISN 0020 IA(8)=5
ISN 0021 IA(9)=9
ISN 0022 IA(10)=2
ISN 0023 IA(11)=6
ISN 0024 IA(12)=5
ISN 0025 IA(13)=3
ISN 0026 IA(14)=5
ISN 0027 IA(15)=8
ISN 0028 IA(16)=9
ISN 0029 IA(17)=7
ISN 0030 IA(18)=9
ISN 0031 IA(19)=3
ISN 0032 IA(20)=2
ISN 0033 IA(21)=3
ISN 0034 IA(22)=8
ISN 0035 IA(23)=4
ISN 0036 IA(24)=6
ISN 0037 IA(25)=2
ISN 0038 IA(26)=6
ISN 0039 IA(27)=4
ISN 0040 IA(28)=3
ISN 0041 IA(29)=3
ISN 0042 IA(30)=8
ISN 0043 IA(31)=3
ISN 0044 IA(32)=2
ISN 0045 IA(33)=7
ISN 0046 IA(34)=9
ISN 0047 IA(35)=5

ISN 004E	IA(36)=0
ISN 0049	IA(37)=2
ISN 0050	IA(38)=8
ISN 0051	IA(39)=8
ISN 0052	IA(40)=4
ISN 0053	IA(41)=1
ISN 0054	IA(42)=9
ISN 0055	IA(43)=7
ISN 0056	IA(44)=1
ISN 0057	IA(45)=6
ISN 0058	IA(46)=9
ISN 0059	IA(47)=3
ISN 0060	IA(48)=9
ISN 0061	IA(49)=9
ISN 0062	IA(50)=3
ISN 0063	IA(51)=7
ISN 0064	IA(52)=5
ISN 0065	IA(53)=1
ISN 0066	IA(54)=0
ISN 0067	IA(55)=5
ISN 0068	IA(56)=8
ISN 0069	IA(57)=2
ISN 0070	IA(58)=0
ISN 0071	IA(59)=9
ISN 0072	IA(60)=7
ISN 0073	IA(61)=4
ISN 0074	IA(62)=9
ISN 0075	IA(63)=4
ISN 0076	IA(64)=4
ISN 0077	IA(65)=5
ISN 0078	IA(66)=9
ISN 0079	IA(67)=2
ISN 0080	IA(68)=3
ISN 0081	IA(69)=0
ISN 0082	IA(70)=7
ISN 0083	IA(71)=8
ISN 0084	IA(72)=1
ISN 0085	IA(73)=6
ISN 0086	IA(74)=4
ISN 0087	IA(75)=0
ISN 0088	IA(76)=6
ISN 0089	IA(77)=2
ISN 0090	IA(78)=8
ISN 0091	IA(79)=6
ISN 0092	IA(80)=2
ISN 0093	IA(81)=0
ISN 0094	IA(82)=8
ISN 0095	IA(83)=9
ISN 0096	IA(84)=9
ISN 0097	IA(85)=8
ISN 0098	IA(86)=6
ISN 0099	IA(87)=2
ISN 0100	IA(88)=8
ISN 0101	IA(89)=0
ISN 0102	IA(90)=3
ISN 0103	IA(91)=4

ISN 0104	IA(92)=8
ISN 0105	IA(93)=2
ISN 0106	IA(94)=5
ISN 0107	IA(95)=3
ISN 0108	IA(96)=4
ISN 0109	IA(97)=8
ISN 0110	IA(98)=2
ISN 0111	A2=2.000
ISN 0112	LK=80
ISN 0113	CALL EQUAL(IB,LK,A2)
ISN 0114	WRITE(6,117) A2
ISN 0115	WRITE(6,121)
ISN 0116	CALL FLOUT(IA)
ISN 0117	WRITE(6,100)
ISN 0118	CALL FLOUT(IB)
ISN 0119	CALL FLADD(IA,IB,IC)
ISN 0120	WRITE(6,101)
ISN 0121	CALL FLOUT(IC)
ISN 0122	CALL FLSUB(IA,IB,IC)
ISN 0123	WRITE(6,102)
ISN 0124	CALL FLOUT(IC)
ISN 0125	CALL FLMUL(IA,IB,IC)
ISN 0126	WRITE(6,103)
ISN 0127	CALL FLOUT(IC)
ISN 0128	CALL FLDIV(IA,IB,IC)
ISN 0129	WRITE(6,104)
ISN 0130	CALL FLOUT(IC)
ISN 0131	CALL FUMSP(IA,IC)
ISN 0132	CALL SUMSP(IA,IC,L2,L2)
ISN 0133	CALL SUMSP(IA,IB,L2,L1)
ISN 0134	CALL SFUMSP(IA,IB)
ISN 0135	WRITE(6,10)
ISN 0136	CALL SFLOUT(IA)
ISN 0137	WRITE(6,100)
ISN 0138	CALL SFLOUT(IB)
ISN 0139	CALL SFLNEG(IB,IB)
ISN 0140	CALL SFLADD(IA,IB,IC)
ISN 0141	WRITE(6,102)
ISN 0142	CALL SFLOUT(IC)
ISN 0143	CALL SFLNEG(IB,IB)
ISN 0144	CALL SFLADD(IA,IB,IC)
ISN 0145	WRITE(6,101)
ISN 0146	CALL SFLOUT(IC)
ISN 0147	CALL SFLMUL(IA,IB,IC)
ISN 0148	WRITE(6,103)
ISN 0149	CALL SFLOUT(IC)
ISN 0150	CALL SFLABS(IB,IB)
ISN 0151	WRITE(6,105)
ISN 0152	CALL SFLOUT(IB)
ISN 0153	CALL SFLSEC(IA,IB,IC)
ISN 0154	WRITE(6,106)
ISN 0155	CALL SFLOUT(IC)
ISN 0156	CALL SFLNEG(IE,IB)
ISN 0157	CALL SUMSP(IA,IC,L1,L1)
ISN 0158	CALL SUMSP(IB,IA,L1,L1)
ISN 0159	CALL VUMSP(IC,IB)

```

ISN 0160      CALL DEQUAL(AA,IA)
ISN 0161      CALL DEQUAL(AB,IB)
ISN 0162      WRITE(6,128)
ISN 0163      WRITE(6,123)
ISN 0164      CALL FLOUT(IA)
ISN 0165      WRITE(6,100)
ISN 0166      CALL FLOUT(IB)
ISN 0167      WRITE(6,127) AA,AB
ISN 0168      CALL IEQUAL(IA,L12,N)
ISN 0169      CALL IEQUAL(IE,L12,N)
ISN 0170      WRITE(6,118)
ISN 0171      WRITE(6,124) M,N
ISN 0172      WRITE(6,122)
ISN 0173      CALL VLOUT(IA)
ISN 0174      WRITE(6,100)
ISN 0175      CALL VLOUT(IB)
ISN 0176      WRITE(6,101)
ISN 0177      CALL VADD(IA,IB,IC)
ISN 0178      CALL VLOUT(IC)
ISN 0179      CALL VSUB(IA,IB,IC)
ISN 0180      WRITE(6,102)
ISN 0181      CALL VLGUT(IC)
ISN 0182      CALL VMULT(IA,IB,IC)
ISN 0183      WRITE(6,103)
ISN 0184      CALL VLQUT(IC)
ISN 0185      CALL VDIV(IA,IB,IC)
ISN 0186      WRITE(6,104)
ISN 0187      CALL VLGUT(IC)
ISN 0188      CALL VIMAL(L2,IA,IC)
ISN 0189      WRITE(6,107)
ISN 0190      CALL VLOUT(IC)
ISN 0191      CALL VGTI(L3,L17,L10,IC)
ISN 0192      WRITE(6,105)
ISN 0193      CALL VLQUT(IC)
ISN 0194      10  FORMAT('IS-PACKAGE'/'0IA')
ISN 0195      11  FORMAT('IF-PACKAGE'/'0IA')
ISN 0196      12  FORMAT('IV-PACKAGE'/'0IA')
ISN 0197      100 FORMAT('0IB')
ISN 0198      101 FORMAT('0IA + IB')
ISN 0199      102 FORMAT('0IA - IB')
ISN 0200      103 FORMAT('0IA * IB')
ISN 0201      104 FORMAT('0IA / IB')
ISN 0202      105 FORMAT('0ABS(IB)
ISN 0203      106 FORMAT('0IA ISEC ABS(IB)
ISN 0204      107 FORMAT('02 * IA
ISN 0205      108 FORMAT('03/17
ISN 0206      109 FORMAT('0IB+IB')
ISN 0207      110 FORMAT('0IB/IA')
ISN 0208      111 FORMAT('0IA*IA')
ISN 0209      112 FORMAT('0IA/IA')
ISN 0210      113 FORMAT('0IA+IA')
ISN 0211      114 FORMAT('0IB-IA')
ISN 0212      115 FORMAT('0IB*IB')
ISN 0213      116 FORMAT('0IB/IB')
ISN 0214      117 FORMAT('1DOUBLE PRECISION TO EXTENDED PRECISION'/'0A2= '0.016.8)
ISN 0215      118 FORMAT('1INTEGER-TO-EXTENDED PRECISION')

ISN 0216      119 FORMAT('0IB-IB')
ISN 0217      120 FORMAT('0S-PACKAGE'/'0IA')
ISN 0218      121 FORMAT('0F-PACKAGE'/'0IA')
ISN 0219      122 FORMAT('0V-PACKAGE'/'0IA')
ISN 0220      123 FORMAT('0IA')
ISN 0221      124 FORMAT('0M= '0.17.' N= '0.17)
ISN 0222      127 FORMAT('0AA= '016.8.' AB= '0.016.8)
ISN 0223      128 FORMAT('1EXTENDED PRECISION TO DOUBLE PRECISION')
ISN 0224      STOP
ISN 0225      END

```

PAGE 004

PAGE 005

DOUBLE PRECISION TO EXTENDED PRECISION

A2= 0.20000000 01

F-PACKAGE

IA

(E 1) .31415926535897932384626433832795028841971693993751058209749445923078164062862

IB

(E 1) .1555555555555559777553550749686915152736663818359375000000000000000000

IA + IB

(E 1) .51415526535897930164180334582481947994708357812110433209749445923078164062862

IA - IE

(E 1) .11415526535897934605072483083108109689235030175391683209749445923078164062862

IA * IB

(E 1) .622318530717558577935151648326258725461578801557096445915412456875521968155

IA / IB

(E 1) .15707963267948567536247465920713657776201966990637740265075303361147605736000

S-PACKAGE

IA

(E 1).15595555595597775553550749686919152736663818359375000000000000000000000
(E 1).31415926535857932384626433832795028841971693993751058209749445923078164062862

IB

(E 1).15595555595597779553550749686919152736663818359375000000000000000000000
(E 1).31415926535897932384626433832795028841971693993751058209749445923078164062862

IA - IE

-(E 1).114155265358575346050724830E3108109689235030175391683209745445923078164062862
(E 1).11415526535897534605072483083108109689235030175391683209749445923078164062862

IA + IE

(E 1).35595555595595914750145537383830547332763671875000000000000000000000000
(E 1).62831853071795E64769252867665590057683943387987502116419458891846156328125724

IA * IE

(E 1).355955555955991118215802598748169649012418405815882330353301741393545754021
(E 1).986960440108935E618834490598761511353136994072407906264133493762200448224152

ABS(IB)

) SFLABS

(E 1).1559555559559597779553550749686919152736663818359375000000000000000000000
(E 1).31415926535857932384626433832795028841971693993751058209749445923078164062862

IA ISEC ABS(IB)

) SFLSEC

(E 1).1559555559559597779553550749686919152736663818359375000000000000000000000
(E 1).31415926535897932384626433832795028841971693993751058209749445923078164062862

EXTENDED PRECISION TC DOUBLE PRECISION

IA

-(E 1).31415926535897932384626433832795028841971693993751058209749445923078164062862

IB

(E 1).1559559959599977553556749686919152736663818359375000000000000000000

AA= -0.314159270 01 AB= 0.290000000 01

INTEGER-TO-EXTENDED PRECISION

M= 125 N= 25

V-PACKAGE

IA

(E 3).1250

IB

(E 2).2500

IA + IB

(E 3).1500

IA - IE

(E 3).1000

IA * IE

(E 4).3100

IA / IE

(E 1).5000

2 * IA

)VIMAL

(E 3).2500

3/17

)VQTI

(E 0).176470

```

COMPILER OPTIONS - NAME= MAIN,OPT=00,LINECNT=58,SIZE=0000K,
SOURCE,EBCDIC,NOLIST,NODECK,LOAD,MAP,NODEFIT,IO,XREF
ISN 0002 SUBROUTINE SIMQ(R,A,DET,NE,KS)
ISN 0003 DIMENSION B(1),A(1),BS(25),AS(25),X(25),XSAVE(25)
ISN 0004 DOUBLE PRECISION A,B,AS,BS,X,XSAVE,DET,SUM,S,S0
ISN 0005 IF (NE.EQ.1) GO TO 12
ISN 0006 NE2=NE**2
ISN 0007 S0=1./C70
ISN 0008 ITCH=0
ISN 0009 ITCHE=2
ISN 0010 IF(NE.GT.25) GO TO 1
ISN 0011 DO 3 I=1,NE
ISN 0012 X(I)=0.D0
ISN 0013 3 AS(I)=A(I)
ISN 0014 8 DO 2 I=1,NE2
ISN 0015 2 BS(I)=B(I)
ISN 0016 CALL SIMQ(BS,AS,DET,NE,KS)
ISN 0017 DO 4 I=1,NE
ISN 0018 4 X(I)=X(I)+AS(I)
ISN 0019 DO 5 I=1,NE
ISN 0020 DO 6 J=1,NE
ISN 0021 INDX=(J-1)*NE+I
ISN 0022 6 AS(J)=B(INDEX)*X(J)
ISN 0023 5 BS(I)=SUM(AS,NE)
ISN 0024 S=0.D0
ISN 0025 DO 7 I=1,NE
ISN 0026 AS(I)=A(I)-BS(I)
ISN 0027 7 S=S+AS(I)**2
ISN 0028 IF(S.LT.S0) GO TO 9
ISN 0029 ITCHE=ITCH+1
ISN 0030 IF(ITCH.LT.ITCHFS) GO TO 8
ISN 0031 DO 11 I=1,NE
ISN 0032 11 A(I)=XSAVE(I)
ISN 0033 RETURN
ISN 0034 9 DO 10 I=1,NE
ISN 0035 10 XSAVE(I)=X(I)
ISN 0036 S0=S
ISN 0037 ITCHE=0
ISN 0038 GO TO 8
ISN 0039 12 A(1)=A(1)/R(1)
ISN 0040 CFT=R(1)
ISN 0041 KS=0
ISN 0042 RETURN
ISN 0043 1 WRITE (6,20) NE2,NE
ISN 0044 20 FORMAT (' DIMENSION OF BS MUST BE INCREASED TO',I4,' AND AS, X TO',
ISN 0045 I,13,' IN SUBROUTINE SIMQ')
ISN 0046 RETURN
ISN 0047 END

```

```

      COMPILER OPTIONS - NAME= MAIN,OPT=00,LINECNT=58,SIZE=0000K,
                        SOURCE,EBCDIC,NOLIST,NODECK,LOAD,MAP,NOEDIT, ID,XREF
ISN 0002      SUBROUTINE ESIMQ (IB,IA,ID,NE,KS,NN)
ISN 0003      INTEGER*2 IBS(25200),IAS(1260),IX(1260),IXSAVE(1260),IB(1),IA(1),
                1 ID(1),IS(63),ISS(63)
ISN 0004      INTEGER*4 INE(20)
ISN 0005      DOUBLE PRECISION SO,S
ISN 0006      DO 12 I=1,NE
ISN 0007      12 INE(I)=NN*(I-1)+1
ISN 0008      IZ=0
ISN 0009      NE2=NE**2
ISN 0010      SO=1.070
ISN 0011      ITCH=0
ISN 0012      ITCHES=3
ISN 0013      IF(NE.GT.25) GO TO 1
ISN 0015      DO 3 I=1,NE
ISN 0016      II=NN*(I-1)+1
ISN 0017      CALL IEQUAL (IX(II),NN,IZ)
ISN 0018      3 CALL FUMSP (IA(II),IAS(II))
ISN 0019      8 DO 2 I=1,NE2
ISN 0020      II=NN*(I-1)+1
ISN 0021      2 CALL FUMSP (IB(II),IBS(II))
ISN 0022      CALL ESIMEQ(IBS,IAS,ID,NE,KS,NN)
ISN 0023      IF(KS.EQ.1) RETURN
ISN 0025      DO 4 I=1,NE
ISN 0026      II=NN*(I-1)+1
ISN 0027      4 CALL FLADD (IX(II),IAS(II),IX(II))
ISN 0028      DO 5 I=1,NE
ISN 0029      DO 6 J=1,NE
ISN 0030      INDEX=(J-1)*NE+I
ISN 0031      II=NN*(INDEX-1)+1
ISN 0032      JJ=NN*(J-1)+1
ISN 0033      6 CALL FLMUL (IB(II),IX(JJ),IAS(JJ))
ISN 0034      5 CALL ISUM (IBS,I,IAS,INE,NE)
ISN 0035      CALL IEQUAL (IS,NN,IZ)
ISN 0036      S=0.00
ISN 0037      DO 7 I=1,NE
ISN 0038      II=NN*(I-1)+1
ISN 0039      CALL FLSUB (IA(II),IBS(II),IAS(II))
ISN 0040      CALL FLMUL (IAS(II),IAS(II),ISS)
ISN 0041      7 CALL FLADD (IS,ISS,IS)
ISN 0042      CALL DEQUAL (S,IS)
ISN 0043      IF(S.LT.SO) GO TO 9
ISN 0045      ITCH=ITCH+1
ISN 0046      IF(ITCH.LT.ITCHES) GO TO 8
ISN 0048      DO 11 I=1,NE
ISN 0049      II=NN*(I-1)+1
ISN 0050      11 CALL FUMSP (IXSAVE(II),IA(II))
ISN 0051      RETURN
ISN 0052      9 DO 10 I=1,NE
ISN 0053      II=NN*(I-1)+1
ISN 0054      10 CALL FUMSP (IX(II),IXSAVE(II))
ISN 0055      SO=S
ISN 0056      ITCH=0
ISN 0057      GO TO 8

                                     PAGE 002

ISN 0058      1 WRITE (6,20) NE2,NE
ISN 0059      20 FORMAT ('* DIMENSION OF BS MUST BE INCREASED TO',I4,' AND AS, X TO'
                1,I3,'* IN SUBROUTINE SIMQ')
ISN 0060      RETURN
ISN 0061      END

```

REFERENCES:

1. Rall, L. B., [65]: Interval arithmetic and its application to error estimation and control. (Moore, R. E.), p. 61-130, Error in Digital Computation, Vol. 1, Wiley, New York.
2. "C.R.C. Standard Mathematical Tables," The Chemical Rubber Publishing Co., 12th Edition.
3. IBM System Reference Library, IBM System/360: FORTRAN IV Language, from C28-6515-6.
4. Ashenurst, R. L.; Metropolis, N., Unnormalized floating point arithmetic. J. Assoc. Comput. Mach. 6 (1959), 415-428. MR 4568, 21 (1960), 844.