PERKIN-ELMER

SYSTEM MATHEMATICAL RUN TIME LIBRARY (RTL)

Reference Manual

48-025 F00R00

The Perkin-Elmer Corporation, Computer Systems Division 2 Crescent Place, Oceanport, New Jersey 07757

Printed in the United States of America

PREFACE

This manual describes the 32-bit run time library (RTL) mathematical functions.

Chapter 1 introduces the RTL mathematical functions. Chapter 2 gives the needed information for all subprograms. Chapter 3 details elementary, min and max, and miscellaneous functions. The appendixes present information on error messages and the subprograms documented in the manual. Arithmetic tables are also included.

The R03 revision of this manual provides more information on the exponentiation routines.

For a survey of the Run Time Library and more information on the construction of programs in the Run Time Library, refer to the following Perkin-Elmer publication:

- FORTRAN VII Run Time Library (RTL) Introduction and Overview Reference Manual,
  Publication Number 29-578.

Additional information applicable to the Run Time Library is contained in the following Perkin-Elmer publications:

- FORTRAN VII Run Time Library (RTL) Language Extensions Reference Manual,
  Publication Number 29-580

- FORTRAN VII Run Time Library (RTL) Real Time Extensions Reference Manual,
  Publication Number 29-581

- FORTRAN VII Reference Manual,
  Publication Number 48-017

The following manuals provide detailed information on related software:

- Common Assembly Language (CAL) Programming Reference Manual,
  Publication Number 29-640

- OS/32 Library Loader Reference Manual, Publication Number 48-020
- OS/32 Operator Reference Manual, Publication Number 29-574
- OS/32 Programmer Reference Manual, Publication Number 29-613

TABLE OF CONTENTS

CHAPTERS (Continued)

APPENDIXES

TABLES

CHAPTER 1

INTRODUCTION

This manual describes the mathematical subprogram Perkin-Elmer provides in the FORTRAN VII Run Time Library. Subprograms described include the Intrinsic and Basic External functions required by ANSI Standard X3J3/77 and the arithmetic routines which the compiler calls. See the FORTRAN VII Run Time Library Language Extensions Manual, Publication Number 29-580, for the subprograms which perform bit and byte manipulation and logical operations.

This manual provides the following information for each subprogram:

1. The purpose served by the subprogram.
2. The types of arguments and values.
3. The range of admissible arguments.
4. The accuracy (where relevant) of the result.
5. Error messages and possible interrupts.
6. A brief description of the algorithm.

The subprograms can be called by assembly language code. The user must be aware of which arguments are passed to subprograms and which values are returned. See the FORTRAN VII Run Time Library Introduction and Overview, Publication Number 29-578.

Chapter 2 contains information pertinent to all of the subprograms. Definitions are given to fix the terminology. Accuracy of the different subprograms, ranges of admissible values, and the generation of error messages are described.

The Run Time Library provides descriptions of the function subprograms. These descriptions include external functions called by explicit statements in FORTRAN, intrinsic functions which can be called explicitly or implicitly (e.g., type conversion), and the involution routines called as a result of compiler-generated code. The programs are grouped as follows:

Elementary functions

Square root:	SQRT,DSQRT,CSQRT,CDSQRT
Exponential:	EXP,DEXP,CEXP,CDEXP
Trigonometric:	SIN,DSIN,CSIN,CDSIN,COS,DCOS,CCOS,CDCOS, TAN,DTAN
Logarithm:	ALOG,ALOG10,DLOG,DLOG10,CLOG,CDLOG
Inverse Trigonometric:	ACOS,ASIN,ATAN,ATAN2,CACOS,DASIN,DATAN, DATAN2
Hyperbolic:	SINH,COSH,TANH,DSINH,DCOSH,DTANH

Min and Max functions:

AMAX0	AMAX1	AMIN0	MIN0	MAX1
AMIN1	DMAX1	DMIN1	MAX0	MIN1

Type conversion subprograms:

DBLE	FLOAT	IDINT	DFLOAT	
IFIX	INT	INT2	SNGL	REAL

Miscellaneous functions:

Remainder

AMOD	DMOD	MOD

Manipulation of complex numbers

AIMAG	DIMAG	CMPLX	CONJG	DCONJG

Absolute value

ABS	CABS	DABS	DCABS	IABS

Transfer of sign

DSIGN	ISIGN	SIGN

Positive difference

DIM	IDIM	DDIM

Truncation and rounding

AINT	DINT	ANINT	DNINT	NINT	IDNINT

Double-precision product of single-precision factors

DPROD

Involution (raising to powers):

.IXXI	.IXXR	.IXXD	.IXXC	.IXXCD
.RXXI	.RXXR	.RXXD	.RXXC	.RXXCD
.DXXI	.DXXR	.DXXD	.DXXC	.DXXCD
.CXXI	.CXXR	.CXXD	.CXXC	.CXXCD
.CDXXI	.CDXXR	.CDXXD	.CDXXC	.CDXCD

Complex arithmetic:

.CXMPY	.CDMPY	.CXDIV	.CDDIV

CHAPTER 2

GENERAL CONSIDERATIONS

2.1 TERMS RELATED TO ACCURACY

Several methods describe the accuracy of a computation in floating-point arithmetic. In this manual, the following terms are used:

Absolute Error: is the absolute value of the difference between the true value of a desired number and the value as computed.

Relative Error: is the ratio between the absolute error and the true value.

Generally, the relative error is the more useful of the two and better adapted to floating-point representation of numbers. To refer to a result as accurate "to five decimal places" is actually saying that the relative error is 1.0E-5 or better.

A statement of the relative accuracy of a computation is based on the assumption that the numerical data, such as the argument of a function, is perfectly accurate.

In practice, numerical data generally has errors. Therefore, it is important to know the manner in which these errors can grow in the course of a computation. This is discussed for each function or group of functions under the heading "Propagation of Errors."

Extended discussions are available for accuracy and error-propagation qualities of floating point programs. The reader can consult:

Fike, C.T., Computer Evaluation of Mathematical Functions, Prentice-Hall, Inc., 1968

Hart, J.F., et al, Computer Approximations, Wiley, 1968

Sterbenz, P., Floating Point Computation, Prentice-Hall

2.2 ACCURACY OF FUNCTION SUBPROGRAMS IN GENERAL

The subprograms in the Run Time Library can be classified by their use of integer or floating-point arithmetic and by the amount of computation they involve.

Floating-point computations: the function subprograms described under Elementary Functions generally return values with a relative accuracy better than 1.0E-7 for single-precision functions and better than 1.0E-16 for double-precision functions. There are exceptions to this rule, however, in the case of logarithms and trigonometric functions. See Paragraphs 3.2.3 and 3.2.4 for details.

The floating-point involution subprograms also have a relative accuracy of 1.0E-7 for single precision or 1.0E-16 for double precision. The other computational subprograms are accurate to a few units in the last significant bit (i.e., a relative error of 1.0E-7 or better in single precision and 1.0E-16 in double precision). The value 1.0E-7 also applies to the subroutines mentioned under Complex Arithmetic, Paragraph 3.7. An exception in the case of AMOD or DMOD is described under Remaindering, Paragraph 3.5.1.

Type conversion functions are accurate to one unit in the least significant bit. Those subprograms using only integer arithmetic are absolutely accurate. The same is true of subprograms which essentially do no arithmetic (such as AMIN1 and SIGN).

Ranges: The Basic External and Intrinsic function subprograms in the Run Time Library give meaningful results for all values of their arguments and are subject to the following range restrictions:

Square root:	The argument must be positive or zero.
Exponential:	The argument x must satisfy -65*alog(16) < x < 63*alog(16), i.e., -180.2<x<174.6
Logarithm:	The argument must be positive.
Trigonometric:	The argument of SIN, COS, or TAN must have an absolute value no greater than 16**6 = 16,777,216. The argument of DSIN, DCOS, or DTAN must have an absolute value no greater than 16**14 = 72,057,594,037,927,936.
Inverse trigonometric:	The argument of ASIN, ACOS, DASIN, or DACOS must be between -1.0 and 1.0.
Hyperbolic:	The argument of SINH, COSH, DSINH, or DCOSH must have an absolute value no greater than 175.0.
Complex functions:	Restrictions on the arguments of complex functions are described under the individual functions.
Type conversion:	The argument of IDINT, INT, or IFIX must be between -(2**31-1) and 2**31-1 (2**31 = 2,147,483,648). The argument of INT2, must be between -32,768 and 32,767.

Miscellaneous functions: The second argument of AMOD, DMOD, or MOD, must be nonzero. The functions DSIGN, and SIGN require the second argument to be non-negative when the first argument is zero.

Involution routines: The integer, real, double-precision, and complex routines reject a zero base with a negative exponent. The real and double-precision routines reject a negative base.

2.3 GENERATION OF ERROR MESSAGES

Even if arguments satisfy the restrictions detailed under Ranges, they can lead to arithmetic underflow or overflow. When these conditions do occur, or when it is known that they will occur, the Run Time Library function subprograms cause an error message to be printed. The same happens if inadmissible arguments are passed to the subprogram. Details on specific error messages are given for each subprogram under Description of Function Subprograms. The error messages are of the form:

ERROR n IN m AT a: d

where: n identifies the diagnostic message,

m is the name of the subprogram,

a is the hexadecimal address of the point in the user's program from which the RTL is called, and

d is a diagnostic message.

See Appendix 1 for operation of the error message facility and suggested response to error messages.

2.4 EXTERNAL AND INTRINSIC VERSIONS OF FUNCTIONS

Except for the involution and complex arithmetic routines, every mathematical function in the RTL exists in two versions, as an intrinsic function or as an external function. The external functions are identified by their purely alphabetical names. The arguments are passed via an argument list. The intrinsic functions have names preceded by an @. They also accept arguments via an argument list. Some intrinsic functions have argument passage in registers. Their names are preceded by a dot. Hence, certain functions in the RTL have 3 different versions, e.g. COS, @COS and .COS.

CHAPTER 3

DESCRIPTION OF FUNCTION SUBPROGRAMS

3.1 INTRODUCTION

This section contains information on each function subprogram and it covers these topics: the purpose served by each function, the range of argument values which it accepts, considerations of accuracy and propagation of errors, an explanation of run-time error messages, subprograms called by a function, and a brief description of the algorithm is included for the more complicated mathematical functions.

When a function is in both the intrinsic and the external versions, the external version calls the intrinsic version to perform the actual computation. Therefore, the references to the function subprograms use the names of the intrinsic versions. Usually, the algorithms for single precision and double precision are similar. To describe them concisely, the function name and its arguments written in lower-case letters are generic names. A reference for nearly all algorithms is the book by Hart, cited earlier in this manual. The largest number representable in floating-point notation is approximately 16**63 or 4.0E75. This number (whether in single or double precision) is referred to as "OMEGA". Small values of the argument are frequently a special case and dependent upon the level of precision. The notation "eps" denotes 16**(-3) in single precision and 16**(-7) in double precision:

EPS=Y`3E100000´ in single precision
EPS=Y`3A100000´, Y`00000000´ in double precision.

The type Integer will always mean either Integer*2 or Integer*4.

3.2 ELEMENTARY FUNCTIONS

3.2.1 Square Root

	Argument	Value
Y=SQRT (X)	Real	Real
DY=DSQRT(DX)	Double precision	Double precision
CWW=CSQRT (CZZ)	Complex*8	Complex*8
CDWW=CDSQRT(CDZZ)	Complex*16	Complex*16

Purpose: To compute the square root of an argument of the appropriate type.

Range of arguments: For SQRT and DSQRT, the argument must not be negative. For CSQRT and CDSQRT, the argument may be any complex number.

Accuracy: SQRT and CSQRT are accurate to seven decimal places or better for all arguments. DSQRT and CDSQRT are accurate to 16 decimal places or better for all arguments.

Propagation of error: The relative error in SQRT(X) is approximately half the relative error in X. The only source of instability is the case where the absolute error in X is sizable compared to X.

Error messages: If a negative x is received, then SQRT and DSQRT print error messages "NEGATIVE ARGUMENT". The square root of the absolute value of x is returned to the called program.

Subroutines called: .CSQRT calls .CABS and .SQRT
.CDSQRT calls .CDABS and .DSQRT

Algorithms: .DSQRT and .SQRT use the same basic algorithm. If x=0.0, then no processing is performed. If x<0.0, then an error message is printed and x is replaced by its absolute value.

```
Let  x=(16**m)*u,        1/16≤u<1
Let  m=2*n+ip,           ip=0 or 1
     u=v*(.25**iq),      iq=0 or 1, 1/4≤v<1
```

A rational approximation, wa, to sqrt(v) is found first (Hart, formulas 0291 or 0293). The approximation is improved by Heron's formula:

wa is replaced by (v/wa+wa)*.5

In double precision, this formula is applied twice.

```
Let  k=2*ip-iq
Then sqrt(x)=(16**n)*(2**k)*wa
```

The algorithm for CSQRT and CDSQRT makes use of SQRT and DSQRT, respectively. Let sqrt(x+iy)=a+ib. If x=y=0, no processing is performed.

```
If x=0, then   a=sqrt(abs(y)/2)
               b=(y/abs(y))a

If y=0, x>0, then a=sqrt(x)
                  b=0
If y=0, x<0, then a=0
                  b=sqrt(-x)

Let r=cabs(x+iy)

If x>0, then   a=sqrt((x+r)/2)
               b=y/(2a)

If x<0, then   b=(y/abs(y))sqrt((-x+r)/2)
               a=y/(2b)
```

3.2.2 Exponential Functions

	Argument	Value
Y=EXP(X)	Real	Real
DY=DEXP(DX)	Double precision	Double precision
CWW=CEXP(CZZ)	Complex*8	Complex*8
CDWW=CDEXP(CDZZ)	Complex*16	Complex*16

Purpose: To compute exp(x) where x is a floating-point number of the appropriate type.

Range of arguments: For EXP and DEXP the argument must lie between -65*alog(16) and 63*alog(16). The interval from -180.0 to +174.0 is within these bounds. For CEXP and CDEXP, the real part must be between -180.0 and +174.0 and the imaginary part must have absolute value less than 16**6.

Accuracy: EXP and CEXP normally give results accurate to seven decimal places or better. When the imaginary part of a complex argument is very large, this accuracy degrades; see remarks in propagation of errors under Trigonometric Functions, Paragraph 3.2.3. DEXP and CDEXP give results accurate to 16 decimal places or better.

Propagation of errors: The relative error of EXP(X) is equal to the absolute error of X.

Error conditions: If the argument of EXP or DEXP is greater than the maximum acceptable value, then a message to that effect is put out and the number OMEGA is returned. If the argument of EXP or DEXP is less than the least acceptable value, then a different message is put out and 0.0 is returned. The same messages are put out if the real part of the argument of CEXP or CDEXP is out of the acceptable range. If the imaginary part of the argument of CEXP or CDEXP is unacceptable, another message is put out. CEXP returns the value (0,0) if the real part of the argument is less than the lower limit or the imaginary part is too large. If the imaginary part is acceptable, but the real part is above the upper limit, a large complex number is returned.

Subroutines called: .CEXP calls .EXP, .SIN, and .COS

.CDEXP calls .DEXP, .DSIN, and .DCOS

Algorithm:

1. EXP and DEXP.

   If abs(x)<eps, then 1+x is returned. If abs(x)>256 then it is treated as an error. Otherwise, let y=x*r, where
   r=2.0/alog(2.0)
   y=n+f, $0 \leq f < 1$

   If n>504 then x is above the upper limit; if n<-520 then x is below the lower limit. Otherwise, exp(x) can be evaluated without overflow as follows:

   Let g=f-0.5
   Let z=2**(g/2)

   z is found by a rational approximation (Hart, formulas 1080 or 1082).

   Let n=8*m+k, $0 \leq k \leq 7$
   s=2**(0.5*k+0.25)
   s is found from a table

   Finally:
   exp(x)=(16**m)*s*z

2. CEXP and CDEXP

   Let the two parts of the argument be (xr, xi)

   If $-180.0 \leq xr \leq 174.0$, let u=exp(xr)

   If xr<-180.0, let u=0
   If xr>174.0, let u=OMEGA

   If abs(xi)>16**6, let s=c=0.
   Otherwise, s=sin(xi), c=cos(xi)

   Then:
   cexp(xi, xr)=(u*c, u*s)

3.2.3 Trigonometric Functions

	Argument	Value
Y=SIN(X)	Real	Real
Y=COS(X)	Real	Real
Y=TAN(X)	Real	Real
DY=DSIN(DX)	Double precision	Double precision
DY=DCOS(DX)	Double precision	Double precision
DY=DTAN(DX)	Double precision	Double precision
CWW=CSIN(CZZ)	Complex*8	Complex*8
CWW=CCOS(CZZ)	Complex*8	Complex*8
CDWW=CDSIN(CDZZ)	Complex*16	Complex*16
CDWW=CDCOS(CDZZ)	Complex*16	Complex*16

Purpose:	To compute the trigonometric sine, cosine, or tangent (assuming radian measure) of an argument of the appropriate type.
Range of arguments:	For SIN, COS, and TAN, the argument must have an absolute value less than 16**6. For DSIN, DCOS, and DTAN the argument must have an absolute value less than 16**4. For CSIN, CDSIN, CCOS, and CDCOS, the argument must have a real part less in absolute value than 16**6 and an imaginary part between -175 and +175.
Accuracy:	Normally, SIN, COS, TAN, CSIN, and CCOS return values which are accurate to seven decimal places or better. DSIN, CDSIN, DCOS, and CDCOS return values accurate to 16 decimal places or better.
Propagation of errors:	The absolute error in SIN(X), COS(X), DSIN(X), and DCOS(X) is no greater on absolute value than the error in X. The absolute error in TAN(X) or DTAN(X) is the absolute error in X times COS(X)**(-2). Thus when COS(X) is small, TAN(X) is large and its evaluation can be unstable. For the complex functions, when the imaginary part of the argument differs from 0.0, the relative error in the value is proportioned to the absolute error in the argument.
Error conditions:	If the argument of SIN, COS, or TAN is greater in absolute value than 16**6, then an error message is put out and 0.0 is returned. If the argument of DSIN, DCOS, or DTAN is greater in absolute value than 16**14, then the same error message is put out and 0.0 is returned. If the argument of TAN or DTAN is such that function evaluation would lead to division by zero, then a different error message is put out and OMEGA is returned. If the real part of the argument of CSIN or CCOS has absolute value greater than 16**6, a message is put out and (0.0, 0.0) is returned. If the real part is acceptable, but the imaginary part is greater in absolute value than 175.0, a message is put out and a large complex number is returned.
Subroutines called:	.CSIN and .CCOS call .SIN, .COS, .SINH, and .COSH .CDSIN and .CDCOS call .DSIN, .DCOS, .DSINH, and .DCOSH

Algorithms:

1. Real and double-precision functions

If abs(x)<eps, then
sin(x)=x, cos(x)=1, tan(x)=x

If abs(x) is too large (see "Error Conditions"), then 0.0 is returned.

Otherwise let

ya=x*2*c where $c=2/\pi$
Set ya=k+u where $-1 \leq u < 1$.

Depending on k, the evaluation of the function of x reduces to one of these forms:

pm*sin (u* π /4)
pm*cos (u* π /4)
tan (u* π /4)
cot (u* π /4)

where pm=1 or -1

Polynomial or fractional approximations are used (Hart, formulae 3040, 3603, 3820, 3823, 4242, and 4245) to evaluate the above 4 formulas. If u=0 and cot(u* π /4) is required, then an error message is put out and OMEGA is returned.

2. Complex functions

Let the argument be cz=(x,y).

If abs(x)<16**6, let
rc=cos(x), rs=sin(x),
Otherwise rc=rs=0

If abs(y)<175.0, let
hc=cosh(y), hs=sinh(y);

If y>175.0, let
hc=OMEGA, hs=OMEGA;

If y<-175.0, let
hc=OMEGA, hs=-OMEGA

Then:
ccos(cz)=(rc*hc, -rs*hs)
csin(cz)=(rs*hc, rc*hs)

3.2.4 Logarithmic Functions

	Argument	Value
Y=ALOG(X)	Real	Real
Y=ALOG10(X)	Real	Real
DY=DLOG(DX)	Double precision	Double precision
DY=DLOG10(DX)	Double precision	Double precision
CWW=CLOG(CZZ)	Complex*8	Complex*8
CDWW=CDLOG(CDZZ)	Complex*16	Complex*16

Purpose: To compute the logarithm of a floating-point number of the appropriate type. ALOG10 and DLOG10 compute logarithms to the base 10; the others, to the base e (natural logarithms).

Range of arguments: ALOG, ALOG10, DLOG, and DLOG10 require positive (non-zero) arguments. CLOG and CDLOG require a non-zero complex argument.

Accuracy: ALOG, ALOG10, and CLOG return values with an absolute error of 1.0E-7 or less. DLOG and DLOG10 return values with an absolute error of 1.0E-16 or less.

Propagation of error: The absolute error of ALOG(X) equals the relative error of X. The absolute error of ALOG10(X) equals the relative error of X multiplied by ALOG10(e)=.434.

Error conditions: When the argument of a real or double-precision function is 0, an error message is printed and -OMEGA is returned. When the argument is negative, a different error message is printed and the logarithm of the absolute value is returned. When the argument of CLOG or CDLOG is (0.0, 0.0) an error message is printed and (-OMEGA 0.0) is returned.

Subroutines called: .ALOG1 branches to .ALOG, and .DLOG1 branches to .DLOG.

.CLOG calls .CABS, .ALOG1, and .ATAN2.

.CDLOG calls .CDABS, .DLOG1, and .DATN2.

Algorithms:

1. ALOG, ALOG10, DLOG, DLOG10.

   After stage (A) in the algorithm, the computation for base 10 is identical to that for base e. Therefore, .ALOG1 passes control to .ALOG, and .DLOG1 passes control to .DLOG.

(A) The error conditions are checked. The constant c is set to alog(2) in the programs .ALOG and .DLOG and to alog10(2) in the programs .ALOG1 and .DLOG1.

(B) Let the argument be x

$$x=(16^{**}m)*f,\ 1/16<f<1$$

Let $g=(2^{**}j)*f,\ 0\leq j\leq 3,\ .5\leq g\leq 1$

```
If g>sqrt(.5), let
     n=4*m-j
     h=(g-1)/(g+1);

If g>sqrt (.5), let
     n=4*m-j-1
     h=(g-.5)/(g+.5)
```

The rational function of h (Hart's formula 2701 or 2705) is found; the result is the logarithm to the base 2 of either g or 2*g, depending on whether g is greater or less than sqrt(.5). If this number is u, then v=n+u is the logarithm of x to the base 2. Multiplication by c gives the logarithm of x to the desired base.

2. CLOG and CDLOG

The error case (0.0, 0.0) is eliminated. If the argument (x,y) is not zero, let

```
u=alog(cabs(x,y))
v=atan2(y,x)
```

Then (u,v) is returned

3.2.5 Inverse Trigonometric Functions

	ARGUMENT	VALUE
Y=ACOS(X)	Real	Real
Y=ASIN(X)	Real	Real
Y=ATAN(X)	Real	Real
Y=ATAN2(X1, X2)	Real	Real
Y=DACOS (X)	Double precision	Double precision
Y=DASIN (X)	Double precision	Double precision
Y=DATAN (X)	Double precision	Double precision
Y2=DATAN2 (X1, X2)	Double precision	Double precision

Purpose:

To compute the inverse sine, cosine, or tangent of a number of the given type. The routines ATAN2 and DATAN2 compute the angle between an upward vertical line and the point with coordinates (x1,x2). When x2>0, this is atan(x1/x2). In general, the value of ATAN2 is between $-\pi$ and π.

Programming note:

The definition of the value of ATAN2 is that which the ANSI standard requires. To get a value between 0 and $2*\pi$, one can take:

π+atan2(-x1, -x2) when x1<0

Range of arguments:

ATAN and DATAN accept any argument of the correct type. ATAN2 and DATAN2 accept any pair of arguments other than (0.0, 0.0). ASIN, ACOS, DASIN, and DACOS accept any arguments between -1.0 and 1.0.

Accuracy:

The single-precision functions are accurate to seven places. The double-precision functions are accurate to 16 places.

Propagation of error:

Both absolute and relative errors are decreased by ATAN and DATAN. When X is large, the absolute error in ATAN (X) is very small. For ATAN2 and DATAN2, it is convenient to define the relative error of a pair of arguments as:

sqrt ((er1**2+er2**2)/(x1**2+x2**2))

where er1 and er2 are the errors in x1 and x2. Then, the absolute error in ATAN2 or DATAN2 is no greater than the relative error in the arguments. The functions ASIN, ACOS, DASIN, and DACOS increase the relative error; they become unstable as the argument nears 1.0 or -1.0.

Error conditions:

If ATAN2 and DATAN2 receive (0.0, 0.0) as arguments, they put out an error message and return $+\pi/4$. If ASIN, DASIN, ACOS, and DACOS receive an argument greater than 1.0 in absolute value, they put out an error message and return the value 0.0.

Subroutines called:

.ASIN and .ACOS call .SQRT. .DASIN and .DACOS call .DSQRT.

Algorithms:

1. ASIN, ACOS, DASIN, DACOS

If abs(x)<eps, then
asin(x)=x, acos(x)=π/2-x

If $-.5 \leq x \leq .5$, then asin(x) is found by a rational approximation (Hart, formula 4693 or 4698);
acos(x)=π/2-asin(x).

If $-1<x<-.5$, let

y=sqrt(0.5*x+0.5)
u=asin(y)
asin(x)=-π/2+2*u
acos(x)=π/2-2*u

If $.5<x \leq 1$, let

y=sqrt(-0.5*x+0.5)
u=asin(y)
asin(x)=π/2-2*u
acos(x)=2*u

2. ATAN, DATAN

If the argument is so large that the value of the function is $+\pi/2$ to the limit of precision involved, then that value is returned. Otherwise, the range is reduced by the following process. The argument belongs to one of seven intervals, the dividing points being:

-tan(75°), -tan(45°), -tan(15°), tan(15°), tan(45°), tan(75°). The argument is shifted to the range $-\tan(15°)<w<\tan(15°)$ by setting:

w=(x-a)/(1+ax)

where a=tan((30°)m), with $-3 \leq m \leq 3$ (in the extreme cases, the transformation is w=-1/x). Then atan(w) is evaluated; it is equal to w if abs(w)<eps, and otherwise w*P(w*w) where P is a polynomial. The coefficients of P are taken from Hart (No. 4940 for ATAN, No. 4945 for DATAN). Then,

atan(x)= atan(w)+m*(π/6).

The algorithms for ATAN2 and DATAN2 are like those for ATAN and DATAN with the exception of the following: the case (0.0,0.0) generates an error message as mentioned above; in the case (x1, 0.0), the value $+\pi/2$ is returned if x1>0.0 and $-\pi/2$ if x1<0; in the case (0.0,x2), the value 0.0 is returned if x2>0 and π if x2<0.

The exponents of the floating-point numbers x1 and, x2 are checked; and, if they differ by more than 6, the smaller is treated as 0.0 according to the previous paragraph. The exception is the case when x1=0.0 and x2>0; then x1/x2 is returned.

In the general case, the argument pairs are partitioned into sectors depending on the signs of the arguments and the size of their ratio. The evaluation is reduced to finding atan(w) where w<tan(15°). The value of atan(w) is computed exactly as in ATAN or DATAN.

3.2.6 Hyperbolic Functions

	ARGUMENT	VALUE
Y=SINH(X)	Real	Real
Y=COSH(X)	Real	Real
Y=TANH(X)	Real	Real
Y=DSINH(X)	Double precision	Double precision
Y=DCOSH(X)	Double precision	Double precision
Y=DTANH(X)	Double precision	Double precision

Range of arguments: SINH, COSH, DSINH, and DCOSH accept arguments between -175.0 and +175.0. TANH and DTANH accept all values.

Accuracy: The single-precision functions are accurate to seven decimal places; the double-precision functions are accurate to 16 decimal places.

Propagation of error: SINH multiplies the absolute error in X by COSH(X) and COSH multiplies it by SINH(X). When X is greater than 1 in absolute value, the relative error in SINH(X) or COSH(X) is close to the absolute error in X. TANH decreases both relative and absolute errors.

Error conditions: If the argument is greater than 175.0, then SINH and COSH put out an error message and return OMEGA. If the argument is less than -175.0, then SINH and COSH put out an error message and return, respectively, -OMEGA and OMEGA.

Subroutines called; .SINH and .COSH call .HYEXP and .TANH calls .EXP.

.DSINH and .DCOSH call .HYDEX, and .DTANH calls .DEXP.

Algorithms:

1. SINH and DSINH

If abs(x)<eps then sinh(x)=x

If abs(x)>175.0 then an error condition exists as described above.

If -175.0<x<-.5, let

```
xa=-x-alog(2)
 u=-exp(xa)
```

If u<-16**3 in single precision or u<-16**7 in double precision,

then

sinh(x)=u,

otherwise

sinh(x)=u-.25/u

If -.5<x<.5, a rational approximation is used (Hart, formula 2023 or 2028).

If .5<x<175.0,let

```
xa=x-alog(2)
 u=exp(xa)
```

If u>16**3 in single precision or u>16**7 in double precision,

then

sinh(x)=u,

otherwise

sinh(x)=u-.25/u

2. COSH and DCOSH

If x<0, replace x by -x
If x>175.0, an error condition exists as described above. If x<eps,

Then

```
cosh(x)=1.0
```

Otherwise let

```
xa=x-alog(2)
 u=exp(xa)
```

If u>16**3 in single precision or u>16**7 in double precision,

Then

```
cosh(x)=u
```

Otherwise

```
cosh(x)=u+.25u
```

3. TANH and DTANH

Let top=8.5 in single precision and 21.0 in double precision.

If x>top, tanh(x)=1
If x<-top, tanh(x)=-1
If .5*alog(2)<x<top, let

```
u=exp(2*x)
tanh(x)=(u-1)/(u+1)
```

If abs(x)<eps, then tanh(x)=x

If eps<abs(x)<.5*alog(2), a rational approximation is used.

If -top<x<-.5*alog(2), let

```
u=exp(-2*x)
tanh(x)=(1-u)/(1+u)
```

3.3 MIN AND MAX FUNCTIONS:

	ARGUMENT	VALUE
I=MAX0 (I1,...,In)	Integer	Integer*4
I=MIN0 (I1,...,In)	Integer	Integer*4
X=AMAX0 (I1,...,In)	Integer	Real
X=AMIN0 (I1,...,In)	Integer	Real
I=MAX1 (X1,...,Xn)	Real	Integer*4
I=MIN1 (X1,...,Xn)	Real	Integer*4
X=AMAX1 (X1,...,Xn)	Real	Real
X=AMIN1 (X1,...,Xn)	Real	Real
DX=DMAX1 (DX1,...,DXn)	Double	Double
DX=DMIN1 (DX1,...,DXn)	Double	Double

where 2<n<256

Purpose: To find the algebraic minimum or maximum of a collection of arguments of a given type and (in some cases) to convert to a different type. Types of arguments and values are shown in Table 3-1.

TABLE 3-1 TYPES OF MIN AND MAX ARGUMENTS

VALUE	ARGUMENT		
	INTEGER	REAL	DOUBLE PRECISION
INTEGER*4	MAX0 MIN0	MAX1 MIN1	
REAL	AMAX0 AMIN0	AMAX1 AMIN1	
DOUBLE PRECISION			DMAX1 DMIN1

Range of arguments:	Unrestricted, except for MAX1 and MIN1, which convert real numbers to integers. MAX1 and MIN1 require the chosen value to be between -2,147,483,648 and +2,147,483,647.
Accuracy:	Absolute.
Propagation of error:	Any error in the argument chosen as minimum or maximum is passed through unchanged.
Error messages:	If MAX1 or MIN1, try to convert a number too large for the appropriate type to fixed point, they print `OVERFLOW' and return the largest integer of the appropriate sign and type.

3.4 TYPE CONVERSIONS

Purpose: To convert a number from one type to another. The types of arguments and values are shown in Table 3-2.

Range of arguments: The functions with INTEGER*2 value require arguments between -32,768 and 32,767. The functions with INTEGER*4 value require arguments between -2,147,483,648 and +2,147,483,647.

Accuracy: SNGL truncates the least significant 32 bits of the fraction in a double-precision number. The functions which convert floating point to integer all truncate those bits which represent values less than one. For large arguments, FLOAT also truncates some bits; the result is accurate to the least significant bit in a floating point number.

Propagation of error: A small error in the argument can be entirely lost or it can result in a large error in the value, depending on whether or not it moves the argument from below an integer to above that integer. If the true value of X is 2.93, then INT(X)=2. If X is computed as 2.85 (an error of .08), then INT(X) is correctly computed as 2. But if X is computed as 3.01 (also an error of .08), then INT(X) is computed as 3.

Error messages: If the functions with integer values receive an argument which is too large, they print "OVERFLOW" and return the largest integer of the appropriate sign and type.

Remarks: The name INT2, INT, REAL, DBLE, and CMPLX are generic names for FORTRAN VII O and FORTRAN VII D. This means that INT2, INT, REAL, DBLE, and CMPLX can take any of the following types of arguments:

- Integer*2
- Integer*4
- Real*4
- Double Precision (Real*8)
- Complex*8
- Complex*16

Table 3-2 lists their names, types of arguments, and return values.

TABLE 3-2 TYPE CONVERSION ROUTINE NAMES, TYPES OF ARGUMENTS AND VALUES

Generic Name	Specific RTL Name	FORTRAN User Name	TYPE OF INPUT ARGUMENT		TYPE OF FUNCTION
			Arg 1	Optional Arg 2	
INT2	@@II2	INT2	Integer		Integer*2
	@INT2		Real		Integer*2
	@@DI2		Double		Integer*2
	@@CI2		Complex*8		Integer*2
	@@CDI2		Complex*16		Integer*2
INT	@@IINT	INT	Integer		Integer
	@INT		Real		Integer
	@IFIX	IFIX	Real		Integer
	@IDINT	IDINT	Double		Integer
	@@CINT		Complex*8		Integer
	@@CDI		Complex*16		Integer
REAL	@REAL	REAL	Integer		Real
	@FLOAT	FLOAT	Integer		Real
	@@RREA		Real		Real
	@SNGL	SNGL	Double		Real
	@@CREA		Complex*8		Real
	@@CDRE		Complex*16		Real
DBLE	@DFLOAT	DFLOAT	Integer		Double
	@DBLE	DBLE	Real		Double
	@@CDBL		Double		Double
	@@DDBL		Complex*8		Double
	@DREAL		Complex*16		Double
CMPLX	@@ICX	CMPLX	Integer	Integer	Complex*8
	@CMPLX		Real	Real	Complex*8
	@@DCX		Double	Double	Complex*8
	@@CCX		Complex*8	Complex*8	Complex*8
	@@CDCX		Complex*16		Complex*16
DCMPLX	@@ICD	DCMPLX	Integer	Integer	Complex*16
	@@RCD		Real*4	Real*4	Complex*16
	@DCMPL		Real*8	Real*8	Complex*16
	@@CXCD		Complex*8		Complex*16
	@@CDCD		Complex*16		Complex*16

CMPLX or DCMPLX takes 2 arguments of the same type, if the type is not complex. CMPLX or DCMPLX will accept only 1 argument of the complex type, in which case CMPLX just returns that argument.

A complex number CZ is stored as two real numbers (Z1, Z2) signifying (Z1) + (Z2) * SQRT(-1). The statement CZ = COMPLX(A1, A2) for A1, A2 of the same type which is not complex, results in making A1, A2 into 2 real numbers Z1, Z2 (by floating or truncation) and setting CZ = (Z1, Z2).

3.5 MISCELLANEOUS FUNCTIONS

3.5.1 Remaindering

	ARGUMENT	VALUE
Y=AMOD(X1,X2)	Real	Real
DY=DMOD(DX1,DX2)	Double precision	Double precision
K=MOD(I1,I2)	Integer	Integer*4

Purpose: To find the remainder as a result of dividing the first argument by the second.

Range of arguments: The second argument must not be zero. In the functions AMOD and DMOD, if the second argument is very small compared to the first, then a floating point overflow can occur.

Accuracy: MOD is precise. AMOD is accurate to about 1.0E-7 times the value of X2 and DMOD to about 1.0E-16 times the value of DX2. These accuracies can be degraded when X1/X2 becomes large. When X1/X2 is greater than 16**6, the returned result is meaningless.

Propagation of error: Absolute error in the first argument of AMOD or DMOD is passed through unchanged, unless it changes the integer part of X1/X2. In that case, it leads to an error of the same size as X2. Absolute error in the second argument is multiplied by the integer part of X1/X2; if it also causes a shift in that number, then an error the size of X2 is introduced.

Examples:

Let the true value of x be 10.0 and the true value of y, 91.0. Then amod (y,x)= 1.0. If y is computed as 92.1, then amod(y,x)=2.1; the error is passed along. If y is computed as 89.9 (same error in opposite direction), then amod(y,x)=9.9; a large error is introduced. If y is computed as 91.0 and x is computed as 10.1, then amod (y,x)=0.1; the error in x has been multiplied by y/x. If x is computed as 10.2, then amod (y,x)=9.4; a large error is introduced.

Error messages: If the second argument is 0, then an error message is printed and the value of the first argument is returned.

3.5.2 Manipulation of Complex Numbers

	ARGUMENT	VALUE
Y=AIMAG(CZ)	Complex*8	Real
R=CABS(CZ)	Complex*8	Real
CW=CONJG(CZ)	Complex*8	Complex*8
CDWW=DIMAG(CDZZ)	Complex*16	Complex*16
CDWW=CDABS(CDZZ)	Complex*16	Complex*16
CDWW=DCONJG(CDZZ)	Complex*16	Complex*16

Purpose: The complex number CZ is stored as two real numbers (Z1, Z2), signifying (Z1)+(Z2)SQRT(-1). AIMAG and DIMAG return Z2. CABS and CDABS return SQRT((Z1)*(Z1)+(Z2)*(Z2)). CONJG and DCONJG return (Z1, -Z2) as its value.

Range of arguments: Unrestricted.

Accuracy: Not applicable except to CABS, which is accurate to seven decimal places; and CDABS to sixteen decimal places.

Error messages: CABS and CDABS can cause arithmtic overflow.

Subroutines called: CABS calls .SQRT

CDABS calls .DSQRT

3.5.3 Absolute Values

	ARGUMENT	VALUE
Y=ABS(X)	Real	Real
DY=DABS(DX)	Double precision	Double precision
I=IABS(J)	Integer	Integer*4

Purpose: To return the absolute value of an appropriate argument.

Range of arguments: Unrestricted, except IABS return +2,147,483,647 for full scale negative number -2,147,483,648.

Accuracy: Not applicable.

Propagation of error: Not applicable.

Error messages: Not applicable.

Subroutines called: None.

3.5.4 Transfer of Sign

	ARGUMENTS	VALUE
K=ISIGN(I,J)	Integer	Integer*4
Z=SIGN(X,Y)	Real	Real
DZ=DSIGN(DX,DY)	Double precision	Double precision

Purpose: The second argument sign is applied to the absolute value of the first.

Range of arguments: The only restriction is the first argument must not be 0 if the second argument is negative.

Accuracy: Does not apply.

Propagation of error: Does not apply.

Remark: Zero is a positive number for these routines.

Error messages: If the first argument is 0 and the second argument is negative, then an error message is printed and 0 is returned.

Subroutines called: None.

3.5.5 Positive Difference

	ARGUMENT	VALUE
K=IDIM(I,J)	Integer	Integer*4
Z=DIM(X,Y)	Real	Real
DZ=DDIM(DX,DY)	Double precision	Double precision

Purpose: To give the value 0 or the value of the first argument minus the second, whichever is greater.

Range of arguments: Unrestricted.

Accuracy: To one bit in the least significant place.

Propagation of error: Not applicable.

Error messages: Not applicable.

Subroutines called: None.

3.5.6 Truncation and Rounding

	Argument	Value	Function
Y=AINT(X)	Real	Real	Truncation
Y=ANINT(X)	Real	Real	Rounding
DY=DINT(DX)	Double precision	Double precision	Truncation
DY=DNINT(DX)	Double precision	Double precision	Rounding
J=NINT(X)	Real	Integer	Rounding
J=IDNINT(DX)	Double precision	Integer	Rounding

To obtain integer values with truncation, use the type conversion routines IDINT, INT, or INT2.

Purpose: The Truncation functions remove the fractional part of a number; the result is equal to or smaller in absolute value. The Rounding functions return the nearest integer value. A fractional part of .5 is rounded up in absolute value; thus, NINT(1.5)=2 and NINT(-1.5)=-2.

Range: Unrestricted for AINT, ANINT, DINT, and DNINT.

For NINT and IDNINT, the value must be no greater than 2**31-1 in absolute value.

Accuracy: Not applicable.

Propagation of error: Not applicable.

Error messages: An overflow condition can be incurred by NINT or IDNINT.

3.5.7 Double-Precision Product of Single-Precision Factors

	Argument	Value
DY=DPROD(X1,X2)	Real	Double-precision

Purpose: The argument's product is computed as a double-precision number.

Range of Arguments: That of floating-point multiplication.

Accuracy: No error is introduced.

Propagation of error: The relative error in the product is the sum of the relative errors in the factors.

Error messages: Multiplicative overflow or underflow results in error messages.

Subroutines called: None.

3.6 INVOLUTION (RAISING TO POWERS)

Program	Base	Exponent	Value
.IXXI	Integer	Integer	Integer*4
.IXXR	Integer	Real	Real
.IXXD	Integer	Double Precision	Double Precision
.IXXC	Integer	Complex	Complex
.IXXCD	Integer	Complex*16	Complex*16
.RXXI	Real	Integer	Real
.RXXR	Real	Real	Real
.RXXD	Real	Double Precision	Double Precision
.RXXC	Real	Complex	Complex
.RXXCD	Real	Complex*16	Complex*16
.DXXI	Double Precision	Integer	Double Precision
.DXXR	Double Precision	Real	Double Precision
.DXXD	Double Precision	Double Precision	Double Precision
.DXXC	Double Precision	Complex	Complex*16
.DXXCD	Double Precision	Complex*16	Complex*16
.CXXI	Complex	Integer	Complex
.CXXR	Complex	Real	Complex
.CXXD	Complex	Double Precision	Complex*16
.CXXC	Complex	Complex	Complex
.CXXCD	Complex	Complex*16	Complex*16
.CDXXI	Complex*16	Integer	Complex*16
.CDXXR	Complex*16	Real	Complex*16
.CDXXD	Complex*16	Double Precision	Complex*16
.CDXXC	Complex*16	Complex	Complex*16
.CDXCD	Complex*16	Complex*16	Complex*16

Purpose: To evaluate expressions of the form a**b, (for complex involutions a**b is defined as the principal value determined by EXP(b*LOG(a))). All arithmetic types are permissable as bases and exponents. The interpretation of exponentiation operations on mixed mode data types is described in the FORTRAN VII Reference Manual.

Range of arguments: These following conditions are not acceptable: zero base with negative exponent; negative base with non-integer exponent.

Accuracy:

.IXXI introduces no error. .RXXI, .RXXR, and .CXXI are accurate to six decimal places. .DXXI, .DXXD, and .CDXXI are accurate to 14 decimal places.

Propagation of error:

The relative error includes: the relative error of the base multiplied by the exponent and the absolute error of the exponent multiplied by the natural logarithm of the base.

Interfaces:

Arguments and values are passed by loading them into registers as follows:

Program	Base	Exponent	Value
.IXXI	G13	G12	G13
.IXXR	G13	F14	F14
.IXXD	G13	D14	D14
.IXXC	G13	(F12,F14)	(F12,F14)
.IXXCD	G13	(D12,D14)	(D12,D14)
.RXXI	F14	G13	F14
.RXXR	F14	F12	F14
.RXXD	F14	D14	D14
.RXXC	F14	(F10,F12)	(F12,F14)
.RXXCD	F14	(D12,D14)	(D12,D14)
.DXXI	D14	G13	D14
.DXXR	D14	F14	D14
.DXXD	D14	D12	D14
.DXXC	D14	(F12,F14)	(D12,D14)
.DXXCD	D14	(D10,D12)	(D12,D14)
.CXXI	(F12,F14)	G13	(F12,F14)
.CXXR	(F12,F14)	F10	(F12,F14)
.CXXD	(F12,F14)	D14	(D12,D14)
.CXXC	(F12,F14)	(F8,F10)	(F12,F14)
.CXXCD	(F12,F14)	(D12,D14)	(D12,D14)
.CDXXI	(D12,D14)	G13	(D12,D14)
.CDXXR	(D12,D14)	F14	(D12,D14)
.CDXXD	(D12,D14)	D10	(D12,D14)
.CDXXC	(D12,D14)	(F12,F14)	(D12,D14)
.CDXCD	(D12,D14)	(D8,D10)	(D12,D14)

Error conditions:

Unacceptable combinations of arguments are detected and error messages are put out. For a zero base and a negative exponent, the largest positive number is returned. For a negative base and floating-point exponent, 0.0 is returned. The routines .IXXI, .RXXR, and .DXXD can anticipate certain occurrences of arithmetic underflow or overflow; .IXXI detects all such occurrences internally. The routines other than .IXXI can suffer exponential

underflows or overflows that cause arithmetic fault interrupts. In these cases, the user sees error messages from both the operating system and the Run Time Library.

If either a fixed-point overflow or a floating-point exponential overflow is detected, the value returned is the largest number of the appropriate sign. If an exponential underflow is detected, the value returned is 0.0.

Algorithm:

If the exponent is of integer type, the subroutine repeatedly squares the base and multiplies together those iterated squares that correspond to '1' bits in the binary expression of the exponent. The routines which receive real exponents of a single or double precision use the formula x**y=exp(y*alog(x)). If the exponent is 0, then the result 1 or 1.0 is invariably returned, whatever the value of the base.

Remarks:

These routines are called by the FORTRAN compiler to evaluate expressions of the form a**b. User can access these routines via assembly code.

APPENDIX 1
ERROR MESSAGES

If a mathematical RTL program detects an error, it calls a routine .ERR to print out an error message in this form:

ERROR n IN m AT a: d

where: n is the error message number,

m is the name of the RTL function,

a is the hexadecimal return address in the user's program, and

d is either a number or a diagnostic message.

The messages are listed by number in the error messages list in this appendix. The following comments are intended to help interpret the messages and correct their causes.

Messages Pertaining to the Argument List

When a mathematical RTL program is called by giving it an argument list with argument checking turned on, a series of messages can be put out as a result of errors in the list. **Please refer to FORTRAN VII Run Time Library Introduction and Overview, Publication Number 29-578 for argument checking.**

The error messages may be one of the following:

1. ILLEGAL CALL (72).
   A function has been called as a subroutine, e.g.

   ```
   CALL SQRT (....).
   ```

   The solution is to revise the text of the source program or, if the user means to call an external subroutine of that name, to ensure that it is brought in at linkage time.

2. INCORRECT TYPE OF VALUE OF FUNCTION (73).
   A function supplied in the RTL has been called as an external function and the user program has specified a value type that is incorrect. Example:

   ```
   IMPLICIT INTEGER (A-H)
   EXTERNAL COS
   A=COS(X)
   ```

   Check implicit and explicit types or include a user-written program of the appropriate name.

3. ARGUMENT OF INCORRECT CLASS (75).
   An argument has been classified as an array or external function and not a number. Check for the presence or absence of a DIMENSION statement, a misnamed argument, or an incorrect EXTERNAL statement.

4. AN ARGUMENT OF INCORRECT TYPE (45).
   The argument supplied is of the wrong type for this function. Check implicit and explicit type statements and types of expressions passed as arguments.

5. INCORRECT NUMBER OF ARGUMENTS (76).
   Check argument lists in source program and for the presence or absence of commas.

Arguments Outside Acceptable Ranges

These messages can occur whether or not a program is called with arguments in registers or by giving it a list.

1. ARGUMENT TOO LARGE (49)
   REAL PART TOO LARGE (52)
   IMAGINARY PART TOO LARGE (55)

The absolute value of the number is too large if the number is positive or negative. Check the numerical stability of the computation being performed. Check if variables have been initialized. Consider building error-checking logic into the program.

2. ARGUMENT OUT OF RANGE: POSITIVE (50)
   ARGUMENT OUT OF RANGE: NEGATIVE (51)
   REAL PART OUT OF RANGE: POSITIVE (53)
   REAL PART OUT OF RANGE: NEGATIVE (54)
   IMAGINARY PART OUT OF RANGE: POSITIVE (56)
   IMAGINARY PART OUT OF RANGE: NEGATIVE (57)

The number is greater than its maximum permissible value or less than its minimum permissible value. Remedies are as in #1.

3. OVERFLOW ON CONVERSION (58)

The absolute value of a floating-point number was too large to be converted to an integer. Besides the messages listed in 1 above, consider if this conversion is really appropriate.

4. ZERO ARGUMENT (46)
   NEGATIVE ARGUMENT (47)
   ARGUMENTS (0,0) (48)
   ZERO DIVISOR (62)
   ARGUMENTS (ZERO, NEGATIVE) (63)

These cases must be averted by building appropriate logic into the source code. Numerical instability should be closely checked; the difference of two numbers may be quite inaccurate.

5. NEGATIVE EXPONENT (64)
   NEGATIVE BASE (74)
   ZERO BASE, NEGATIVE EXPONENT (65)

These are conditions that can arise in computing powers. Check the validity of the computations which determined the base and exponent; consider building in logic to avoid these erroneous conditions.

6. EXPONENTIAL OVERFLOW (60)
   EXPONENTIAL UNDERFLOW (61)

These are conditions that can arise when the computation of a number raised to a power has led to arithmetic faults. To safeguard against these conditions, build logic into the program to handle extreme values of either base or exponent.

RTL PROGRAM ERROR MESSAGES

ERROR MESSAGES THAT MAY BE INVOKED FROM MATHEMATICAL RTL PROGRAMS.

NUMBER	DIAGNOSTIC MESSAGE
45	AN ARGUMENT OF INCORRECT TYPE
46	ZERO ARGUMENT
47	NEGATIVE ARGUMENT
48	ARGUMENTS (0,0)
49	ARGUMENT TOO LARGE
50	ARGUMENT OUT OF RANGE : POSITIVE
51	ARGUMENT OUT OF RANGE : NEGATIVE
52	REAL PART TOO LARGE
53	REAL PART OUT OF RANGE : POSITIVE
54	REAL PART OUT OF RANGE : NEGATIVE
55	IMAGINARY PART TOO LARGE
56	IMAGINARY PART OUT OF RANGE : POSITIVE
57	IMAGINARY PART OUT OR RANGE : NEGATIVE
58	OVERFLOW ON CONVERSION
59	VALUE OF SIZE IS ILLEGAL
60	EXPONENTIAL OVERFLOW
61	EXPONENTIAL UNDERFLOW
62	ZERO DIVISOR
63	ARGUMENTS (ZERO, NEGATIVE)
64	NEGATIVE EXPONENT
65	ZERO BASE, NEGATIVE EXPONENT

RTL PROGRAM ERROR MESSAGES (Continued)

NUMBER	DIAGNOSTIC MESSAGE
66	NEGATIVE INDEX
67	SIZE TOO LARGE
68	SIZE NOT POSITIVE
69	ILLEGAL FUNCTION CODE
70	ILLEGAL TRAP CODE
71	ILLEGAL LEVEL OF TRAPPING
72	ILLEGAL CALL
73	INCORRECT TYPE OF VALUE OF FUNCTION
74	NEGATIVE BASE
75	ARGUMENT OF INCORRECT CLASS
76	INCORRECT NUMBER OF ARGUMENTS

APPENDIX 2
ALPHABETICAL LIST OF SUBPROGRAMS DOCUMENTED IN THIS MANUAL

Math Function	Paragraph Reference
@ABS	3.5.3
@ACOS	3.2.5
@AIMAG	3.5.2
@AINT	3.5.6
@ALOG	3.2.4
@ALOG1	3.2.4
@AMAX0	3.3
@AMAX1	3.3
@AMIN0	3.3
@AMIN1	3.3
@ANINT	3.5.6
@ASIN	3.2.5
@ATAN	3.2.5
@ATAN2	3.2.4
@CABS	3.2.4
@CCOS	3.2.3
@CDABS	3.5.2
@CDCOS	3.2.3
@CDEXP	3.2.2
@CDLOG	3.2.4
@CDSIN	3.2.3
@CDSQR	3.2.1
@CEXP	3.2.2
@CLOG	3.2.4
@CMPLX	3.4
@CONJG	3.5.2
@COS	3.2.2
@COSH	3.2.3
@CSIN	3.2.3
@CSQRT	3.2.1
@DABS	3.5.3
@DACOS	3.2.5
@DASIN	3.2.5
@DATAN	3.2.5
@DATN2	3.2.5
@DBLE	3.4
@DCMPL	3.4
@DCONJG	3.5.2
@DCOS	3.2.3
@DCOSH	3.2.6
@DDIM	3.5.5
@DEXP	3.2.6
@DFLOAT	3.4
@DIM	3.5.5
@DIMAG	3.5.2
@DINT	3.5.6
@DLOG	3.2.4
@DLOG1	3.2.4
@DMAX1	3.3
@DMIN1	3.3
@DMOD	3.5.1
@DNINT	3.5.6
@DPROD	3.5.7
@DREAL	3.4
@DSIGN	3.5.4
@DSIN	3.2.3
@DSINH	3.2.6
@DSQRT	3.2.1, 3.2.5
@DTAN	3.2.3
@DTANH	3.2.6
@EXP	3.2.2., 3.2.6
@FLOAT	3.4
@IABS	3.5.3
@IDIM	3.5.5
@IDINT	3.4
@IDNIN	3.5.6
@IFIX	3.4
@INT	3.4
@INT2	3.4
@ISIGN	3.5.4
@MAX0	3.3
@MAX1	3.3
@MIN0	3.3
@MIN1	3.3
@MOD	3.5.1
@NINT	3.5.6
@REAL	3.4
@SIGN	3.5.4
@SIN	3.2.2, 3.2.3
@SINH	3.2.3, 3.2.6
@SNGL	3.4
@SQRT	3.2.1, 3.2.5, 3.5.2
@TAN	3.2.3
@TANH	3.2.6

Math Function	Paragraph Reference
@@CCX	3.4
@@CDBL	3.4
@@CDCD	3.4
@@CDCX	3.4
@@CDI	3.4
@@CDI2	3.4
@@CDRE	3.4
@@CI2	3.4
@@CINT	3.4
@@CREA	3.4
@@CXCD	3.4
@@ICD	3.4
@@RCD	3.4
.ACOS	3.2.5
.AINT	3.5.6
.ALOG	3.2.4
.ALOG1	3.2.4
.ANINT	3.5.6
.ASIN	3.2.5
.ATAN	3.2.5
.ATAN2	3.2.4
.CABS	3.2.4
.CCOS	3.2.3
.CDABS	3.2.4
.CDCOS	3.2.3
.CDDIV	3.7
.CDEXP	3.2.2
.CDLOG	3.2.4
.CDMPY	3.7
.CDSIN	3.2.3
.CDSQRT	3.2.1
.CDXXC	3.6
.CDXXCD	3.6
.CDXXD	3.6
.CDXXI	3.6
.CDXXR	3.6
.CEXP	3.2.2
.CLOG	3.2.4
.COS	3.2.2, 3.2.3
.COSH	3.2.3, 3.2.6
.CSIN	3.2.3
.CSQRT	3.2.1
.CXDIV	3.7
.CXMPY	3.7
.CXXC	3.6
.CXXCD	3.6
.CXXD	3.6
.CXXR	3.6
.CXXI	3.7

Math Function	Paragraph Reference
.DACOS	3.2.5
.DASIN	3.2.5
.DATAN	3.2.5
.DATN2	3.2.5
.DCOS	3.2.3
.DCOSH	3.2.6
.DEXP	3.2.6
.DINT	3.5.6
.DLOG	3.2.4
.DLOG1	3.2.4
.DNINT	3.5.6
.DSIN	3.2.3
.DSINH	3.2.6
.DSQRT	3.2.1, 3.2.5
.DTAN	3.2.3
.DTANH	3.2.6
.DXXC	3.6
.DXXCD	3.6
.DXXD	3.6
.DXXI	3.6
.DXXR	3.6
.EXP	3.2.2, 3.2.6
.HYDEX	3.2.6
.HYEXP	3.2.6
.IXXC	3.6
.IXXCD	3.6
.IXXI	3.6
.IXXD	3.6
.IXXR	3.6
.RXXC	3.6
.RXXCD	3.6
.RXXI	3.6
.RXXD	3.6
.SIN	3.2.2, 3.2.3
.SINH	3.2.3, 3.2.6
.SQRT	3.2.1, 3.2.5, 3.5.2
.TAN	3.2.3
.TANH	3.2.6

Math Function	Paragraph Reference
ABS	3.5.3
ACOS	3.2.5
AIMAG	3.5.2
AINT	3.5.6
ALOG	3.2.4
ALOG10	3.2.4
AMAX0	3.3
AMAX1	3.3
AMIN0	3.3
AMIN1	3.3
AMOD	3.5.1
ANINT	3.5.6
ASIN	3.2.5
ATAN	3.2.5
ATAN2	3.2.5
CABS	3.5.1
CCOS	3.2.3
CDABS	3.5.2
CDCOS	3.2.3
CDEXP	3.2.2
CDLOG	3.2.4
CDSIN	3.2.3
CDSQRT	3.2.1
CEXP	3.2.2
CLOG	3.2.4
CMPLX	3.4
CONJG	3.5.2
COS	3.2.3
COSH	3.2.6
CSIN	3.2.3
CSQRT	3.2.1
DABS	3.5.3
DACOS	3.2.5
DASIN	3.2.5
DATAN	3.2.5
DATAN2	3.2.5
DBLE	3.4
DCMPLX	3.4
DCONJG	3.5.2
DCOS	3.2.3
DCOSH	3.2.6
DDIM	3.5.5
DEXP	3.2.2
DFLOAT	3.4
DIM	3.5.5
DIMAG	3.5.2
DINT	3.5.6
DLOG	3.2.4
DLOG10	3.2.4
DMAX1	3.3
DMIN1	3.3
DMOD	3.5.1
DNINT	3.5.6
DPROD	3.5.7
DREAL	3.4
DSIGN	3.5.4
DSIN	3.2.3
DSINH	3.2.6
DSQRT	3.2.1
DTAN	3.2.3
DTANH	3.2.6
EXP	3.2.2
FLOAT	3.4
IABS	3.5.3
IDIM	3.5.5
IDINT	3.4
IDNINT	3.5.6
IFIX	3.4
INT	3.4
INT2	3.4
ISIGN	3.5.4
MAX0	3.3
MAX1	3.3
MIN0	3.3
MIN1	3.3
MOD	3.5.1
NINT	3.5.6
REAL	3.4
SIGN	3.5.4
SIN	3.2.3
SINH	3.2.6
SNGL	3.4
SQRT	3.2.1
TAN	3.2.3
TANH	3.2.6

APPENDIX 3
ARITHMETIC TABLES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

HEXADECIMAL ADDITION TABLE

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION TABLE

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

APPENDIX 3 (Continued)

TABLE OF POWERS OF SIXTEEN

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADECIMAL TO DECIMAL CONVERSION TABLE

BYTE				BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

TABLE OF MATHEMATICAL CONSTANTS

Constant	Decimal Value	Hexadecimal Value
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\mathrm{Ln}\,\pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10}e$	0.43429 44819 03252	0.6F2D EC55
$\log_{2}e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\mathrm{Ln}\,\gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
Ln2	0.69314 71805 59945	0.B172 17F8
$\log_{10}2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
Ln10	2.30258 50929 94046	2.4D76 3777

ASCII CARD CODE CONVERSION TABLE

GRAPHIC	7-BIT ASCII CODE	CARD CODE	GRAPHIC	7-BIT ASCII CODE	CARD CODE
SPACE	20	BLANK	@	40	8-4
!	21	11-8-2	A	41	12-1
"	22	8-7	B	42	12-2
#	23	8-3	C	43	12-3
$	24	11-8-3	D	44	12-4
%	25	0-8-4	E	45	12-5
&	26	12	F	46	12-6
'	27	8-5	G	47	12-7
(	28	12-8-5	H	48	12-8
)	29	11-8-5	I	49	12-9
*	2A	11-8-4	J	4A	11-1
+	2B	12-8-6	K	4B	11-2
,	2C	0-8-3	L	4C	11-3
-	2D	11	M	4D	11-4
.	2E	12-8-3	N	4E	11-5
/	2F	0-1	O	4F	11-6
0	30	0	P	50	11-7
1	31	1	Q	51	11-8
2	32	2	R	52	11-9
3	33	3	S	53	0-2
4	34	4	T	54	0-3
5	35	5	U	55	0-4
6	36	6	V	56	0-5
7	37	7	W	57	0-6
8	38	8	X	58	0-7
9	39	9	Y	59	0-8
:	3A	8-2	Z	5A	0-9
;	3B	11-8-6	[	5B	12-8-2
<	3C	12-8-4	\	5C	0-8-2
=	3D	8-6	]	5D	12-8-7
>	3E	0-8-6	↑	5E	11-8-7
?	3F	0-8-7	←	5F	0-8-5

INDEX

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

No postage necessary if mailed in U.S.A.

STAPLE

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD

FOLD

STAPLE

STAPLE