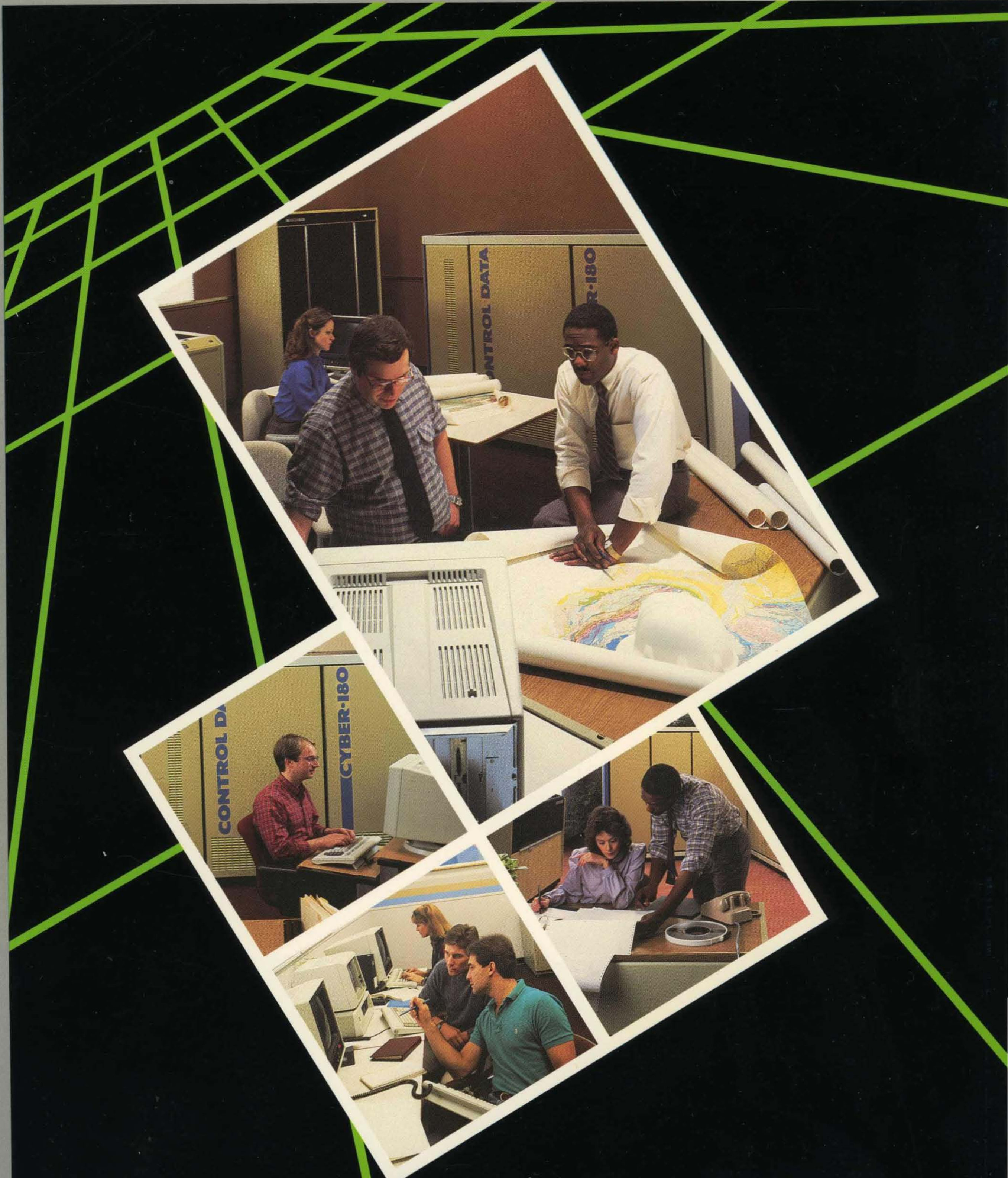
Math Library for NOS/VE

CONTROL DATA



Usage

60486513

Math Library for NOS/VE

Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60486513

Manual History

Revision	System Version	PSR Level	Product Version	Date
A	1.0.2	598	1.0	October 1983
B	1.1.2	630	1.0	April 1985
C	1.1.4	649	1.0	January 1986
D	1.2.1	664	1.0	October 1986
E	1.2.2	678	1.0	April 1987
F	1.2.3	688	1.0	September 1987
G	1.3.1	700	1.0	April 1988
H	1.4.1	716	1.0	January 1989

Revision H documents the Math Library for NOS/VE at release level 1.4.1, PSR level 716. This revision obsoletes all previous editions.

This revision documents the following new features:

- C language support.
- Enhanced CYBIL support.
- Performance improvements to the following math functions:

  ACOS
  ALOG
  ALOG10
  ASIN
  ATAN
  COS
  EXP
  SIN
  SQRT
  TAN

- Algorithm changes to functions DTOI and XTOI.

This revision also includes the following organizational changes:

- Number types, calling routines, error handling, scalar classification tables, calls from supported languages, and vector processing modules have been restructured as separate chapters.
- Short examples of each function have been added to illustrate the required number of arguments and number types.
- A bibliography has been added.

Changes are not marked with vertical bars in this revision because this manual has been reorganized.

Contents

Figures

Tables

About This Manual

This manual describes the math functions available in the CONTROL DATA® Common Modules Mathematical Library (CMML), referred to in this manual as the Math Library.

These math functions can be accessed by programs written in Ada, APL, Assembler, BASIC, C, CYBIL, FORTRAN Version 1, FORTRAN Version 2, LISP, Pascal, and Prolog. The Math Library is available under Control Data's Network Operating System/Virtual Environment (NOS/VE) operating system and can also be accessed from Control Data's UNIX[1] Virtual Environment (VX/VE) operating system. See the C for NOS/VE Usage manual for information about the C/VE Math Library.

Audience

To use the information in this manual, you should be familiar with the programming language from which you plan to call Math Library functions and with the NOS/VE® or VX/VE® operating system. In addition, you should have a basic knowledge of exponentiation, logarithms, trigonometry, and other functional areas depending upon how you plan to use the Math Library.

1. UNIX is a registered trademark of AT&T.

Manual Organization

This manual is organized into the following chapters:

- Chapter 1 - Introduction

  Introduces the Math Library and its mathematical and exponential functions. Defines the Math Library and its uses. Discusses strengths and limitations. Categorizes the functions. Explains entry points.

- Chapter 2 - Number Types

  Describes the number types used by the Math Library: integer, single precision floating-point, double precision floating-point, and complex.

- Chapter 3 - Calling Routines

  Describes the call-by-reference and call-by-value calling routines.

- Chapter 4 - Calls From Languages

  Provides examples of how these functions can be accessed by Ada, Assembler, C, and CYBIL programs. Also discusses other languages such as FORTRAN Version 1, FORTRAN Version 2, Pascal, APL, BASIC, COBOL, LISP, and Prolog.

- Chapter 5 - Error Handling

  Describes error handling for scalar processing including errors caused by bad input and inaccuracy caused by computer approximations. Contrasts call-by-reference and call-by-value error handling.

- Chapter 6 - Scalar Classification Tables

  Provides classification tables for easy identification of types of arguments, type of results, input domains, output ranges, and other detailed information.

- Chapter 7 - Vector Processing

  Describes vector processing and how it is used including hardware selection and error handling. Provides tables that summarize specific vector processing features.

- Chapter 8 - Function Descriptions

  Presents the functions in alphabetical order with specific information about the purpose of each function, the handling of the calling routines, and applicable algorithmic or error handling information. Short examples are provided to illustrate the required number of arguments and number types.

- Chapter 9 - Auxiliary Routines

  Presents detailed information on auxiliary routines that are called only by other math functions (for example, most of the computation for DTANH is performed in function DEULER).

Additional information is available in the following appendixes:

Appendix A - Glossary

Defines commonly-used terms and phrases.

Appendix B - Related Manuals

Lists manuals related to the Math Library including NOS/VE manuals and applicable language manuals.

Appendix C - ASCII Character Set

Provides the standard ASCII character set. Additional character sets are available in the applicable language manuals.

Appendix D - Bibliography

Lists mathematical reference works that were used as sources for algorithms or provide related background information.

Typographical Conventions

This manual uses the following typographical conventions:

...	In formulas, a horizontal ellipsis indicates that the preceding item can be repeated as necessary.
*	In formulas, an asterisk indicates multiplication.
**	In formulas, two successive asterisks indicate exponentiation.
| |	In formulas, vertical bars indicate the absolute value of the quantity.
( )	In intervals, parentheses indicate an open interval (the end points are not included).
[ ]	In intervals, brackets indicate a closed interval (the end points are included).
( ]	In intervals, closure by a left parenthesis and a right bracket includes the right end point, but not the left end point.
[ )	In intervals, closure by a left bracket and a right parenthesis includes the left end point, but not the right end point.
italic type	Used for special emphasis (for example, to highlight C data types and C statement names).

Mathematical Conventions

This manual uses the following mathematical conventions:

- All numbers used in this manual are decimal unless otherwise indicated. Other number systems are indicated by a notation after the number (for example, FA34 hexadecimal).
- All references to logarithm (log) are base e unless otherwise indicated.
- All references to infinite values include positive and negative infinity unless otherwise indicated.

For rules about standard and nonstandard floating-point numbers, see Floating-Point Computation Rules in chapter 2, Number Types.

Submitting Comments

The last page of this manual is a comment sheet. Please use it to give us your opinion of the manual's usability, to suggest specific improvements, and to report technical or typographical errors. If the comment sheet has already been used, you can mail your comments to:

Control Data Corporation
Technical Publications
P. O. Box 3492
Sunnyvale, California 94088-3492

Please indicate whether you would like a written response.

Also, if you have access to SOLVER, an online facility for reporting problems, you can use it to submit comments about the manual. For example, use FN8 as the product identifier for problems that are related to FORTRAN Version 1 and FV8 as the product identifier for problems related to FORTRAN Version 2.

In Case You Need Assistance

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help beyond that provided in the documentation or find that the Math Library for NOS/VE does not perform as described, call us at one of the following numbers and a support analyst will work with you.

From the USA and Canada: (800) 345-9903

From other countries: (612) 851-4131

The preceding numbers are for help on product usage. Address questions about the physical packaging and/or distribution of printed manuals to Literature and Distribution Services at the following address:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

or you can call (612) 292-2101. If you are a Control Data employee, call CONTROLNET® 243-2100 or (612) 292-2100.

Introduction 1

Introduction 1

This manual describes the mathematical functions available in Control Data's Common Modules Mathematical Library (CMML). CMML, referred to as the Math Library in this manual, contains a wide assortment of mathematical functions.

These functions can be accessed by programs written in Ada, APL, Assembler, BASIC, C, CYBIL, FORTRAN Version 1, FORTRAN Version 2, LISP, Pascal, and Prolog. See chapter 4, Calls From Languages, for detailed information and examples of how to call Math Library functions from various languages.

Figure 1-1 shows the relationship of the Math Library to the NOS/VE operating system. The NOS/VE Math Library environment, in addition to the compilers listed above, includes the System Command Language (SCL), the NOS/VE block-structured interpreter, LIB99, a library of subroutines and functions that can be called from FORTRAN (or Ada through FORTRAN), and the Debug utility.

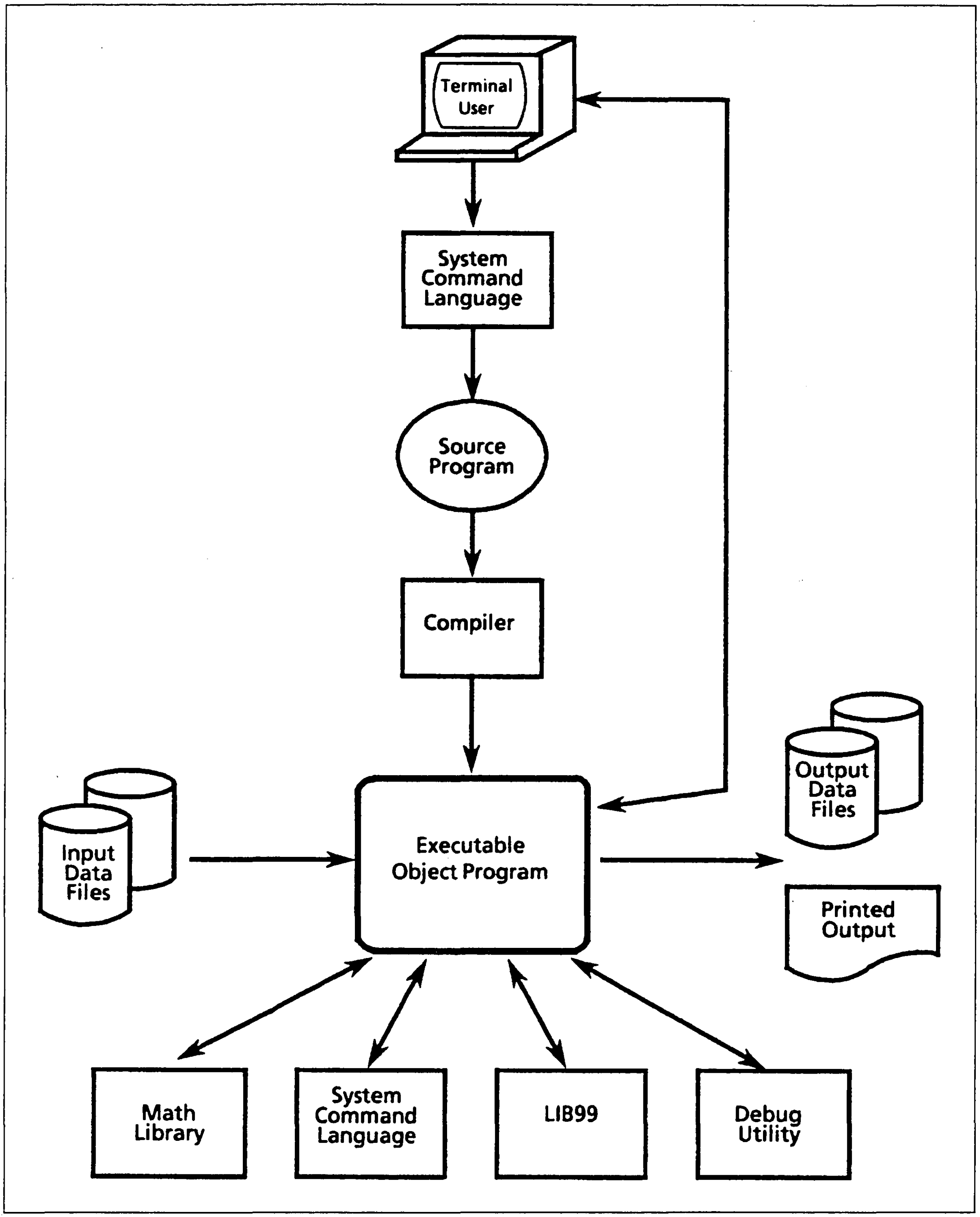


Figure 1-1. NOS/VE Math Library Environment

Functions Available

The Math Library provides approximately 100 math functions. In order to provide an overview, the NOS/VE Math Library functions are categorized in this manual as follows:

- Exponential
- Logarithmic
- Trigonometric
- Hyperbolic
- Conversion and maximum/minimum
- Bit manipulation
- Random number
- Error

The above categories are not precisely defined (for example, exponentiation is used in many of the trigonometric algorithms), but these categories are provided so you can more easily understand the contents of the Math Library.

See chapter 6, Scalar Classification Tables, for tables that categorize the available functions according to the above list. See chapter 8, Function Descriptions, for an alphabetical presentation of each function including a description, a discussion of the calling routines, algorithmic and error analysis information, and the effects of argument error, if applicable.

Additional FORTRAN for NOS/VE LIB99 functions can be called from FORTRAN Version 1 or FORTRAN Version 2. These functions in turn can be accessed by any language that can interface with FORTRAN (for example, Ada and CYBIL can make calls to FORTRAN). LIB99 can perform the following tasks:

- Perform basic vector arithmetic
- Perform basic matrix algebra
- Solve linear systems of equations
- Compute Fast Fourier Transforms
- Sort lists
- Compute eigenvalues and eigenvectors

Exponential Functions

The exponential functions are as follows:

Function	Description
CEXP	Complex exponential (base e)
CSQRT	Complex square root
DEXP	Double precision exponential (base e)
DSQRT	Double precision square root
DTOD	Exponentiation with double precision base and double precision exponent
DTOI	Exponentiation with double precision base and integer exponent
DTOX	Exponentiation with double precision base and real exponent
DTOZ	Exponentiation with double precision base and complex exponent
EXP	Exponential (base e)
ITOD	Exponentiation with integer base and double precision exponent
ITOI	Exponentiation with integer base and integer exponent
ITOX	Exponentiation with integer base and real exponent
ITOZ	Exponentiation with integer base and complex exponent
SQRT	Square root
XTOD	Exponentiation with real base and double precision exponent
XTOI	Exponentiation with real base and integer exponent
XTOX	Exponentiation with real base and real exponent
XTOZ	Exponentiation with real base and complex exponent
ZTOD	Exponentiation with complex base and double precision exponent
ZTOI	Exponentiation with complex base and integer exponent
ZTOX	Exponentiation with complex base and real exponent
ZTOZ	Exponentiation with complex base and complex exponent

Chapters 6 and 8 provide detailed information about each of these functions. Chapter 6 provides two tables with exponentiation information including the number types of the results of the exponentiation functions.

Logarithmic Functions

The logarithmic functions are as follows:

Function	Description
ALOG	Natural logarithm (base e)
ALOG10	Common logarithm (base 10)
CLOG	Complex natural logarithm (base e)
DLOG	Double precision natural logarithm (base e)
DLOG10	Double precision common logarithm (base 10)

Chapters 6 and 8 provide detailed information about each of these functions.

Trigonometric Functions

The trigonometric functions return values in radians except for COSD, SIND, and TAND which return values in degrees. The trigonometric functions are as follows:

Function	Description
ACOS	Inverse cosine
ASIN	Inverse sine
ATAN	Inverse tangent
ATAN2	Inverse tangent of the ratio of two arguments
CCOS	Complex cosine
COS	Cosine
COSD	Cosine in degrees
COTAN	Cotangent
CSIN	Complex sine
DACOS	Double precision inverse cosine
DASIN	Double precision inverse sine
DATAN	Double precision inverse tangent
DATAN2	Double precision inverse tangent of the ratio of 2 arguments
DCOS	Double precision cosine
DSIN	Double precision sine
DTAN	Double precision tangent
SIN	Sine
SIND	Sine in degrees
TAN	Tangent
TAND	Tangent in degrees

Chapters 6 and 8 provide detailed information about each of these functions.

Hyperbolic Functions

The hyperbolic functions are as follows:

Function	Description
ATANH	Inverse hyperbolic tangent
COSH	Hyperbolic cosine
DCOSH	Double precision hyperbolic cosine
DSINH	Double precision hyperbolic sine
DTANH	Double precision hyperbolic tangent
SINH	Hyperbolic sine
TANH	Hyperbolic tangent

Chapters 6 and 8 provide detailed information about each of these functions.

Conversion and Maximum/Minimum Functions

The conversion and maximum/minimum functions are as follows:

Function	Description
ABS	Absolute value
AIMAG	Imaginary part of a complex argument
AINT	Truncation
AMOD	Returns the remainder of a ratio (uses real numbers)
ANINT	Nearest whole number
CABS	Complex absolute value
CONJG	Conjugate
DABS	Double precision absolute value
DDIM	Double precision positive difference
DIM	Positive difference
DINT	Double precision truncation
DMOD	Returns the remainder of a ratio (uses double precision numbers)
DNINT	Double precision nearest whole number
DPROD	Double precision product
DSIGN	Double precision transfer of sign
IABS	Integer absolute value
IDIM	Integer positive difference
IDNINT	Double precision nearest integer
ISIGN	Integer transfer of sign
MOD	Returns the remainder of a ratio (uses integers)
NINT	Nearest integer
SIGN	Transfer of sign

Chapters 6 and 8 provide detailed information about each of these functions.

Bit Manipulation Functions

The bit manipulation functions are as follows:

Function	Description
EXTB	Extract bits
INSB	Insert bits
SUM1S	Sum of 1 bits in one word

NOTE

The number of bits in a CYBER 180 word is always 64.

Chapters 6 and 8 provide detailed information about each of these functions.

Random Number Functions

The random number functions are as follows:

Function	Description
RANF	Generates the next random number in a series
RANGET	Returns the current random number seed of a task
RANSET	Sets the seed of the random number generator

Chapters 6 and 8 provide detailed information about each of these functions.

Error Functions

The error functions are as follows:

Function	Description
ERF	Computes the error function
ERFC	Computes the complementary error function

Chapters 6 and 8 provide detailed information about each of these functions.

Entry Point

Depending upon the language you use, you may want to call a Math Library function by an entry point other than its function name. (Assembly language calls cannot be made to function names.) Math Library functions have two types of entry point: call-by-reference and call-by-value. For example, the function ABS can be called by the call-by-reference entry point MLP$RABS (or ABS) or by the call-by-value entry point MLP$VABS. (See chapter 4, Calls From Languages, for an example.)

NOTE

The function name (for example, ABS) is also a call-by-reference entry point.

Figure 1-2 shows the naming conventions for entry points.

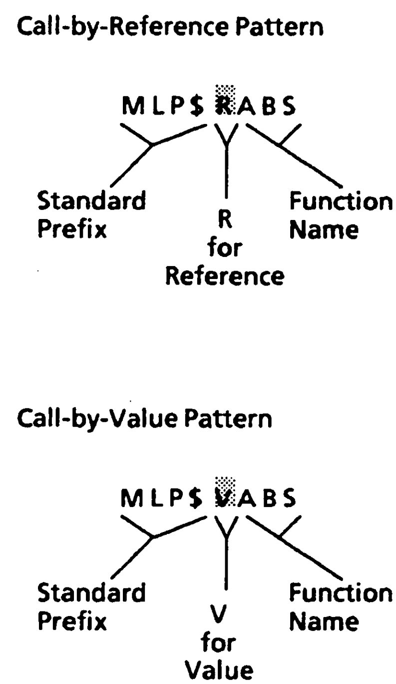


Figure 1-2. Pattern Diagram for Entry Points

Number Types 2

Number Types 2

This chapter discusses how the Math Library functions perform computations on the following number types:

- Integer
- Single precision floating-point (real)
- Double precision floating-point (long real)
- Complex

The following paragraphs describe how these number types are used by the Math Library.

Integer

An integer is a one-word, right-justified, two's complement 64-bit representation of all integers from −(2**63) through (2**63)−1. See figure 2-1 for an illustration of 8-byte integer format. All 8-byte integers up to the absolute value of 9,223,372,036,854,775,807 are accepted by the Math Library.

The implementation of type integer varies across languages. The C language, for example, has a 32-bit integer (*short int*) and a 64-bit long integer (*int*). (For an explanation of how to use the left-bit-shift (<<) operator to left justify *short int*, see C Calling the Math Library in chapter 4, Calls From Languages.)

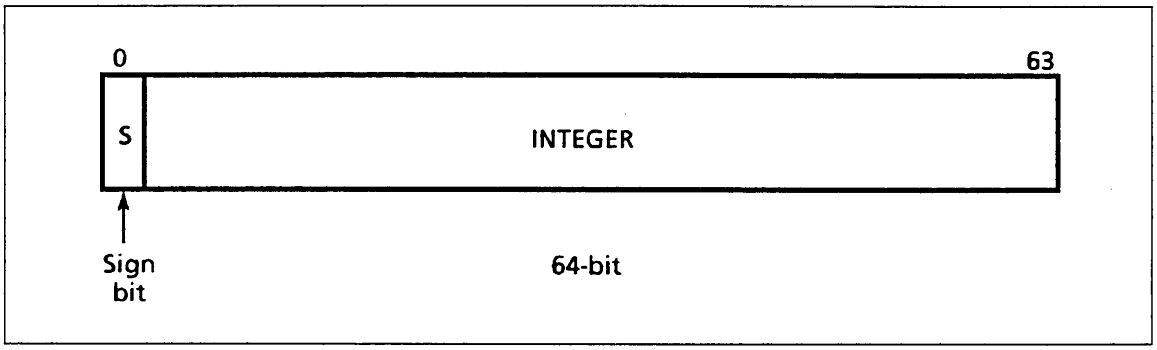


Figure 2-1. Bit Diagram of 8-Byte Integer Format

Single Precision Floating-Point Numbers

A single precision floating-point number consists of a sign bit, S, which is the sign of the fraction, a signed biased exponent (15 bits), and a fraction (48 bits) which is also called a coefficient or a mantissa. Figure 2-2 illustrates the internal representation of this format.

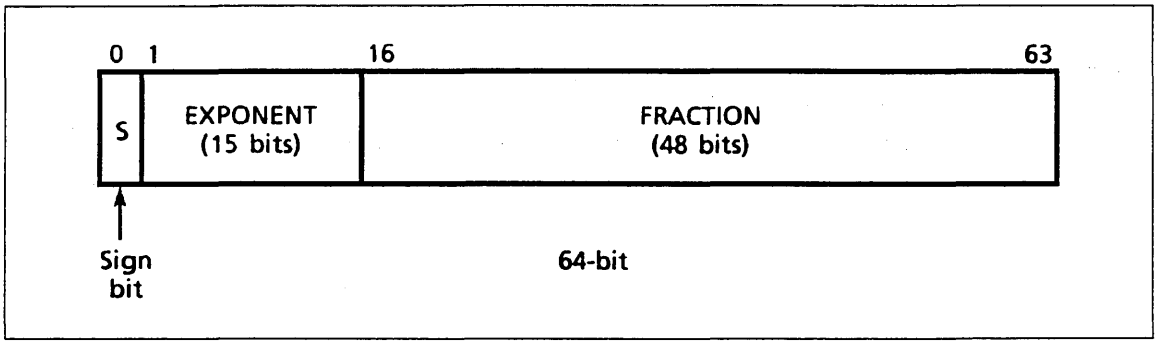


Figure 2-2. Bit Diagram of Single Precision Floating-Point Format

Single precision floating-point numbers consist of two types: standard and nonstandard.

Single Precision Standard Numbers

Standard numbers are numbers that have exponents in the range of 3000 hexadecimal through 4FFF hexadecimal, inclusive, and have a nonzero fraction or 0. Standard numbers can be normalized or unnormalized. A normalized standard number has a 1 (one) in bit position 16 (the most significant bit of the fraction), where bit position zero is the leftmost bit.

The range in magnitude, M, covered by standard, normalized single precision numbers is as follows:

```
-1* (1 -2** -48) * 2** 4095  <= M  <= -2** -4097
 0
 2** - 4097 <= M  <= (1 -2** -48)*2**4095
```

The above range provides approximately 14.4 decimal digits of precision.

Single Precision Nonstandard Numbers

Nonstandard floating-point numbers have the following representations:

- A nonzero unnormalized floating-point number with a zero fraction and a standard exponent
- A floating-point number with an exponent in the range 5000 through 6FFF hexadecimal (+infinite) and D000 through EFFF hexadecimal (-infinite)
- A floating-point number with an exponent in the range 7000 through 7FFF hexadecimal (+indefinite) and F000 through FFFF (-indefinite)
- A nonzero floating-point number with an exponent in the range 0000 through 0FFF hexadecimal (+Z1) and 8000 through 8FFF (-Z1)
- A floating-point number with an exponent in the range 1000 through 2FFF hexadecimal (+Z2) and 9000 through AFFF (-Z2)

The last item includes a sign bit followed by 63 zero bits. Nonstandard numbers are not used in computations, but some are returned as default error values as described later in this chapter under Default Error Values.

Double Precision Floating-Point Numbers

A double precision floating-point number consists of two words, each a single precision floating-point number. The coefficient of the second word is considered to be an extension of the fraction of the first word, yielding a 96-bit fraction. The exponent of the second word following an arithmetic operation is identical to that of the first word. The number type of the first word determines the type of the second word.

See figure 2-3 for an illustration of the internal representation of a double precision floating-point format.

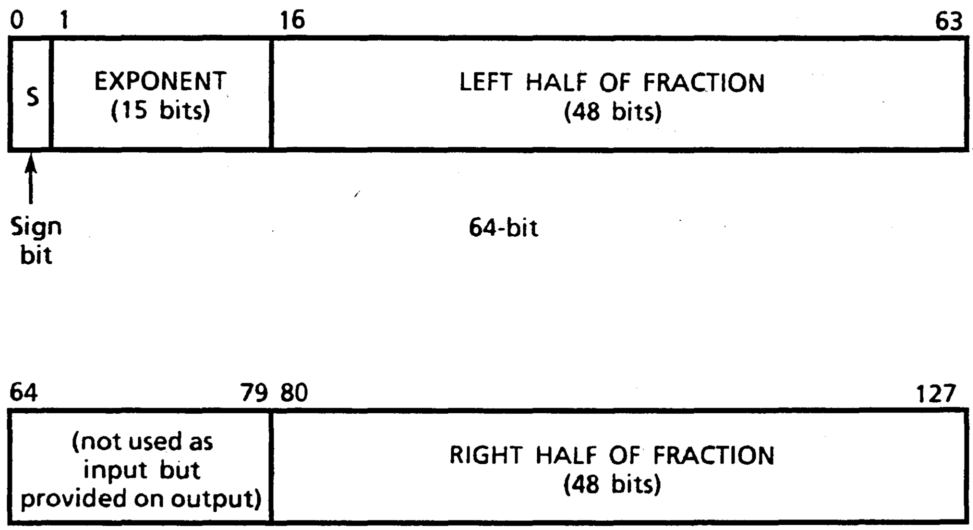


Figure 2-3. Bit Diagram of Double Precision Floating-Point Format

The range in magnitude, M, covered by standard, normalized double precision numbers is:

```
-1* (1 -2** -96) * 2** 4095 <= M <= -2** -4097
0
2** - 4097 <= M <= (1 -2** -96)*2**4095
```

The above range yields approximately 28.9 decimal digits of precision. See figure 2-4 for a summary of NOS/VE floating-point representation.

Coefficient Sign	Hexadecimal Exponent Including Coefficient Sign	Actual Exponent (To The Base 2)	Input Arguments	Results
Coefficient Sign Equal To 0 (Positive Numbers)	7XXX	----	Indefinite	7000.0 → 0
	6FFF ↑ 5000	$2^{12,287}$ ↑ $2^{4,096}$	Infinite	Overflow Mask = 0 : 5000.0 → 0 Overflow Mask = 1 : As Shown
	4FFF ↑ 4000 3FFF ↓ 3000	$2^{4,095}$ ↑ 2^{0} 2^{-1} ↓ $2^{-4,096}$	Standard	As Shown
	2FFF ↓ 1000	$2^{-4,097}$ ↓ $2^{-12,288}$	Zero	Underflow Mask = 0 : 0000.0 → 0 Underflow Mask = 1 : As Shown
	0XXX	- - - - -	Zero	Not Applicable
Coefficient Sign Equal To 1 (Negative Numbers)	8XXX			
	9000 ↑ AFFF	$2^{-12,288}$ ↑ $2^{-4,097}$	Zero	Underflow Mask = 0 : 0000.0 → 0 Underflow Mask = 1 : As Shown
	B000 ↑ BFFF C000 ↓ CFFF	$2^{-4,096}$ ↑ 2^{-1} 2^{0} ↓ $2^{4,095}$	Standard	As Shown
	D000 ↓ EFFF	$2^{4,096}$ ↓ $2^{12,287}$	Infinite	Overflow Mask = 0 : D000.0 → 0 Overflow Mask = 1 : As Shown
	FXXX	---	Indefinite	7000.0 → 0

Figure 2-4. Summary of NOS/VE Floating-Point Representation

Complex Numbers

A complex number consists of two words, each a single precision floating-point number. The first word represents the real part of the complex number; the second word represents the imaginary part.

A complex number is considered to be indefinite if either the real or imaginary part is indefinite. Similarly, a complex number is considered to be infinite if either the real or imaginary part is infinite.

Floating-Point Computation Rules

Throughout this manual, unless otherwise documented, the following rules apply to floating-point computation:

1. If a standard form of a number type is used in a computation, a standard form of the same type results, unless the answer computed exceeds the range of values for standard numbers or if a mathematically invalid operation is attempted.

2. If a nonstandard number other than zero is used in a computation, or if the limits to a standard form of a number type are exceeded, error handling occurs, unless various nonstandard numbers are within the domain of the function.

Default Error Values

The Math Library uses the following default error values:

- Positive indefinite (+IND)
- Negative indefinite (–IND)
- Zero (0)
- Positive infinity (+INF)
- Negative infinity (–INF)

Most of the Math Library functions have a default error value of positive indefinite. The following functions have a default error value of zero: CCOS, DEXP, ERFC, EXP, IDIM, IDNINT, ITOI, MOD, and NINT.

Calling Routines 3

Calling Routines 3

Routines and Calls

The Math Library functions are predefined functions that can be called from Ada, APL, Assembler, BASIC, CYBIL, FORTRAN Version 1, FORTRAN Version 2, LISP, Pascal, or Prolog programs according to the attributes of the calling language. Some functions are not available to every language; see chapter 4, Calls From Languages, for information about specific languages.

The Math Library provides two types of calling routines:

- Call-by-reference
- Call-by-value

These calling routines are discussed in the following paragraphs.

NOTE

This chapter deals with scalar processing only. For a discussion of vectors, see chapter 7, Vector Processing.

Call-by-Reference

A call to a call-by-reference routine consists of the following process:

1. The user program sets up a parameter list (argument list) in memory.
2. The call to the instruction causes the first-word address to be stored in register A4 as the routine is invoked.
3. The call-by-reference routine is called through one of two entry points (for example, ABS or MLP$RABS). Argument error processing is set up in this routine.
4. If the argument list is valid, the routine calls or branches to the call-by-value routine, depending on the function.
5. The call-by-value routine performs the appropriate computation and returns a result.

If the argument list is invalid, the call-by-value routine is not executed and an error message is returned. See the appropriate language manual (printed or online) for information about a compilation message. See the NOS/VE Diagnostic Messages manual (printed or online) for information about a runtime message.

Figure 3-1 is a Nassi-Shneiderman chart[1] illustrating the logical flow of the call-by-reference routine.

NOTE

Call-by-reference is synonymous with call-by-address.

---

1. Nassi-Shneiderman charts (also called Chapin charts) are read like flow charts: a rectangle indicates a process, an inverted isosceles triangle indicates a decision, and a right triangle indicates a branch from a decision.

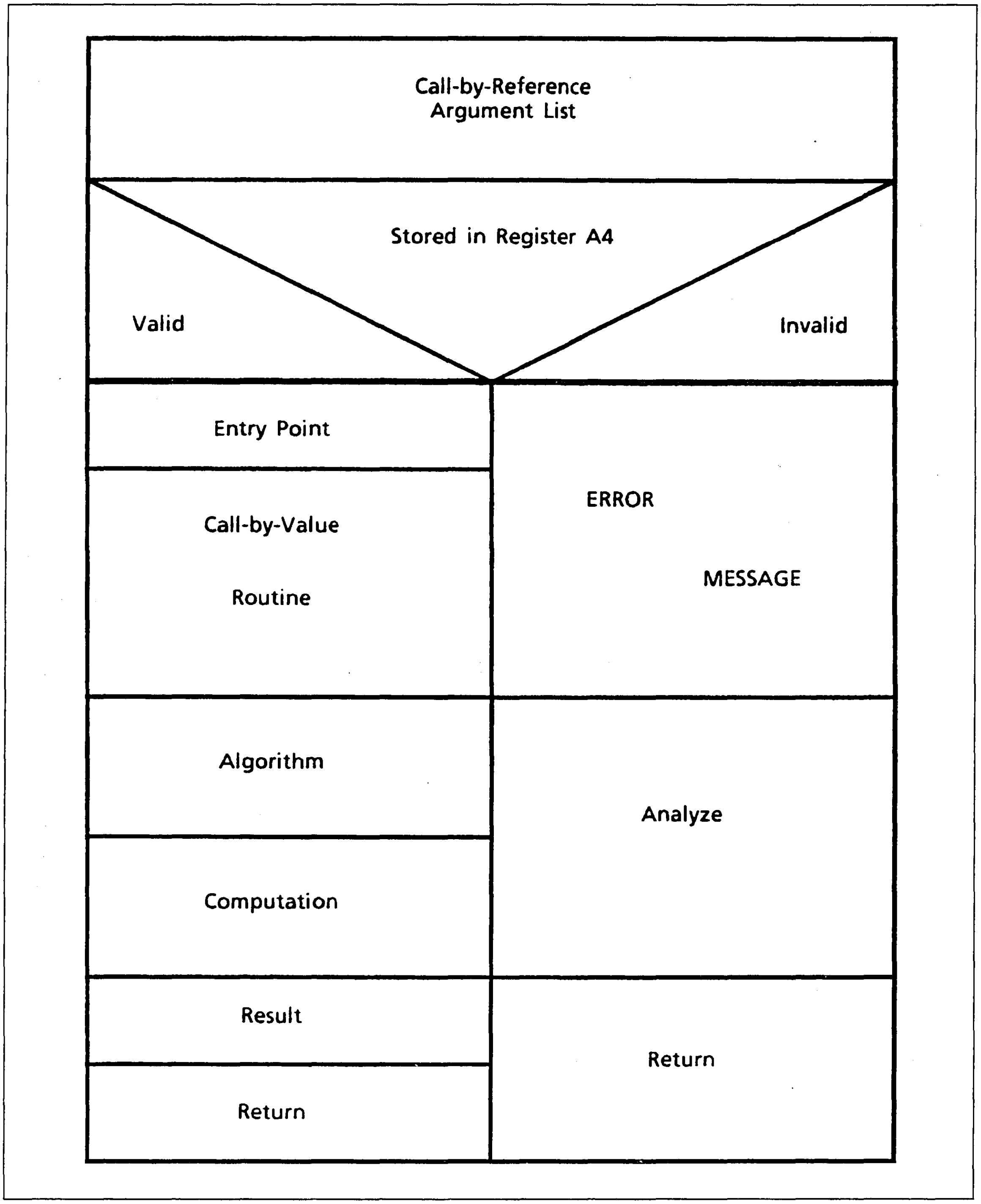


Figure 3-1. Call-by-Reference Logic Diagram (Scalar)

Call-by-Value

A call to a call-by-value routine consists of the following process:

1. The user program sets up a parameter list (argument list) directly into the X registers before the routine is invoked.

2. The call to the instruction causes the first word of the first argument to be entered into register X2; the remaining words of each argument are entered into the registers successively.

   For example, the calling procedure for the exponentiation function ITOD (exponentiation with integer base and double precision exponent) uses registers X2, X3, and X4. Register X2 holds the integer base, and registers X3 and X4 hold the double precision exponent.

   The first and second words of a complex argument contain the real and imaginary parts, respectively. The first and second words of a double precision argument contain the high-order and low-order bits, respectively.

3. The call-by-value routine performs the appropriate computation, and when valid computations occur, returns a result. The result is returned in registers XE and XF.

   One-word results (type integer and single precision) are returned in register XF. Two-word results (type double precision and complex) are returned in registers XE and XF; the second word is stored in register XF.

If the call-by-value routine is called directly and the arguments are out-of-range, the job aborts during the computation and an error message is returned. See the NOS/VE Diagnostic Messages manual (printed or online) for information about a runtime message.

Figure 3-2 is a Nassi-Shneiderman chart[2] illustrating the logical flow of the call-by-value routine.

2. Nassi-Shneiderman charts (also called Chapin charts) are read like flow charts: a rectangle indicates a process, an inverted isosceles triangle indicates a decision, and a right triangle indicates a branch from a decision.

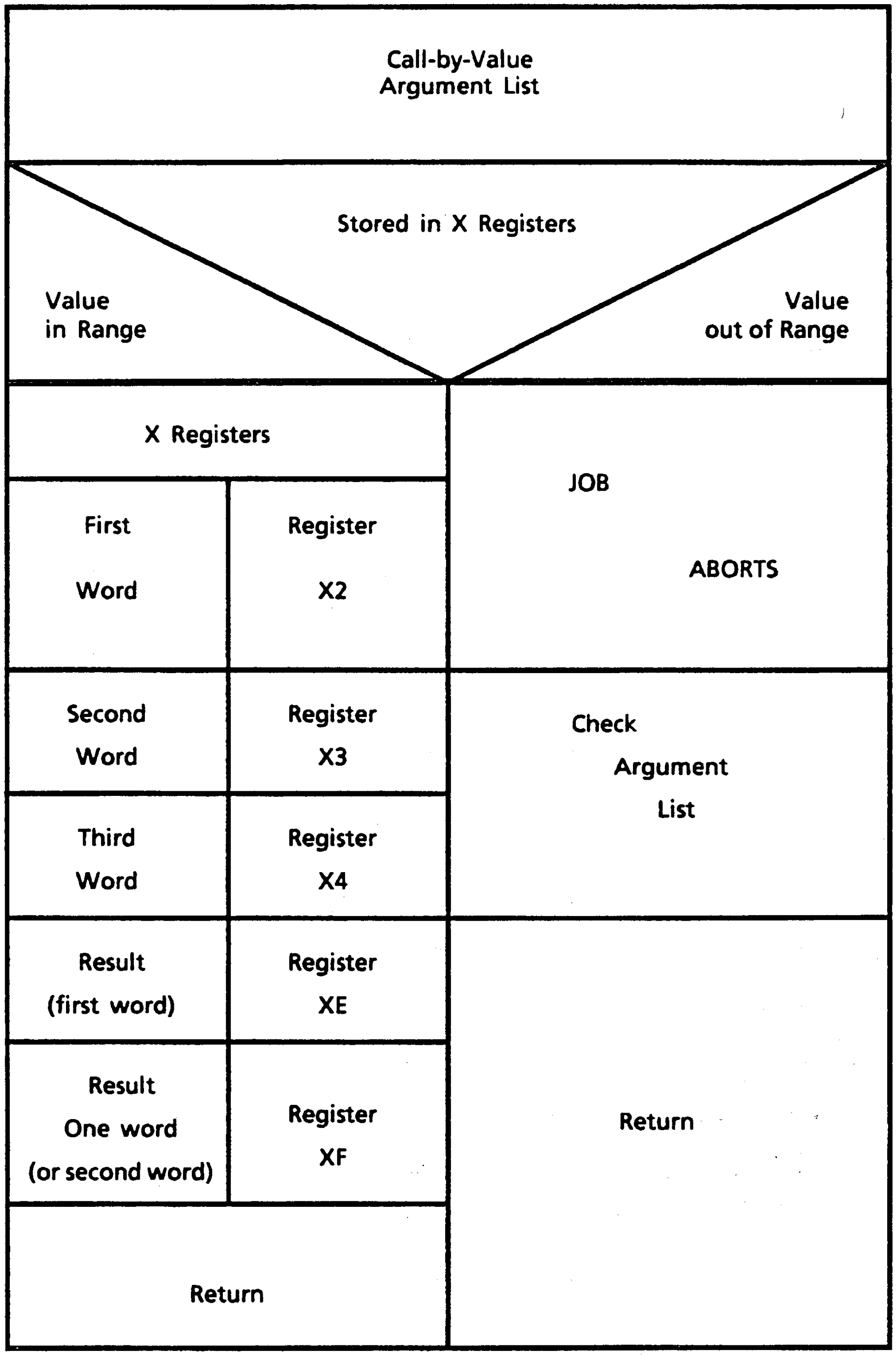


Figure 3-2. Call-by-Value Logic Diagram (Scalar)

Call-by-Reference Versus Call-by-Value Matrix

Some languages, such as FORTRAN and Pascal, can access the Math Library through call-by-reference or call-by-value routines. Other languages can use only one type of routine.

The language matrix provided as table 3-1 outlines the distinction between call-by-reference (addresses of arguments are passed) and call-by-value (values of arguments are passed) across the supported languages.

Transparency is defined as the apparent invisibility of the Math Library. APL, BASIC, LISP, and Prolog programmers do not need to know that the Math Library exists unless they get a range or type error, or need to perform error analysis.

FORTRAN and Pascal programmers have compile option EXPRESSION_EVALUATION=REFERENCE which selects call-by-reference over call-by-value, but the functioning of the Math Library is essentially transparent. Ada, Assembler, C, and CYBIL programmers have a few programming options which are discussed in the following sections.

Table 3-1. Language Matrix

Languages (Providing Interfaces)	Call-by-Reference	Call-by-Value
Ada	Yes	No
Assembler	Yes	Yes
C	Yes	Yes
CYBIL	Yes	Yes
Languages (With Transparent Access)	**Call-by-Reference**	**Call-by-Value**
APL	No	Yes
BASIC	No	Yes
FORTRAN Version 1	Yes	Yes
FORTRAN Version 2	Yes	Yes
LISP	Yes	No
Pascal	Yes	Yes
Prolog	Yes	No
Language (With No Access)	**Call-by-Reference**	**Call-by-Value**
COBOL	No	No

Inline Versus Out-of-Line Routines

Several of the NOS/VE compilers such as FORTRAN Version 1, FORTRAN Version 2, and Pascal have added some of the Math Library algorithms to their code generators; this inline process improves execution time significantly, but may slow down compilation slightly.

The following functions are available to the FORTRAN and Pascal compilers as inline routines:

ACOS	ATAN	SIN
ALOG	COS	SQRT
ALOG10	EXP	TAN
ASIN		

See chapter 4, Calls From Languages, for information about specific languages calling the Math Library.

Calls From Languages 4

Calls From Languages 4

This chapter provides examples and explanations of how to call the Math Library from the supported languages. The Math Library functions can be called from the following languages:

- Ada
- APL
- Assembler
- BASIC
- C
- CYBIL
- FORTRAN Version 1
- FORTRAN Version 2
- LISP
- Pascal
- Prolog

Many of these calls are transparent to the end user, but a working knowledge of the calling options could improve program design, performance, or accuracy.

APL, BASIC, FORTRAN, LISP, Pascal, and Prolog provide interfaces to the Math Library that are transparent to the user. COBOL has no direct access to the Math Library. Table 4-1 summarizes how each language calls the Math Library.

Table 4-1. Language Summary

Languages (Providing Interfaces)	Description
Ada	Calls the Math Library through pragma MATH_LIBRARY; also provides an interface to FORTRAN Version 1 or 2. Uses call-by-reference. See figure 4-1.
Assembler	Allows call-by-reference and call-by-value. See figure 4-2.
C	Allows call-by-reference and call-by-value. See figures 4-3 and 4-4.
CYBIL	Some functions can call with call-by-reference or call-by-value. Double precision functions can be called with call-by-reference only. See figure 4-5.

Languages (With Transparent Access)	Description
APL	No knowledge of the Math Library is necessary.
BASIC	No knowledge of the Math Library is necessary.
FORTRAN Version 1	Use EXPRESSION_EVALUATION=REFERENCE for call-by-reference on the FORTRAN command; otherwise, call-by-value is used. See table 4-4 for a summary of FORTRAN functions.
FORTRAN Version 2	Use EXPRESSION_EVALUATION=REFERENCE for call-by-reference on the VFORTRAN command; otherwise, call-by-value is used. Has array-processing intrinsic functions. See table 4-4 for a summary of FORTRAN functions.
LISP	No knowledge of the Math Library is necessary.
Pascal	Use the EXPRESSION_EVALUATION=REFERENCE parameter for call-by-reference on the PASCAL command; otherwise, call-by-value is used.
Prolog	No knowledge of the Math Library is necessary.

Language (With No Access)	Description
COBOL	The language cannot return a value from a function.

Ada Calling the Math Library

Ada supports calls to the Math Library functions. For each Math Library subroutine to be called from Ada, the Ada program must provide the following:

- Ada Subprogram Declaration:

  subprogram_specification ::=
     function identifier (formal_parameter_specifications)
     return type_mark

- Ada Interface Specification:

  subprogram_body ::= **pragma** INTERFACE (MATH_LIBRARY, identifier)

The Ada subprogram declaration provides:

- The name of the Math Library function in the function identifier field
- The types of the formal parameters (in formal_parameter_specifications)
- The subtype of the returned value (the result subtype) in the type_mark field

The name of the Math Library function must also appear as the identifier in the Ada interface specification. The name must be one of the Math Library function names. See table 4-1 for a summary of the input and output data types.

The next section provides an Ada example.

Ada Subprogram Declaration and Interface Specification

To use the Math Library function RANF (random number generator), enter the following Ada subprogram declaration and interface specification:

```
function RANF return FLOAT;
pragma INTERFACE (MATH_LIBRARY, RANF);
```

The subprogram declaration tells the following:

- RANF is the name of the Math Library function.
- RANF has no formal input parameters (parameters of mode **in**).
- The result is of type FLOAT (single precision real number).

NOTE

If an incorrect data type is passed to a Math Library function (for example, an INTEGER instead of a FLOAT), an incorrect value may be returned. Your program should check that the correct data type is passed.

Figure 4-1 illustrates how to implement the Ada MATH_LIBRARY pragma interface (interface specification). Procedure CALL_MATHLIB has a pragma interface to the Math Library function SQRT. SQRT has one formal input parameter (parameter of mode **in**), x of type FLOAT.

```
with TEXT_IO; use TEXT_IO;

   procedure CALL_MATHLIB is

      function SQRT(x : in FLOAT) return FLOAT;
      pragma INTERFACE (MATH_LIBRARY, SQRT);
      package FLT_IO is new FLOAT_IO (FLOAT);
      use FLT_IO;
      y : FLOAT;

   begin — CALL_MATHLIB

      PUT_LINE ("Start Ada");
      y := SQRT(225.0);
      PUT ("The square root of 225 is: ");
      PUT (y, fore => 2, aft => 2, exp => 0);
      NEW_LINE;
      PUT_LINE ("End Ada");

   end CALL_MATHLIB;
```

Figure 4-1. Ada Program Calling the Math Library

With the Ada MATH_LIBRARY pragma interface, you can access the Math Library functions. Table 4-2 summarizes the Ada MATH_LIBRARY functions and their input-output data types. The table lists the following for each function:

- Function name
- Precision type
- Description of the function
- Input type
- Output type

Table 4-2. Data Types for Ada MATH_LIBRARY Functions

Function	Precision	Description	Input Type	Output Type
ACOS	Single	Inverse circular cosine	FLOAT	FLOAT
AINT	Single	Integer part	FLOAT	FLOAT
ALOG	Single	Natural logarithm	FLOAT	FLOAT
ALOG10	Single	Common logarithm	FLOAT	FLOAT
ANINT	Single	Nearest integer	FLOAT	FLOAT
ASIN	Single	Inverse circular sine	FLOAT	FLOAT
ATAN	Single	Inverse circular tangent	FLOAT	FLOAT
ATANH	Single	Inverse hyperbolic tangent	FLOAT	FLOAT
ATAN2	Single	Inverse circular tangent of a ratio of two arguments	FLOAT	FLOAT
COS	Single	Circular cosine	FLOAT	FLOAT
COSD	Single	Circular cosine in degrees	FLOAT	FLOAT
COSH	Single	Hyperbolic cosine	FLOAT	FLOAT
COTAN	Single	Circular cotangent	FLOAT	FLOAT
DACOS	Double	Inverse circular cosine	LONG_FLOAT	LONG_FLOAT
DASIN	Double	Inverse circular sine	LONG_FLOAT	LONG_FLOAT
DATAN	Double	Inverse circular tangent	LONG_FLOAT	LONG_FLOAT
DATAN2	Double	Inverse circular tangent of a ratio of two arguments	LONG_FLOAT	LONG_FLOAT
DCOS	Double	Circular cosine	LONG_FLOAT	LONG_FLOAT
DCOSH	Double	Hyperbolic cosine	LONG_FLOAT	LONG_FLOAT
DDIM	Double	Positive difference	LONG_FLOAT	LONG_FLOAT
DEXP	Double	Exponentiation function	LONG_FLOAT	LONG_FLOAT
DIM	Single	Positive difference	FLOAT	FLOAT
DINT	Double	Integer part	LONG_FLOAT	LONG_FLOAT
DLOG	Double	Natural logarithm	LONG_FLOAT	LONG_FLOAT
DLOG10	Double	Common logarithm	LONG_FLOAT	LONG_FLOAT
DNINT	Double	Nearest whole number	LONG_FLOAT	LONG_FLOAT
DPROD	Double	Product	LONG_FLOAT	LONG_FLOAT
DSIGN	Double	Transfer of sign	LONG_FLOAT	LONG_FLOAT
DSIN	Double	Circular sign	LONG_FLOAT	LONG_FLOAT
DSINH	Double	Hyperbolic sine	LONG_FLOAT	LONG_FLOAT

(Continued)

Table 4-2. Data Types for Ada MATH_LIBRARY Functions *(Continued)*

Function	Precision	Description	Input Type	Output Type
DSQRT	Double	Square root	LONG_FLOAT	LONG_FLOAT
DTAN	Double	Circular tangent	LONG_FLOAT	LONG_FLOAT
DTANH	Double	Hyperbolic tangent	LONG_FLOAT	LONG_FLOAT
ERF	Single	Error function	FLOAT	FLOAT
ERFC	Single	Error function complement	FLOAT	FLOAT
EXP	Single	Exponentiation	FLOAT	FLOAT
EXTB	—	Extract bits	INTEGER	INTEGER
IDIM	—	Positive difference of two integers	INTEGER	INTEGER
IDNINT	Double	Nearest whole number	LONG_FLOAT	INTEGER
INSB	—	Insert bits	INTEGER	INTEGER
ISIGN	—	Integer transfer of sign	INTEGER	INTEGER
NINT	Single	Nearest whole number	FLOAT	INTEGER
RANF	Single	Random number generator	None	FLOAT
RANGET	Single	Returns random number seed	None	FLOAT
RANSET	Single	Sets random number seed	FLOAT	FLOAT
SIGN	Single	Transfer of sign	FLOAT	FLOAT
SIN	Single	Circular sine	FLOAT	FLOAT
SIND	Single	Circular sine in degrees	FLOAT	FLOAT
SINH	Single	Hyperbolic sine	FLOAT	FLOAT
SQRT	Single	Square root	FLOAT	FLOAT
TAN	Single	Circular tangent	FLOAT	FLOAT
TAND	Single	Circular tangent in degrees	FLOAT	FLOAT
TANH	Single	Hyperbolic tangent	FLOAT	FLOAT

Ada Uses Call-by-Reference

The call-by-reference interface is supported by NOS/VE Ada for the Math Library. The NOS/VE Ada compiler appends the call-by-reference prefix (MLP$R) to the abbreviated Math Library function name.

In a call-by-reference computation, a parameter list is formed in memory and the first-word-address of this list is stored in register A4 before the routine is invoked.

Additional Ada Functions

Ada provides several other language defined functions. For example, exponentiation is provided with the exponentiation operator (**). Other functions such as **mod** (modulus) and **rem** (remainder) are provided as reserved words. See the Ada for NOS/VE reference or usage manual for additional information.

Calling FORTRAN and the Math Library From Ada

Ada supports calls to FORTRAN Version 1 and Version 2 subprograms. For each FORTRAN subprogram, the following Ada interface must be provided:

Formal Grammar:

subprogram_specification ::=
  procedure identifier (formal_parameter_specifications)
  | **function** identifier (formal_parameter_specifications)
    return type_mark

subprogram_body ::=
  pragma INTERFACE (FORTRAN, identifier)

Through the **pragma** INTERFACE (FORTRAN, identifier), an Ada program can call a FORTRAN Version 1 or Version 2 intrinsic function or a Math Library function.

For example, figure 4-1 could be modified to call SQRT through FORTRAN:

pragma INTERFACE (MATH_LIBRARY, SQRT);

The Ada FORTRAN interface has the following characteristics:

- FORTRAN subroutines and functions expect parameters to be passed by reference.
- The Ada compiler passes scalar parameters by value but array and string parameters by reference.
- When calling a FORTRAN subprogram, the Ada compiler passes, for scalar parameters, pointers to a copy of the value; for other types of parameters, the compiler passes pointers to the actual values.
- The NOS/VE Ada compiler does not check the modes and types of the Ada actual parameters and FORTRAN formal parameters for agreement.

Since in Ada the length of an array or a string parameter is always known at the time of the subprogram call, the Ada compiler can, when passing an address, pass the length of a string with a string address and the array descriptor with an array address. This allows parameters of type array or string to be declared in FORTRAN as either fixed or assumed size.

Exponentiation Using Ada

Ada provides the exponentiation operator (**), which is predefined for INTEGER, FLOAT, and LONG_FLOAT. Unlike FORTRAN, however, the right operand (the exponent) must be an integer for Ada exponentiation. The left operand (the base) can be any integer type or floating-point type, but not a fixed-point type.

Calls to FORTRAN can be made if exponents of different types are required. For a summary of the exponentiation functions, see table 6-3 (in chapter 6).

Assembler Calling the Math Library

The program illustrated in figure 4-2 calls the Math Library with assembly language. This program, identified as MATHEXM, illustrates both the call-by-reference and the call-by-value calling routines.

```
mathexm         ident
.       This program shows both methods of calling the Math Library
.       functions, call-by-reference and call-by-value.
.
.       The binding section contains the links to external code and data.
.       Its entries are set by the loader and the Object Code Utilities.
.
        use     binding                 . select the binding section
        ref     mlp$rsin                . define links to the SIN function
sinr    address c,mlp$rsin              . set the call-by-reference version
        ref     mlp$vsin
sinv    address c,mlp$vsin              . set the call-by-value version
data    address p,wseg                  . the link to the working section

        use     working                 . select working storage
wseg    align   0,8                     . ensure start at word boundary
pi      float   3.141592654
one     float   1.00
two     float   2.00
result1         bssz    8
result2         bssz    8

.       starting procedure
        use     code
        def     prog
prog    align   0,8
        la      a5,a3,data              . A5 gets address of working section
        addaq   a0,a0,16                . allocate space for the parameter list
        lx      xa,a5,pi                . XA is loaded with the value of pi
        lx      xb,a5,two               . XB is loaded with the value of 2.0
        cpyxx   xc,xa                   . XA is copied to XC
        divf    xc,xb                   . XC <= pi/2
        sx      xc,a1,0                 . store the result in the stack
        sa      a1,a1,8                 . and its address one word later
        addaq   a4,a1,8                 . set A4 to the address of the parameter list
        ente    x0,0a5c(16)             . save A0-A5, XA-XC on the stack
        callseg sinr,a3,a4              . call MLP$RSIN
        sx      xf,a5,result1           . XF contains result of SIN(pi/2.0)
        lx      x2,a5,one               . load x2 with the value of 1.0
        ente    x0,0a5c(16)             . save A0-A5, XA-XC on the stack
        callseg sinv,a3,af              . parameter list not used
        sx      xf,a5,result2           . XF contains result of SIN(1.0)
        return
        end     prog
```

Figure 4-2. Assembler Program Calling the Math Library

C Calling the Math Library

The C language programmer has access to both the NOS/VE Math Library and the C/VE Math Library. See the C for NOS/VE Usage Manual for information on how to call the C/VE Math Library.

The following C program (figure 4-3) calls the NOS/VE Math Library SIND function to compute a full sine wave.

The SIND function uses call-by-reference, which means the function expects an address. Since this program is calling the Math Library and not a specific C function, the SIND function expects left-justified addresses. This program uses the left-bit-shift (<<) operator to left-justify the addresses.

```
/*  This C program uses the SIND function to compute a full sine wave.
*/

#define MAX_DEGREES 360

main()
{

    int count = 0,      /* loop counter                                 */
        address_deg;    /* left-justified address of the degree         */

    float degree,       /* 0 to 360 degrees                             */
        sin_of_degree,  /* Sine of degree                               */
        SIND();         /* declaration of the Math Library function     */
                        /* declaration must be capitalized              */

    for (count=1; count <= MAX_DEGREES; ++count)
    {

/*  Get the address of DEGREE.  Then use the left bit-shift operator (<<)
    to left justify the address 16 bits.  This is necessary because C
    uses a 48-bit right-justified pointer and NOS/VE expects left-justified
    addresses.
*/

      degree = count;

      address_deg = (int) &degree << 16;

      sin_of_degree = SIND(address_deg);

      printf("The sine of %3.0f is %f.\n", degree, sin_of_degree);

     }  /* end for loop */


}
```

Figure 4-3. C Program Calling the Math Library Using Call-by-Reference

Figure 4-4 illustrates how to call the Math Library using call-by-value. The C *#define* statement declares VMOD as a call-by-value routine. The call-by-reference Math Library function name is also illustrated.

NOTE

A Math Library function call from C must be capitalized.

```
/*  This C program names VMOD as an alias to the call-by-value entry point
    of MLP$VMOD.
*/
#define VMOD MLP$VMOD
main()
{
   int i = 83;
   int j = 8;
   int k;

   printf (" The size of short int is %d\n", sizeof(short int));
   printf (" The size of int is %d\n ", sizeof(int));
   printf (" The size of long int is %d\n ", sizeof(long int));
/*  Call MOD by reference.

   k = MOD((int)(&i)<<16,(int) (&j)<<16);
   printf (" The modulus of %d\n is", k);
/*  Call MOD by value.

   k = VMOD (i, j);
   printf (" The Mod of %d\n is", k);
   exit (0);
}
```

Figure 4-4. C Program Calling the Math Library Using Call-by-Value

CYBIL Calling the Math Library

CYBIL can call the Math Library with either a call-by-reference or a call-by-value entry point. Double precision functions can only be called with call-by-reference entry points.

NOTE

CYBIL parameters to Math Library routines must be VAR parameters.

The following example (figure 4-5) illustrates a call-by-reference hyperbolic sine (SINH) function.

```
MODULE math_example;

{ The following "*copyc" can be expanded by the following command:
{ EXPAND_SOURCE_FILE ..
{ file ALTERNATE_BASE=$SYSTEM.COMMON.PSF$EXTERNAL_INTERFACE_SOURCE

*copyc mlp$rsinh

FUNCTION hyperbolic_sine (
         VAR x: real ): real;

{ Parameters to Math Library routines are VAR parameters

hyperbolic_sine := mlp$rsinh ( x );

FUNCEND hyperbolic_sine;

MODEND math_example;
```

Figure 4-5. CYBIL Program Calling the Math Library Using Call-by-Reference

CYBIL programs can call complex numbers only if they are defined as type MLT$COMPLEX. Figure 4-6 illustrates how to call a complex function from CYBIL.

```
MODULE cmml_complex_example;
*copyc mlp$rccos


{     This module shows how to handle complex numbers in CYBIL.
{     Complex numbers are defined as type MLT$COMPLEX, which is
{     a two-word record.  CYBIL will not accept functions which
{     return a record.  Hence, the complex functions are defined
{     as returning longreal.  To convert a longreal to an mlt$complex,
{     use the CYBIL intrinsic #UNCHECKED_CONVERSION.  This example
{     calls the complex cosine routine.

  PROCEDURE complex_cosine
    (VAR z_in: mlt$complex;
     VAR z_out: mlt$complex);

    VAR
      d: longreal;

    d := mlp$rccos (z_in);
    #UNCHECKED_CONVERSION (d, z_out);
  PROCEND complex_cosine;

  PROGRAM example;

{      A FORTRAN program to print a complex value using FORTRAN I/O

    PROCEDURE [XREF] complex_print
      (VAR value: mlt$complex);

    VAR
      z,
      result: mlt$complex;

    z.real_ := 3.4;
    z.imag := -2.1;
    complex_cosine (z, result);
    complex_print (result);
  PROCEND example;
MODEND cmml_complex_example;
```

Figure 4-6. CYBIL Program Calling Complex Function CCOS Using MLT$COMPLEX

Figure 4-7 illustrates the FORTRAN program COMPLEX_PRINT which prints the complex result to the screen.

```
      SUBROUTINE complex_print (value)
C
      COMPLEX value
      PRINT *, value
      END
```

Figure 4-7. FORTRAN Program COMPLEX_PRINT

Figure 4-8 illustrates the NOS/VE commands needed to expand the source file and run the CYBIL and FORTRAN object code in order to print the CCOS result.

```
/expand_source_file $user.cmml_complex_example ..
../alternate_base=$system.psf$external_interface_source
/cybil compile b=cybil_binary
/fortran $user.complex_print b=fortran_binary
/execute_task (cybil_binary  fortran_binary)
```

Figure 4-8. NOS/VE Commands To Run CYBIL and FORTRAN Object Code

Figure 4-9 illustrates the output from this program.

```
(-4.006714482636,-1.027749704085)
```

Figure 4-9. CYBIL CCOS Output

For Better Performance

The Afterburner can eliminate call and return instructions and improve execution time.

The AFTERBURN_OBJECT_TEXT command optimizes FORTRAN and CYBIL programs by inlining subprograms. Inlining a subprogram places the subprogram statements where they are called, thus eliminating call and return instructions. This reduces the overhead associated with passing parameters, saving registers, and branching to and from the subprogram. See the section Improving Execution Time in the NOS/VE Object Code Management manual for additional information.

FORTRAN Version 1 Calling the Math Library

FORTRAN Version 1 supports calls to the Math Library and provides several language-specific intrinsic functions.

A FORTRAN Version 1 intrinsic function is a compiler-defined procedure that returns a single value. Intrinsic functions are referenced in the same way as user-written (external) function subprograms. If, in a particular program unit, a variable, array, or statement function is declared with the same name as an intrinsic function, the name cannot refer to the intrinsic function within that program unit. If a function subprogram is written with the same name as an intrinsic function, use of the name references the intrinsic function, unless the name is declared as the name of an external function with the EXTERNAL statement. (This is described in chapter 3 of the FORTRAN Version 1 for NOS/VE Language Definition manual.)

Intrinsic functions are typed by default and need not appear in any explicit type statement in the program. Explicitly typing a generic intrinsic function name does not remove the generic properties of the name. If you attempt to type an intrinsic function as something other than its default type, a warning message is displayed and the type statement is disregarded.

A function accepting integer, byte, real, complex, or double precision type arguments also accepts boolean arguments. A boolean argument is converted to integer, if integer is an allowable argument type, or to real, if real is an allowable argument type; otherwise, it is converted to double precision or complex, before computation. An IMPLICIT NONE statement does not affect the type of the results of any intrinsic functions.

Inlined Functions

The following functions are available for inlining by the FORTRAN Version 1 compiler:

ACOS
ALOG
ALOG10
ASIN
ATAN
COS
EXP
SIN
SQRT
TAN

FORTRAN Version 1 Uses Call-by-Value or Call-by-Reference

Most of the FORTRAN Version 1 intrinsic functions are in the Math Library and are accessed through the call-by-value routine. FORTRAN Version 1 calls Math Library functions with call-by-value unless call-by-reference is explicitly declared. To access an intrinsic function through the call-by-reference calling procedure, specify EXPRESSION_EVALUATION=REFERENCE (EE=R) on the FORTRAN command.

If an execution error occurs, the use of call-by-reference causes internal FORTRAN routines to generate descriptive error messages. If call-by-reference is not selected, the operating system produces error messages which generally provide less information.

NOTE

Always use normalized standard floating-point form for real, double precision, and complex arguments to intrinsic functions; unnormalized or nonstandard arguments can cause undefined results. FORTRAN automatically normalizes all real, double precision, and complex constants. Results of all floating-point operations (with standard normalized or zero operands) are normalized or zero. However, it is possible to generate unnormalized or nonstandard operands by means of boolean expressions, equivalencing, or various input operations.

The FORTRAN Version 1 intrinsic functions are summarized in table 4-3 (FORTRAN Version 2, as discussed in the following section, supports the same functions as well as a set of array-processing functions.) The functions are listed in alphabetical order by generic name or, where no generic name exists, by specific name. An asterisk in the Generic Name column indicates that the function is a Control Data extension. For specific names, the types of the arguments and results are shown. Boolean arguments are not listed in the table, but follow the conversion rules described above. Integer denotes 8-byte integer. Real denotes 8-byte real. Double precision denotes 16-byte real.

Table 4-3. FORTRAN Intrinsic Functions

Generic Name	Specific Names	Type of Argument	Type of Function	Description
ABS	IABS	Integer (2-byte)	Integer	Absolute value
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	ABS	Real	Real	
	DABS	Double	Double	
	CABS	Complex	Real	
ACOS	ACOS	Real	Real	Arccosine
	DACOS	Double	Double	
None	AIMAG	Complex	Real	Imaginary part of complex argument
AINT	AINT	Real	Real	Truncation
	DINT	Double	Double	
None	AMAX0	Integer	Real	Maximum value
None	AMIN0	Integer	Real	Minimum value
None	AND	Any type but character	Boolean	Boolean product
ANINT	ANINT	Real	Real	Nearest whole number
	DNINT	Double	Double	
ASIN	ASIN	Real	Real	Arcsine
	DASIN	Double	Double	
ATAN	ATAN	Real	Real	Arctangent
	DATAN	Double	Double	
ATAN2	ATAN2	Real	Real	Arctangent (two arguments)
	DATAN2	Double	Double	
None*	ATANH	Real	Real	Hyperbolic arctangent
BOOL*	—	Any type but logical	Boolean	Conversion to boolean
None	CHAR	Integer (2-byte)	Character	Integer conversion to character
		Integer (4-byte)	Character	
		Integer	Character	
		Byte	Character	
None*	COMPL	Any type but character	Boolean	Complement

Integer denotes full word (8-byte) integers. An asterisk indicates a Control Data extension.

(Continued)

Table 4-3. FORTRAN Intrinsic Functions *(Continued)*

Generic Name	Specific Names	Type of Argument	Type of Function	Description
None*	COTAN	Real	Real	Cotangent (argument in radians)
CMPLX	—	Integer (2-byte)	Complex	Conversion to complex
		Integer (4-byte)	Complex	
		Integer	Complex	
		Byte	Complex	
	—	Real	Complex	
	—	Double	Complex	
	—	Complex	Complex	
COS	COS	Real	Real	Cosine, argument in radians
	DCOS	Double	Double	
	CCOS	Complex	Complex	
None*	COSD	Real	Real	Cosine, argument in degrees
COSH	COSH	Real	Real	Hyperbolic cosine
	DCOSH	Double	Double	
None	CONJG	Complex	Complex	Negation of imaginary part
DBLE	—	Integer (2-byte)	Double	Conversion to double precision
		Integer (4-byte)	Double	
		Integer	Double	
		Byte	Double	
	—	Real	Double	
	—	Double	Double	
	—	Complex	Double	
DIM	IDIM	Integer (2-byte)	Integer	Positive difference
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	DIM	Real	Real	
	DDIM	Double	Double	
None	DPROD	Real	Double	Double precision product
None*	EQV	Any type but character	Boolean	Equivalence
None*	ERF	Real	Real	Error function
None*	ERFC	Real	Real	Complementary error function.

Integer denotes full word (8-byte) integers. An asterisk indicates a Control Data extension.

(Continued)

Table 4-3. FORTRAN Intrinsic Functions *(Continued)*

Generic Name	Specific Names	Type of Argument	Type of Function	Description
EXP	EXP DEXP CEXP	Real Double Complex	Real Double Complex	Exponential function
EXTB	None	a1: Any type but character a2,a3: Integer	Boolean	Extract a string of bits
None	ICHAR	Character	Integer	Character conversion to integer
None	INDEX	Character	Integer	Index of a substring
INSB	None	a1,a4: Any type but character a2,a3: Integer	Boolean	Insert a string of bits
INT	INT INT IFIX IDINT —	Integer (2-byte) Integer (4-byte) Integer Byte Real Real Double Complex	Integer Integer Integer Integer Integer Integer Integer Integer	Conversion to integer
None	LEN	Character	Integer	Length of character string
None	LGE	Character	Logical	Lexically greater than or equal
None	LGT	Character	Logical	Lexically greater than
None	LLE	Character	Logical	Lexically less than or equal
None	LLT	Character	Logical	Lexically less than
LOG	ALOG DLOG CLOG	Real Double Complex	Real Double Complex	Natural logarithm
LOG10	ALOG10 DLOG10	Real Double	Real Double	Common logarithm
None*	MASK	Integer or Boolean	Boolean	Mask

Integer denotes full word (8-byte) integers. An asterisk indicates a Control Data extension.

(Continued)

Table 4-3. FORTRAN Intrinsic Functions *(Continued)*

Generic Name	Specific Names	Type of Argument	Type of Function	Description
MAX	MAX0	Integer (2-byte)	Integer	Largest value
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	AMAX1	Real	Real	
	DMAX1	Double	Double	
None	MAX1	Real	Integer	Largest value
MIN	MIN0	Integer (2-byte)	Integer	Smallest value
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	AMIN1	Real	Real	
	DMIN1	Double	Double	
None	MIN1	Real	Integer	Smallest value
MOD	MOD	Integer (2-byte)	Integer	Remaindering
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	AMOD	Real	Real	
	DMOD	Double	Double	
None*	NEQV	Any type but character	Boolean	Nonequivalence
NINT	NINT	Real	Integer	Nearest integer
	IDNINT	Double	Integer	
None*	OR	Any type but character	Boolean	Boolean sum
PTR*	—	Any type	Boolean	Parameter address; used only when passing parameters to C or CYBIL routines
None*	RANF	None	Real	Random number generator

Integer denotes full word (8-byte) integers. An asterisk indicates a Control Data extension.

(Continued)

Table 4-3. FORTRAN Intrinsic Functions *(Continued)*

Generic Name	Specific Names	Type of Argument	Type of Function	Description
REAL	FLOAT	Integer (2-byte)	Real	Conversion to real
		Integer (4-byte)	Real	
		Integer	Real	
		Byte	Real	
	REAL	Integer	Real	
		Real	Real	
		Complex	Real	
	SNGL	Double	Real	
None*	SHIFT	Any type but character for a1; integer or Boolean for a2	Boolean	Shift
SIGN	ISIGN	Integer (2-byte)	Integer	Transfer of sign
		Integer (4-byte)	Integer	
		Integer	Integer	
		Byte	Integer	
	SIGN	Real	Real	
	DSIGN	Double	Double	
SIN	SIN	Real	Real	Sine (argument in radians)
	DSIN	Double	Double	
	CSIN	Complex	Complex	
None*	SIND	Real	Real	Sine (argument in degrees)
SINH	SINH	Real	Real	Hyperbolic sine
	DSINH	Double	Double	
SQRT	SQRT	Real	Real	Square root
	DSQRT	Double	Double	
	CSQRT	Complex	Complex	
SUM1S	—	Integer	Integer	Sum of 1 bits that are set in a word
		Real	Integer	
		Double	Integer	
		Complex	Integer	
TAN	TAN	Real	Real	Tangent (argument in radians)
	DTAN	Double	Double	
None*	TAND	Real	Real	Tangent (argument in degrees)
TANH	TANH	Real	Real	Hyperbolic tangent
	DTANH	Double	Double	
None*	XOR	Any type but character	Boolean	Exclusive OR

Integer denotes full word (8-byte) integers. An asterisk indicates a Control Data extension.

Table 6–2 (in chapter 6) shows the domain and range for a subset of the Math Library functions.

For Better Performance

The Afterburner can eliminate call and return instructions and improve execution time.

The AFTERBURN_OBJECT_TEXT command optimizes FORTRAN and CYBIL programs by inlining subprograms. Inlining a subprogram places the subprogram statements where they are called, thus eliminating call and return instructions. This reduces the overhead associated with passing parameters, saving registers, and branching to and from the subprogram. See Improving Execution Time in the NOS/VE Object Code Management manual for additional information.

FORTRAN Version 2 Calling the Math Library

FORTRAN Version 2 supports calls to the Math Library and provides the same language-specific intrinsic functions as FORTRAN Version 1. FORTRAN Version 2 also provides several array-processing functions in addition to the functions handled by the Math Library. FORTRAN Version 2 arguments can be array-valued.

Inlined Functions

The following functions are available for inlining by the FORTRAN Version 2 compiler:

ACOS
ALOG
ALOG10
ASIN
ATAN
COS
EXP
SIN
SQRT
TAN

The primary Math Library interface difference between FORTRAN Version 1 and FORTRAN Version 2 is that the arguments can be array-valued and the programs can be vectorized. Refer to chapter 7, Vector Processing, for a discussion of array-valued arguments.

For Better Performance

The Afterburner can eliminate call and return instructions and improve execution time.

The AFTERBURN_OBJECT_TEXT command optimizes FORTRAN and CYBIL programs by inlining subprograms. Inlining a subprogram places the subprogram statements where they are called, thus eliminating call and return instructions. This reduces the overhead associated with passing parameters, saving registers, and branching to and from the subprogram. See Improving Execution Time in the NOS/VE Object Code Management manual for additional information.

FORTRAN Function Summary

Table 4-4 lists the FORTRAN Version 1 and FORTRAN Version 2 intrinsic functions. For multiple-argument functions, a1 indicates argument 1, a2 indicates argument 2, and so forth. The generic and specific names are listed in alphabetical order. See the FORTRAN Version 1 or FORTRAN Version 2 manual for complete descriptions.

Table 4-4. FORTRAN Function Summary

Name	Source	Description
ABS	Math Library	Absolute value
ACOS	Math Library	Arccosine
AIMAG	Math Library	Imaginary part of complex argument
AINT	Math Library	Truncation
ALL	FORTRAN Version 2	True if every element of a1, along the optional dimension specification a2, has the logical value true
ALLOCATED	FORTRAN Version 2	Scalar logical value indicating whether or not an allocatable array is allocated
ALOG	Math Library	Natural logarithm
ALOG10	Math Library	Common logarithm (base 10)
AMAX0	FORTRAN Versions 1 and 2	Maximum value
AMAX1	FORTRAN Versions 1 and 2	Largest value
AMIN0	FORTRAN Versions 1 and 2	Minimum value
AMIN1	FORTRAN Versions 1 and 2	Smallest value
AMOD	Math Library	Remainder of a ratio (uses real numbers)
AND	FORTRAN Versions 1 and 2	Boolean product
ANINT	Math Library	Nearest whole number
ANY	FORTRAN Version 2	Logical value true is one or more elements of a1, along the optional dimension specification a2, has the logical value true
ASIN	Math Library	Arcsine
ATAN	Math Library	Arctangent
ATANH	Math Library	Hyperbolic arctangent
ATAN2	Math Library	Arctangent (two arguments)
BOOL	FORTRAN Versions 1 and 2	Conversion to boolean
CABS	Math Library	Absolute value
CCOS	Math Library	Cosine, argument in radians
CEXP	Math Library	Exponential function
CHAR	FORTRAN Versions 1 and 2	Integer conversion to character

(Continued)

Table 4-4. FORTRAN Function Summary *(Continued)*

Name	Source	Description
CLOG	Math Library	Natural logarithm
CMPLX	FORTRAN Versions 1 and 2	Conversion to complex
COMPL	FORTRAN Versions 1 and 2	Complement
CONJG	Math Library	Negation of imaginary part
COS	Math Library	Cosine, argument in radians
COSD	Math Library	Cosine, argument in degrees
OSH	Math Library	Hyperbolic cosine
COTAN	Math Library	Cotangent (argument in radians)
COUNT	FORTRAN Version 2	Number of true elements in a1 along the optional dimension specification a2
CSIN	Math Library	Sine (argument in radians)
CSQRT	Math Library	Square root
DABS	Math Library	Absolute value
DACOS	Math Library	Arccosine
DASIN	Math Library	Arcsine
DATAN	Math Library	Arctangent
DATAN2	Math Library	Arctangent (two arguments)
DBLE	FORTRAN Versions 1 and 2	Conversion to double precision
DCOS	Math Library	Cosine, argument in radians
DCOSH	Math Library	Hyperbolic cosine
DDIM	Math Library	Positive difference
DEXP	Math Library	Exponential function
DIM	Math Library	Positive difference
DINT	Math Library	Truncation
DLOG	Math Library	Natural logarithm
DLOG10	Math Library	Common logarithm
DMAX1	FORTRAN Versions 1 and 2	Largest value
DMIN1	FORTRAN Versions 1 and 2	Smallest value
DMOD	Math Library	Remainder of a ratio (uses double precision numbers)
DNINT	Math Library	Nearest whole number
DOTPRODUCT	FORTRAN Version 2	Dot product of a1 and a2
DPROD	Math Library	Double precision product
DSIGN	Math Library	Transfer of sign
DSIN	Math Library	Sine (argument in radians)
DSINH	Math Library	Hyperbolic sine

(Continued)

Table 4-4. FORTRAN Function Summary *(Continued)*

Name	Source	Description
DSQRT	Math Library	Square root
DTAN	Math Library	Tangent (argument in radians)
DTANH	Math Library	Hyperbolic tangent
EQV	FORTRAN Versions 1 and 2	Equivalence
FLOAT	FORTRAN Versions 1 and 2	Conversion to real
ERF	Math Library	Error function
ERFC	Math Library	Complementary error function
EXP	Math Library	Exponential function
EXTB	Math Library	Extract a string of bits
IABS	Math Library	Absolute value
ICHAR	FORTRAN Versions 1 and 2	Character conversion to integer
IDIM	Math Library	Positive difference
IDINT	FORTRAN Versions 1 and 2	Conversion to integer
IDNINT	Math Library	Nearest integer
IFIX	FORTRAN Versions 1 and 2	Conversion to integer
INDEX	FORTRAN Versions 1 and 2	Index of a substring
INSB	Math Library	Insert a string of bits
INT	FORTRAN Versions 1 and 2	Conversion to integer
ISIGN	Math Library	Transfer of sign
LBOUND	FORTRAN Version 2	Lower bound of dimension a2 of a1
LEN	FORTRAN Versions 1 and 2	Length of character string
LGE	FORTRAN Versions 1 and 2	Lexically greater than or equal
LGT	FORTRAN Versions 1 and 2	Lexically greater than
LLE	FORTRAN Versions 1 and 2	Lexically less than or equal
LLT	FORTRAN Versions 1 and 2	Lexically less than
LOG	FORTRAN Versions 1 and 2	Natural logarithm
LOG10	FORTRAN Versions 1 and 2	Common logarithm
MASK	FORTRAN Versions 1 and 2	Boolean result
MATMUL	FORTRAN Version 2	Product of arguments a1 and a2
MAX	FORTRAN Versions 1 and 2	Largest value
MAXVAL	FORTRAN Version 2	Maximum element of a1 along dimension a2 corresponding to true elements of a3

(Continued)

Table 4-4. FORTRAN Function Summary *(Continued)*

Name	Source	Description
MAX0	FORTRAN Versions 1 and 2	Largest value
MAX1	FORTRAN Versions 1 and 2	Largest value
MERGE	FORTRAN Version 2	Result containing the values of a1 corresponding to true elements of a3, and the values of a2 corresponding to false elements of a3
MIN	FORTRAN Versions 1 and 2	Smallest value
MINVAL	FORTRAN Version 2	Minimum element of a1 along dimension a2 corresponding to true elements of a3
MIN0	FORTRAN Versions 1 and 2	Smallest value
MIN1	FORTRAN Versions 1 and 2	Smallest value
MOD	Math Library	Remainder of a ratio
NEQV	FORTRAN Versions 1 and 2	Nonequivalence
NINT	Math Library	Nearest integer
OR	FORTRAN Versions 1 and 2	Boolean sum
PACK	FORTRAN Version 2	One-dimensional array consisting of all elements of a1 corresponding to true elements of a2
PRODUCT	FORTRAN Version 2	Product of elements in argument a1 along dimension a2 corresponding to .TRUE. elements of a3
PTR	FORTRAN Versions 1 and 2	Parameter address; used only when passing parameters to C or CYBIL routines
RANF	Math Library	Random number generator
RANK	FORTRAN Version 2	Number of dimensions in a1
REAL	FORTRAN Versions 1 and 2	Conversion to real
SEQ	FORTRAN Version 2	Returns a one-dimensional array
SHIFT	FORTRAN Versions 1 and 2	Shift

(Continued)

Table 4-4. FORTRAN Function Summary *(Continued)*

Name	Source	Description
SIGN	Math Library	Transfer of sign
SIN	Math Library	Sine (argument in radians)
SIND	Math Library	Sine (argument in (degrees)
SINH	Math Library	Hyperbolic sine
SIZE	FORTRAN Version 2	Size of an array
SNGL	FORTRAN Versions 1 and 2	Conversion to real
SQRT	Math Library	Square root
SUM	FORTRAN Version 2	Sum of elements in argument a1 along dimension a2 corresponding to .TRUE. elements in a3
SUM1S	Math Library	Sum of 1 bits that are set in a word
TAN	Math Library	Tangent (argument in radians)
TAND	Math Library	Tangent (argument in degrees)
TANH	Math Library	Hyperbolic tangent
UNBOUND	FORTRAN Version 2	Upper bound of dimension a2 of a1
UNPACK	FORTRAN Version 2	Array with the same shape as a3 and the same type as a1
XOR	FORTRAN Versions 1 and 2	Exclusive OR

Pascal Calling the Math Library

The Pascal compiler provides a transparent interface to several Math Library functions. The language also provides several predefined functions. Pascal makes no distinction between Math Library functions and predefined functions. In some cases Pascal uses a different name for a function actually provided by the Math Library (for example, the Pascal ARCTAN, ARCTAN2, and ARCTANH are different names for the Math Library functions ATAN, ATAN2, and ATANH, respectively).

The following functions are available to the Pascal programmer:

ABS	COTAN	POWER
ACOS	DIM	RANF
AMOD	ERF	SIGN
ANINT	ERFC	SIN
ARCTAN	EXP	SINH
ARCTAN2	IDIM	SQR
ARCTANH	ISIGN	SQRT
ASIN	LN	TAN
COS	LN10	TANH
COSH	NINT	

The Pascal function POWER combines the Math Library functions ITOI, ITOX, XTOI, and XTOX. POWER accepts integer or real arguments.

In NOS/VE Pascal, most math functions are called from the Math Library, except ABS and SQR are implemented directly by Pascal generated code. All Pascal function calls are transparent to the user.

Inlined Functions

At the user's option, the Pascal compiler generates inline code structures for the following functions:

ACOS
LN (ALOG)
LN (ALOG10)
ASIN
ARCTAN (ATAN)
COS
EXP
SIN
SQRT
TAN

Pascal Calling Routines

When you call Pascal functions, the compile time of your program is affected by the EXPRESSION_EVALUATION parameter on the PASCAL command. If you specify EXPRESSION_EVALUATION=REFERENCE, the compiler selects call-by-reference, which does more argument checking and may be slower. The default is EXPRESSION_EVALUATION=NONE, where the compiler selects call-by-value, which does less argument checking.

Pascal Math Function Attributes

Table 4-5 lists the domain and range for applicable Pascal math functions.

Table 4-5. Mathematical Intrinsic Functions

Function	Domain	Range
ACOS(a)	|a|≤1	0≤ACOS(a)≤ pi
ARCTAN(a)	−infinity≤a≤infinity	−pi/2≤ARCTAN(a)≤pi/2
ARCTAN2(a1,a2)	a2<0, a1<0 a2<0, a1≥0 a2=0, a1<0 a2=0, a1>0 a2>0, a1<0 a2>0, a1≥0 a2=0, a1=0 (error)	−pi<ARCTAN2(a1,a2)<−pi/2 pi/2≤ARCTAN2(a1,a2)≤pi ARCTAN2(a1,a2)=−pi/2 ARCTAN2(a1,a2)=pi/2 −pi/2<ARCTAN2(a1,a2)<0 0≤ARCTAN2(a1,a2)<pi/2
ARCTANH(a)	|a|≤1	All valid real quantities
ASIN(a)	|a|≤1	−pi/2≤ASIN(a)≤pi/2
COS(a)	|a|<2**47	−1≤COS(a)≤1
COSH(a)	|a|<4095*log(2)	COSH(a)≥1
COTAN(a)	|a|<2**47	All valid real quantities
ERF(a)	−infinity≤a≤infinity	−1≤ERF(a)≤1
ERFC(a)	−infinity≤a≤25.923	0≤ERFC(a)≤2
EXP(a)	a<4095*LOG(2)	All valid real quantities
LN(a)	a>0	|LN(a)|<4095*LN(a)
LN10(a)	a>0	|LN10(a)|<4095*LN(2) base 10
SIN(a)	|a|<2**47	−1≤SIN(a)≤1
SINH(a)	|a|<4095*log(2)	All valid real quantities
SQRT(a)	a≥0	SQRT(a)≥0
TAN(a)	|a|<2**47	All valid real quantities
TANH(a)	All valid real quantities	−1≤TANH(a)≤1

Error Handling 5

Error Handling 5

This chapter discusses two different kinds of errors:

- Processing errors (algorithm error and machine round-off error)
- Input errors (arguments that are out of range)

This chapter also discusses accuracy measurement and NOS/VE condition handling. Understanding how the Math Library handles errors and how the NOS/VE operating system handles conditions may improve your ability to use the Math Library as a resource.

NOTE

This chapter discusses Math Library error handling in general. See chapter 7, Vector Processing, for additional information on vector error handling.

Processing Error

Processing error is defined as the computed value of a function minus the true value.

A certain amount of processing error occurs during the computation of the Math Library functions, and is composed of two parts:

- Algorithm error
- Machine round-off error

Algorithm Error

Algorithm error is caused by inaccuracies inherent in the mathematical process used to compute the result. It includes error in coefficients used in the algorithm.

A curve representing the algorithm error is usually smooth with discontinuities at breaks in the range reduction technique. Error in the coefficients that are involved in range reduction can also occur. Usually, a good algorithm which uses good coefficients will not have an error greater than one-half in the last bit of the result.

Machine Round-Off Error

Machine round-off error is caused by the finite nature of the computer. Because only a finite number of bits can be represented in each word of memory, some precision is lost.

Round-off error is difficult to predict or graph. A graph of round-off error is extremely discontinuous, but maximum and minimum error over small intervals can be shown.

Input Error

Input error is handled differently by the call-by-reference and call-by-value routines. Error handling takes place when the argument or result is outside the range of the function.

If you are accessing the Math Library from a language other than FORTRAN, you can establish a condition handler to be used in conjunction with the error handling mechanism under the call-by-reference routine. The Math Library automatically establishes this condition handler for FORTRAN programs.

Call-By-Reference Error Handling

When the argument or result is out-of-range in a call-by-reference routine, an error message is displayed and the corresponding default error value is placed in the result registers XE and XF. Figure 5-1 is a Nassi-Shneiderman chart[1] illustrating the logical flow of call-by-reference error handling.

Call-By-Value Error Handling

If the call-by-value routine is called directly, that is, if the call-by-reference routine is not called, the job aborts if either of the following situations occurs:

- An out-of-range argument is passed to the call-by-value routine.
- The result of the computation in a call-by-value routine is out-of-range.

The call-by-value routine does not guarantee any other type of error handling, and the values in registers XE and XF are undefined unless otherwise specified.

1. Nassi-Shneiderman charts (also called Chapin charts) are read like flow charts: a rectangle indicates a process, an inverted isosceles triangle indicates a decision, and a right triangle indicates a branch from a decision.

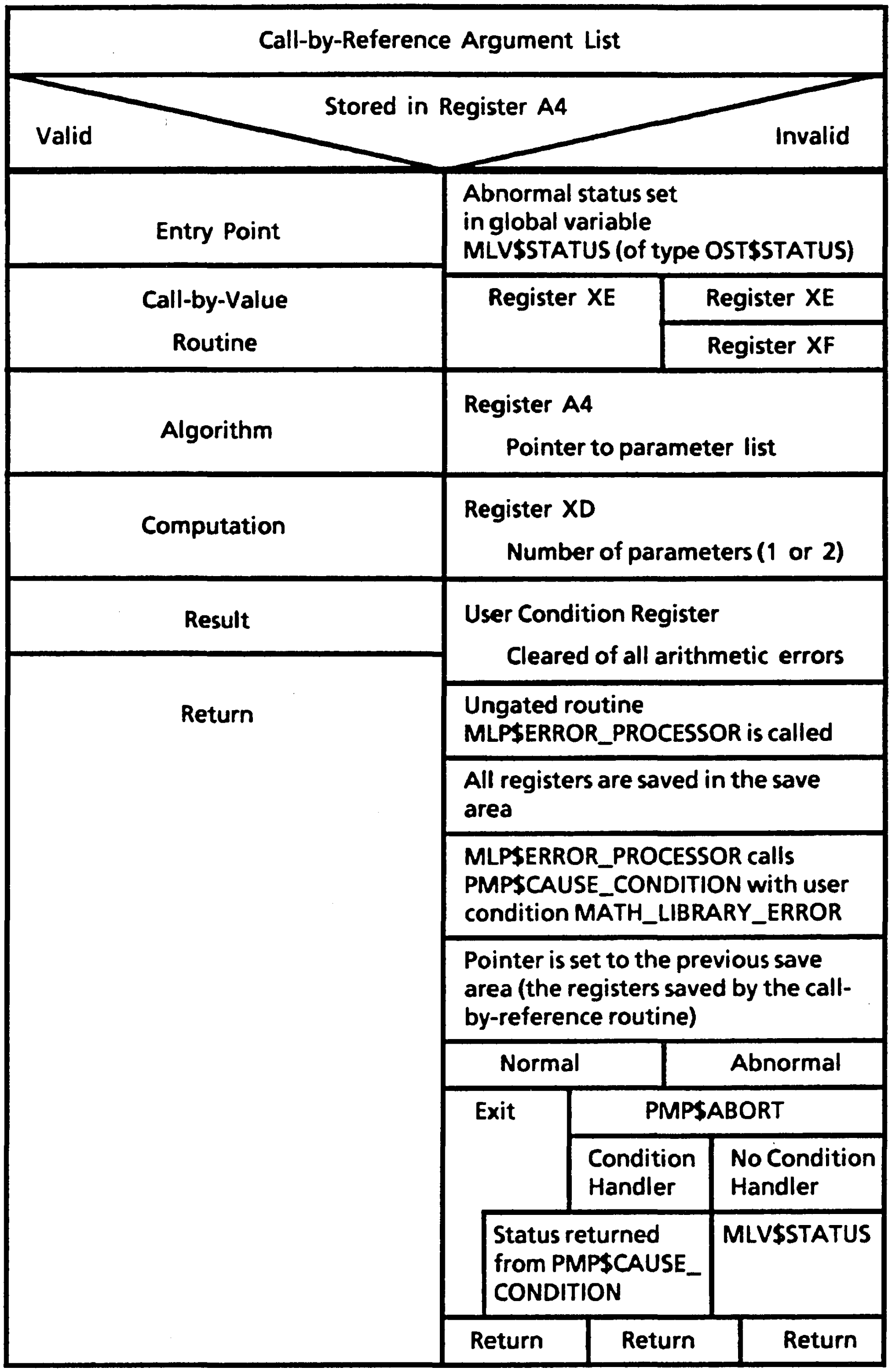


Figure 5-1. Logical Flow of Call-by-Reference Error Handling

Accuracy Measurements

When performance improvements are made to Math Library functions, the following accuracy measurements are calculated:

- Relative error
- Root mean square relative error

The Taylor series of a Math Library function is sometimes used in the calculation of relative error and root mean square relative error. For a discussion of Taylor series, refer to any calculus text (for example, *Calculus and Analytic Geometry* by G. B. Thomas). The following paragraphs discuss these accuracy measurements.

Relative Error

Relative error is the processing error divided by the true value. The magnitude of relative error can be analyzed in two ways:

- Using the relative error formula
- Examining bit error

Using the Relative Error Formula

Relative error can be calculated by using the following formula:

Relative Error = (Function Value - Exact Value) / Exact Value

This method is used for single precision algorithms accurate to less than 2E−15, and round-off errors less than 10E−15.

Changing the last bit in a single precision number produces a relative change between 3.5E−15 for a large mantissa and 7.1E−15 for a small but normalized mantissa. This method is used for the error analysis of the Math Library functions.

Examining Bit Error

The second method of analyzing relative error is finding out how many bits the routine differs from the exact value. This is called bit error.

To determine how many bits off a routine is, a function is evaluated in double precision and rounded to single precision. Then, assuming the exponents are the same, the mantissas are subtracted and the integer difference is the bit error.

Root Mean Square Relative Error

Root mean square error is the square root of the average of the squares of the relative errors of all the arguments.

NOS/VE Condition Handler

Under call-by-reference, the Math Library generates the special software condition MATH_LIBRARY_ERROR. The language under which you are executing ordinarily handles the processing of this condition. If no condition handler for MATH_LIBRARY_ERROR has been established, NOS/VE handles the processing of this condition.

You can also write your own condition handler. NOS/VE provides two mechanisms for specifying the action to be taken when an abnormal condition occurs:

- Error processing
- Condition handling

Error Processing

Error processing is available when the STATUS parameter is included in a NOS/VE command and the command terminates with an abnormal status.

All NOS/VE commands have an optional parameter called STATUS. When you specify the STATUS parameter, you must supply a previously declared variable of type STATUS as its value. This variable is used by the System Command Language (SCL) interpreter to hold the completion status of the command.

If you include the STATUS parameter on a command, the SCL interpreter proceeds to the next command even if an abnormal condition is encountered. Most commands do not inform you of an error if you include the STATUS parameter, but succeeding commands may check the contents of the status variable and alter the flow of statements based on abnormal conditions.

If you do not include a value for the STATUS parameter and an error occurs, the SCL interpreter skips succeeding commands in the current input block as described in the NOS/VE System Usage manual.

Condition Handling

When you specify the STATUS parameter on a command, you can alter the command stream based on the completion status of the command. You can also provide condition handlers to alter the command stream in the event of certain system conditions.

Condition handling is activated when a condition exists for a command that does not contain a STATUS parameter, or is beyond the scope of STATUS variable error processing. When condition handling is activated, your batch or interactive job is usually terminated. If you receive a condition handling error, see the NOS/VE Diagnostic Messages manual for a description of the error and a recommended action.

The following information defines the interface between the Math Library and the operating system, whether or not a condition handler has been established. For detailed information on the procedures used in establishing a user-defined condition handler, see the NOS/VE System Usage manual.

When an error occurs in a Math Library function under a call-by-reference routine, the following events occur:

1. An appropriate abnormal status is set into global variable MLV$STATUS (of type OST$STATUS).

2. The appropriate default error value is placed in the result register(s) XE and/or XF. Register A4 contains the pointer to the parameter list passed to the call-by-reference routine. Register XD contains the number of parameters for the call-by-reference routine. For example, register XD will contain a 1 for the call-by-reference routine MLP$RSIN, and a 2 for MLP$RZTOZ. The User Condition Register is cleared of all arithmetic errors.

3. Ungated routine MLP$ERROR_PROCESSOR is called with all registers saved in the save area so that they can be accessed by a condition handler.

4. MLP$ERROR_PROCESSOR calls the PMP$CAUSE_CONDITION procedure with user condition MATH_LIBRARY_ERROR and a pointer to the previous save area (the registers saved by the call-by-reference routine) as the condition descriptor.

5. Upon return from the PMP$CAUSE_CONDITION procedure, MLP$ERROR_PROCESSOR is exited if the returned status is normal. If the return status is not normal, the PMP$ABORT procedure is called with one of two conditions:

   - If no established condition handler exists for MATH_LIBRARY_ERROR, then status MLV$STATUS is used.

   - If an established condition handler does exist for MATH_LIBRARY_ERROR, then the status returned from the PMP$CAUSE_CONDITION procedure is used.

6. The call-by-reference routine immediately returns to the calling program if it is returned to from MLP$ERROR_PROCESSOR.

Refer to chapter 7, Vector Processing, for a discussion of vector error handling.

Scalar Classification Tables 6

Scalar Classification Tables 6

This chapter provides a series of tables that categorize the Math Library functions according to various classifications and provide information such as domains and ranges and types of results.

Table 6-1 gives a summary of the math functions, grouping the functions by related generic and specific function names (alphabetized by generic name). COSD, SIND, and TAND are grouped with COS, SIN, and TAN, respectively. COSD, SIND, and TAND are not related to the generic functions because they return results in degrees.

The functions in table 6-1 are grouped as follows:

- Absolute value (ABS, CABS, DABS, IABS)
- Inverse cosine (ACOS, DACOS)
- Imaginary part of a complex argument (AIMAG)
- Truncation (AINT, DINT)
- Natural logarithm (ALOG, CLOG, DLOG)
- Common logarithm (ALOG10, DLOG10)
- Remaindering (AMOD, DMOD, MOD)
- Nearest whole number (ANINT, DNINT)
- Inverse sine (ASIN, DASIN)
- Inverse tangent (ATAN, ATAN2, DATAN, DATAN2)
- Cosine (CCOS, COS, DCOS)
- Conjugate (CONJG)
- Cotangent (COTAN)
- Exponential (CEXP, DEXP, EXP)
- Hyperbolic cosine (COSH, DCOSH)
- Sine (CSIN, DSIN, SIN, SIND)
- Square root (CSQRT, DSQRT, SQRT)
- Inverse hyperbolic tangent (ATANH)
- Positive difference (DDIM, DIM, IDIM)
- Product (DPROD)
- Transfer of sign (DSIGN, ISIGN, SIGN)
- Hyperbolic sine (DSINH, SINH)
- Tangent (DTAN, TAN)
- Hyperbolic tangent (DTANH, TANH)
- Error function (ERF)
- Complementary error function (ERFC)
- Extract bits (EXTB)
- Nearest integer (IDNINT, NINT)
- Insert bits (INSB)
- Random number generator (RANF)
- Returns random number seed (RANGET)
- Sets seed for random number generator (RANSET)
- Sum of 1 bits in one word (SUM1S)

Table 6-2 shows the input domain[1] and output range for all of the math functions, except the exponentiation functions. For the exponentiation functions, table 6-3 lists the bases, exponents, and results and table 6-4 summarizes the domains and ranges.

1. Indefinite and infinite are not in the input domain unless specifically stated. This applies to tables 6-2 and 6-4.

Summary of Math Functions

Table 6-1. Mathematical Functions

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Absolute value	|x|; if x is complex, square root of ((real x)**2 + (imag x)**2)	ABS CABS DABS IABS	Real Complex Double Integer	1	Real Real Double Integer
Inverse cosine	arccos(x)	ACOS DACOS	Real Double	1	Real Double
Imaginary part of a complex argument	Imaginary part of (xr,xi) = xi	AIMAG	Complex	1	Real
Truncation	int(x)	AINT DINT	Real Double	1	Real Double
Natural logarithm	log e (x)	ALOG CLOG DLOG	Real Complex Double	1	Real Complex Double
Common logarithm	log 10 (x)	ALOG10 DLOG10	Real Double	1	Real Double
Remain-dering	x − int(x/y)*y	AMOD DMOD MOD	Real Double Integer	1	Real Double Integer
Nearest whole number	int(x + 0.5), if x ≥ 0 int(x − 0.5), if x < 0	ANINT DNINT	Real Double	1	Real Double
Inverse sine	arcsin(x)	ASIN DASIN	Real Double	1	Real Double

(Continued)

Table 6-1. Mathematical Functions *(Continued)*

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Inverse tangent	arctan(x)	ATAN	Real	1	Real
		DATAN	Double		Double
	arctan(y/x)	ATAN2	Real	2	Real
		DATAN2	Double		Double
Cosine	cos(x), where x is in radians	CCOS	Complex	1	Complex
		COS	Real		Real
		DCOS	Double		Double
	cos(x), where x is in degrees	COSD	Real	1	Real
Conjugate	Negation of imaginary part (xr,−xi)	CONJG	Complex	1	Complex
Cotangent	cotan(x), where x is in radians	COTAN	Real	1	Real
Exponential	e**x	CEXP	Complex	1	Complex
		DEXP	Double		Double
		EXP	Real		Real
Hyperbolic cosine	cosh(x)	COSH	Real	1	Real
		DCOSH	Double		Double
Sine	sin(x), where x is in radians	CSIN	Complex	1	Complex
		DSIN	Double		Double
		SIN	Real		Real
	sin(x), where x is in degrees	SIND	Real	1	Real
Square root	x**(1/2)	CSQRT	Complex	1	Complex
		DSQRT	Double		Double
		SQRT	Real		Real
Inverse hyperbolic tangent	arctanh(x)	ATANH	Real	1	Real

(Continued)

Table 6-1. Mathematical Functions *(Continued)*

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Positive difference	x − y, if x > y 0, if x ≤ y	DDIM DIM IDIM	Double Real Integer	2	Double Real Integer
Product	x*y	DPROD	Real	2	Double
Transfer of sign	|x|, if y ≥ 0 −|x|, if y < 0	DSIGN ISIGN SIGN	Double Integer Real	2	Double Integer Real
Hyperbolic sine	sinh(x)	DSINH SINH	Double Real	1	Double Real
Tangent	tan(x), where x is in radians tan(x), where x is in degrees	DTAN TAN TAND	Double Real Real	1 1	Double Real Real
Hyperbolic tangent	tanh(x)	DTANH TANH	Double Real	1	Double Real
Error function	erf(x)	ERF	Real	1	Real
Complementary error function	1 − erf(x)	ERFC	Real	1	Real
Extract bits	extb(x, i1, i2); extracts bits from x starting with position i1 with length of i2	EXTB	x: Boolean Complex Double Integer Logical Real i1: Integer i2: Integer	3	Boolean

(Continued)

Table 6-1. Mathematical Functions *(Continued)*

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Nearest integer	int(x + 0.5), if x ≥ 0	IDNINT	Double	1	Integer
	int(x − 0.5), if x < 0	NINT	Real		Integer
Insert bits	insb(x, i1, i2, y); inserts bits from x starting with position i1 with length of i2 into copy of y	INSB	x,y: Boolean Complex Double Integer Logical Real i1: Integer i2: Integer	4	Boolean
Random number generator	Random number in range (0,1)	RANF	None	0	Real
Returns random number seed	Seed is in range (0,1)	RANGET	Real	1	Real
Sets seed for random number generator	ranset(x)	RANSET	Real	1	Real
Sum of 1 bits in one word	sum1s(x)	SUM1S	Boolean Complex Double Integer Real	1	Integer

Input Domains and Output Ranges

Table 6-2. Input Domains and Output Ranges

Function	Input Domain	Output Range
ACOS(x)	$\|x\| \leq 1$	$0 \leq$ ACOS(x) $\leq$ pi
DACOS(x)	$\|x\| \leq 1$	$0 \leq$ DACOS(x) $\leq$ pi
ALOG(x)	$x > 0$	|ALOG(x)| < 4095*log(2)
CLOG(xr,xi)	(xr, xi) $\neq$ (0,0) (xr**2 + xi**2)**1/2 in machine range	-pi < CLOG(xi) $\leq$ pi
DLOG(x)	$x > 0$	|DLOG (x)| < 4095*log(2)
ALOG10(x)	$x > 0$	|ALOG10(x)| < 4095*log(2) base 10
DLOG10(x)	$x > 0$	|DLOG10(x)| < 4095*log(2) base 10
ASIN(x)	$\|x\| \leq 1$	-pi/2 $\leq$ ASIN(x) $\leq$ pi/2
DASIN(x)	$\|x\| \leq 1$	-pi/2 $\leq$ DASIN(x) $\leq$ pi/2
ATAN(x)	-infinity $\leq$ x $\leq$ infinity	-pi/2 $\leq$ ATAN(x) $\leq$ pi/2
DATAN(x)	-infinity $\leq$ x $\leq$ infinity	-pi/2 $\leq$ DATAN(x) $\leq$ pi/2
ATAN2(x,y)	$y < 0, x < 0$	-pi < ATAN2(x,y) < - pi/2
	$y < 0, x \geq 0$	pi/2 $\leq$ ATAN2(x,y) $\leq$ pi
	$y = 0, x < 0$	ATAN2(x,y) = -pi/2
	$y = 0, x > 0$	ATAN2(x,y) = pi/2
	$y > 0, x < 0$	-pi/2 < ATAN2(x,y) < 0
	$y > 0, x \geq 0$	0 $\leq$ ATAN2(x,y) < pi/2
DATAN2(x,y)	$y < 0, x < 0$	-pi < DATAN2(x,y) < - pi/2
	$y < 0, x \geq 0$	pi/2 $\leq$ DATAN2(x,y) $\leq$ pi
	$y = 0, x < 0$	DATAN2(x,y) = -pi/2
	$y = 0, x > 0$	DATAN2(x,y) = pi/2
	$y > 0, x < 0$	-pi/2 < DATAN2(x,y) < 0
	$y > 0, x \geq 0$	0 $\leq$ DATAN2(x,y) < pi/2
ATANH(x)	$\|x\| < 1$	The set of valid real quantities.
COS(x)	|x| < 2**47	-1 $\leq$ COS(x) $\leq$ 1
CCOS(xr,xi)	|xr| < 2**47 |xi| < 4095*log(2)	-1 $\leq$ CCOS(x) $\leq$ 1
DCOS(x)	|x| < 2**47	-1 $\leq$ DCOS(x) $\leq$ 1
COSD(x)	|x| < 2**47	-1 $\leq$ COSD(x) $\leq$ 1
COSH(x)	|x| < 4095*log(2)	COSH(x) $\geq$ 1
DCOSH(x)	|x| < 4095*log(2)	DCOSH(x) $\geq$ 1

(Continued)

Table 6-2. Input Domains and Output Ranges *(Continued)*

Function	Input Domain	Output Range
COTAN(x)	0 < |x| < 2**47	The set of valid real quantities.
ERF(x)	-infinity ≤ x ≤ infinity	0 ≤ ERF(x) ≤ 1
ERFC(x)	-infinity ≤ x ≤ infinity	0 ≤ ERFC(x) ≤ 2
EXP(x)	x < 4095*log(2) and x ≥ -4097*log(2)	0 < EXP(x)
CEXP(xr,xi)	xr < 4095*log(2) and xr > -4097*log(2) xi < 2**47 xi ≥ -4097*log(2)	The set of valid complex quantities.
DEXP(x)	x < 4095*log(2) & x ≥ -4097*log(2)	The set of valid double precision quantities.
SIN(x)	|x| < 2**47	-1 ≤ SIN(x) ≤ 1
CSIN(xr,xi)	|xr| < 2**47 |xi| < 4095*log(2)	
DSIN(x)	|x| < 2**47	-1 ≤ DSIN(x) ≤ 1
SIND(x)	|x| < 2**47	-1 ≤ SIND(x) ≤ 1
SINH(x)	|x| < 4095*log(2)	
DSINH(x)	|x| < 4095*log(2)	
SQRT(x)	x ≥ 0	SQRT(x) ≥ 0
CSQRT(xr,xi)	(xr**2 + xi**2)**1/2 + |xr| in machine range	Value in right half of plane CSQRT(xr) ≥ 0)
DSQRT(x)	x ≥ 0	The set of valid double precision quantities.
TAN(x)	|x| < 2**47	The set of valid real quantities.
DTAN(x)	|x| < 2**47	The set of valid double precision quantities.
TAND(x)	|x| < 2**47 x cannot be exact odd multiple of 90	The set of valid real quantities.
TANH(x)	-infinity ≤ x ≤ infinity	-1 ≤ TANH(x) ≤ 1

Exponentiation Functions

Table 6-3 illustrates that the result type of an exponentiation function is determined by the order of precedence of the two input arguments. The result of exponentiation always takes the type of the argument with the higher precedence according to the following hierarchy:

1. Integer (the lowest precedence)

2. Single precision floating-point

3. Double precision floating-point

4. Complex (the highest precedence)

Table 6-3 lists the bases, exponents, and result types of the exponentiation functions by order of precedence. Table 6-4 summarizes the input domains and output ranges of the exponentiation functions.

Table 6-3. Arguments and Results of the Exponentiation Functions

Name	Base	Exponent	Result Type[1]
ITOI	Integer	Integer	Integer
ITOX	Integer	Single precision floating-point	Single precision floating-point
ITOD	Integer	Double precision floating-point	Double precision floating-point
ITOZ	Integer	Complex	Complex
XTOI	Single precision floating-point	Integer	Single precision floating-point
XTOX	Single precision floating-point	Single precision floating-point	Single precision floating-point
XTOD	Single precision floating-point	Double precision floating-point	Double precision floating-point
XTOZ	Single precision floating-point	Complex	Complex
DTOI	Double precision floating-point	Integer	Double precision floating-point
DTOX	Double precision floating-point	Single precision floating-point	Double precision floating-point
DTOD	Double precision floating-point	Double precision floating-point	Double precision floating-point
DTOZ	Double precision floating-point	Complex	Complex
ZTOI	Complex	Integer	Complex
ZTOX	Complex	Single precision floating-point	Complex
ZTOD	Complex	Double precision floating-point	Complex
ZTOZ	Complex	Complex	Complex

1. The argument (base or exponent) with the higher precedence always determines the number type of the result.

Table 6-4. Domains and Ranges of the Exponentiation Functions

Name	Type of Argument	Input Domain	Output Range
ITOI	Integer Integer	|x**y| < 2**63; if x = 0, then y > 0	The set of valid integer quantities
ITOX	Integer Real	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
ITOD	Integer Double	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
ITOZ	Integer Complex	x ≥ 0; if x = 0, then yr > 0, yi = 0	x**y ≥ 0
XTOI	Real Integer	if x = 0, then y > 0	The set of valid real quantities
XTOX	Real Real	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
XTOD	Real Double	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
XTOZ	Real Complex	if x = 0, then yr > 0, yi = 0	The set of valid complex quantities
DTOI	Double Integer	if x = 0, then y > 0	The set of valid double precision quantities
DTOX	Double Real	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
DTOD	Double Double	x ≥ 0; if x = 0, then y > 0	x**y ≥ 0
DTOZ	Double Complex	if x = 0, then yr > 0, yi = 0	The set of valid double precision quantities

(Continued)

Table 6-4. Domains and Ranges of the Exponentiation Functions *(Continued)*

Name	Type of Argument	Input Domain	Output Range
ZTOI	Complex Integer	if (xr,xi) = (0,0), then y > 0	The set of valid complex quantities
ZTOX	Complex Real	if (xr,xi) = (0,0) then y > 0	The set of valid complex quantities
ZTOD	Complex Double	if (xr,xi) = (0,0) then y > 0	The set of valid complex quantities
ZTOZ	Complex Complex	if (xr,xi) = (0,0) then y > 0, yi = 0	The set of valid complex quantities

Vector Processing 7

Vector Processing 7

This chapter discusses the vector processing capabilities of the Math Library, including both single argument and double argument vector math functions. This chapter also discusses vector error handling.

Vector Functions

Vector math functions accept vectors as arguments and return vectors as results. A vector is a one-dimensional set of numbers.

While the vector math functions are available and can be referenced on any CYBER 180 mainframe model, they perform array-processing only on models which include vector hardware facilities, currently limited to the CYBER 990/995 Series running FORTRAN Version 2.

If a vector math function is called on a non-vector machine, an unimplemented instruction trap (hardware condition) may occur (the vectorization is not implemented).

The FORTRAN Version 2 compiler guarantees that the length (L) of the vector sent to the Math Library will be within the range $0 \leq L \leq 512$ words. When the vector length is not within this valid range, an error message is displayed. See the section in this chapter entitled Vector Error Handling. When the length of the vector argument sent to the Math Library vector routine is zero, no operation occurs and the contents of the vector are returned without any values changed.

Vector Function Calling Routines

Scalar functions (depending upon the calling language) can be called by a call-by-reference or a call-by-value calling routine (linkage). Vector routines always use the call-by-reference linkage.

Under call-by-reference, register A4 points to the actual parameter list. Vector routines can have four different parameter lists as described in tables 7-1 through 7-4.

The calling sequence for all vector functions has one entry point defined for each function. In all cases, register A4 contains the Process Virtual Address (PVA) to the first entry in the parameter list.

The Math Library provides two types of vector processing functions:

- Single Argument Vector Math Functions
- Double Argument Vector Math Functions

The following sections discuss these types of functions.

Single Argument Vector Math Functions

The Math Library provides the following single argument vector processing functions:

ACOS	CSIN	DTAN
ALOG	CSQRT	DTANH
ALOG10	DACOS	ERF
ASIN	DASIN	ERFC
ATAN	DATAN	EXP
ATANH	DCOS	SIN
CCOS	DCOSH	SIND
CEXP	DEXP	SINH
CLOG	DLOG	SQRT
COS	DLOG10	TAN
COSD	DSIN	TAND
COSH	DSINH	TANH
COTAN	DSQRT	

Table 7-1 describes the internal representation of the parameter list for real, double precision, and complex single argument vector math functions. Single valued vector routines always follow this format. Each word is a decimal value.

Table 7-1. Parameter List for Single Argument Vector Math Functions

Word	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array.
Word 3	Pointer to the result array descriptor.
Word 4	Pointer to the source array descriptor.

Double Argument Vector Math Functions

The Math Library provides the following double argument vector processing functions:

ATAN2	DTOZ	XTOZ
DATAN2	ITOZ	ZTOD
DTOD	XTOD	ZTOI
DTOI	XTOI	ZTOX
DTOX	XTOX	ZTOZ

The double argument vector math functions are divided into three categories:

function_name(scalar, vector) See table 7-2 for a (scalar, vector) parameter list.

function_name(vector, scalar) See table 7-3 for a (vector, scalar) parameter list.

function_name(vector, vector) See table 7-4 for a (vector, vector) parameter list.

where *function_name* is a double argument function name, such as ATAN2.

Double Argument Vector Math Functions (Scalar, Vector)

Table 7-2 provides the internal representation of the parameter list for double argument vector math functions where argument 1 is scalar and argument 2 is vector. Each word is a decimal value.

Table 7-2. Parameter List for (Scalar, Vector) Functions

Word	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source scalar (argument 1).
Word 3	Pointer to the source array (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	0
Word 6	Pointer to the source array descriptor (argument 2).

Double Argument Vector Math Functions (Vector, Scalar)

Table 7-3 provides the internal representation of the parameter list for double argument vector math functions where argument 1 is vector and argument 2 is scalar. Each word is a decimal value.

Table 7-3. Parameter List for (Vector, Scalar) Functions

Word	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array (argument 1).
Word 3	Pointer to the source scalar (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	Pointer to the source array descriptor (argument 1).
Word 6	0

Double Argument Vector Math Functions (Vector, Vector)

Table 7-4 provides the internal representation of the parameter list for double argument vector math functions where argument 1 is vector and argument 2 is vector. Each word is a decimal value.

Table 7-4. Parameter List for (Vector, Vector) Functions

Word	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array (argument 1).
Word 3	Pointer to the source array (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	Pointer to the source array descriptor (argument 1).
Word 6	Pointer to the source array descriptor (argument 2).

Result Array and Source Array Descriptors

Table 7-5 provides the internal representation of the result array descriptor and the source array descriptor for all vector math functions.

Table 7-5. Result Array and Source Array Data Locations

Word	Description of Contents
Word 1	Number of elements in vector.
Word 2	Distance (or stride) measured in terms of array elements between two consecutive elements of the same dimension. Always equal to one for the Math Library.
Word 3	Lower bound of array. Always zero for the Math Library.

Vector Error Handling

The vector math functions use call-by-reference error handling. For example, if an argument within a set of arguments is illegal or produces an out-of-range value, an error message is displayed for that argument. The first argument in error is supplied in the error message. The default error value (usually an indefinite value indicated by +IND) is placed in the result location corresponding to the argument in error within the set.

Processing continues and correct results are generated for all arguments which are not in error. However, once an argument is found to be in error, further arguments which are in error are not detected and results are not guaranteed.

NOTE

For all vector routines, only the first illegal or out-of-range-producing argument produces an error message.

Function Descriptions 8

Function Descriptions 8

This chapter provides a summary of each Math Library function. The functions are organized alphabetically. Each function description includes the following:

- A description including the entry points for the function and the input domains and output ranges for the arguments in each function
- The call-by-reference routine
- The call-by-value routine
- An example call from Ada, C, FORTRAN, or Pascal

The following additional information is included, if applicable:

- Conditions that cause an argument to be invalid, resulting in an error
- The vector routine
- Formulas used to compute the result
- Error analysis and the effect of argument error

Entry points to the call-by-reference and call-by-value routines are places in the routines where execution can begin. Some routines can evaluate more than one function (for example, one algorithm may calculate a generic function and a specific function). Some routines call auxiliary routines (as described in chapter 9, Auxiliary Routines) to compute a portion of the function.

NOTE

If a category of information is not applicable (for example, Vector Routine, Error Analysis, or Effect of Argument Error), it is omitted from the function description.

Generic and Specific Names

Some functions have a generic name and one or more specific names. For example, ABS is a generic name; CABS, DABS, and IABS are specific names. For these functions, either the generic name or one of the specific names can be used. The generic name provides more flexibility because it can be used with any of the valid data types; except for functions performing type conversion, nearest integer, and absolute value with a complex argument, the type of the argument determines the type of the result.

A 2-byte or 4-byte integer or byte value, used as an argument to a function, is converted to a full word (8-byte) integer before being used as an argument. The conversion does not change the sign of the argument. A function accepting integer, real, complex or double precision type arguments also accepts boolean arguments. A boolean argument is converted to integer if it is an allowable argument type; otherwise, it is converted to real before computation. However, only a specific name can be used as an actual argument when passing the function name to a user-defined subprogram. Using a specific name requires a specific argument type.

For example, the generic function LOG computes the natural logarithm of an argument. Its argument can be real, double precision, complex or boolean (converted to real). The type of the result is the same as the type of the argument. Specific functions ALOG, DLOG, and CLOG also compute the natural logarithm. The specific function ALOG computes the log of a real or boolean argument and returns the result. The specific function DLOG is for double precision (or boolean) arguments and double precision results and the specific function CLOG is for complex (or boolean) arguments and complex results.

ABS

ABS computes the absolute value of an argument. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RABS and ABS, and the call-by-value entry point is MLP$VABS.

The input domain is the collection of all valid real quantities. The output range is included in the set of nonnegative real quantities.

Call-By-Reference Routine

No errors are generated by ABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with its sign bit forced positive. The rightmost 63 bits remain the same.

Example of ABS Called From FORTRAN

Source Code:

```
      PROGRAM ABS_EXAMPLE
C
      EXTERNAL ABS
      REAL r,t
      r=-88.9
      t=ABS(r)
      PRINT *, 'Absolute value = ', t
      END
```

Output:

```
Absolute value = 88.9
```

ACOS

ACOS computes the inverse cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RACOS and ACOS, the call-by-value entry point is MLP$VACOS, and the vector entry point is MLP$ACOSV.

The input domain is the collection of all valid real quantities in the interval [−1.0,1.0]. The output range is included in the set of nonnegative real quantities less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),  x < -.5
arcos(x)  = pi - arcos(-x),  x < -.5
arcsin(x) = x + x**3*s*((w + z -j)*w + a + m/(e - x**2)),
  where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),  -.5 <= x < .5
arcsin(x) = pi/2 - arcos(x),  .5 <= x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,  .5 <= x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
        w  = (x**2 - c)*z + k
        z  = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =   3.173 170 078 537 13
e =   1.160 394 629 739 02
m =  50.319 055 960 798 3
c =  -2.369 588 855 612 88
i =   8.226 467 970 799 17
j = -35.629 481 597 455 5
k =  37.459 230 925 758 2
a = 349.319 357 025 144
s =    .746 926 199 335 419*10**-3
```

The approximation of arcsin(−.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

```
a.  If ACOS entry, go to step g.

b.  If |x| > = .5, go to step h.

c.  n = 0 (Loop counter).
    q = x
    y = x**2
    u = 0, if ASIN entry.
    u = pi/2, if ACOS entry.

d.  z = (y + r)*y + i
    w = (y - c )*z + k
    p = q + s*q*y*((w + z - j)*w + a + m/(e - y))
    p = u - p
    Y1 = p/2**n

e.  If ASIN entry, go to step k.

f.  If x is in (-.5,1.0), return.
    XF = 2*u - (Y1)
    Return.

g.  If |x| < .5, go to step c.

h.  If x = 1.o or -1.0, go to step 1.
    If x is invalid, go to step m.
        n = 0 (Loop counter).
        y = 1.0 - |x|, and normalize y.

i.  h = 4*y - 2*y**2
    n = n + 1.0
    If 2*y ≤ 2 - sqrt(3) = .267949192431, y = h, and go to step i.

j.  q = 1.0 - h, and normalize q.
    y = q**2
    u = pi/2
    Go to step d.

k.  Y1 = u - (Y1), and normalize Y1.
    Affix sign of x to Y1 = XF.
    Return.

l.  XF = pi/2, if x = 1.0.
    XF = -pi/2, if x = -1.0.
    If ASIN entry, return.
    XF = 0, if x = 1.0.
    XF = pi, if x = -1.0.
    Return.

m.  Return.
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of relative error of the ACOS approximation over (-.5,.5) is 1.996*E-15.

The function ACOS was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-1 shows a summary of these statistics.

Table 8-1. Relative Error of ACOS

Interval From	Interval To	Maximum	Root Mean Square
-.1250E+00	.1250E+00	.4916E-14	.3233E-14
-.1000E+01	-.7500E+00	.5875E-14	.2068E-14
.7500E+00	.1000E+01	.1987E-13	.7749E-14

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/(1.0 − x**2)**.5.

Example of ACOS Called From FORTRAN

Source Code:

```
      PROGRAM ACOS_EXAMPLE
C
      x=0.5
      PRINT *, 'Inverse cosine of x is:'
      PRINT *, ACOS(x)
      END
```

Output:

```
Inverse cosine of x is:
1.047197551197
```

AIMAG

AIMAG returns the imaginary part of an argument. It accepts a complex argument and returns a real result.

The call-by-reference entry points are MLP$RAIMAG and AIMAG, and the call-by-value entry point is MLP$VAIMAG.

The input domain is the collection of all valid complex quantities. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

No errors are generated by AIMAG. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The imaginary part of the complex argument is returned. The real part of the argument is not used.

Example of AIMAG Called From FORTRAN

Source Code:

```
      PROGRAM AIMAG_EXAMPLE
C
      EXTERNAL AIMAG
      COMPLEX xray
      xray=(3.14159, -1.0)
      PRINT *, 'The imaginary part of xray is:'
      PRINT *, AIMAG (xray)
      END
```

Output:

```
The imaginary part of xray is:
-1.
```

NOTE

AIMAG accepts a complex argument and returns a real result.

AINT

AINT returns an integer part of an argument after truncation. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RAINT and AINT, and the call-by-value entry point is MLP$VAINT.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument is added to a special floating-point zero that forces truncation. The result is returned.

Example of AINT Called From FORTRAN

Source Code:

```
      PROGRAM AINT_EXAMPLE
C
      EXTERNAL AINT
      x=1234.567890
      PRINT *, 'The integer part of x is:'
      PRINT *, AINT(x)
      END
```

Output:

```
The integer part of x is:
1234.
```

ALOG

ALOG computes the natural logarithm function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RALOG and ALOG, the call-by-value entry point is MLP$VALOG, and the vector entry point is MLP$ALOGV.

The input domain is the collection of all valid, positive real quantities. The output range is included in the set of valid real quantities whose absolute value is less than 4095*log(2).

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [1, 2) and n an integer such that x = y*2**n. Log(x) is evaluated by:

```
log(x) = log(y) + n*log(2)
```

To evaluate log(y), the interval [1, 2) is divided into 33 subintervals such that on each the abs(t) < 1/129 where t = (y − c)/(y + c). To achieve this, the subintervals are offset by 1/64. The subintervals are:

```
[1, 65/64)
[65/64, 67/64)
  :
[125/64, 127,64)
[127/64, 2)
```

Log(y) is then computed using the identity:

```
log(y) = log(c) + log((1 + t)/(1 - t))
```

and the center point c is chosen close to the midpoint of the subinterval containing y, except for the first and last subintervals, where the center points are 1 and 2, respectively. By selecting these center points, it ensures that abs(t) < 1/129.

Log ((1 + t)/(1 - t)) is approximated with a 7th degree minimax polynomial of the form:

```
2*t + c3*t**3 + c5*t**5 + c7*t**7
```

The coefficients are:

```
c3 = .6666666666667
c5 = .3999999995486
c7 = .2857343176917
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-2 shows a summary of these statistics.

Table 8-2. Relative Error of ALOG

Test	Interval From	Interval To	Maximum	Root Mean Square
ALOG(x) against ALOG(17x/16)-ALOG(17/16)	.7071E+00	.9375E+00	.1782E−13	.5463E−14
ALOG(x*x) against 2*LOG(x)	.1600E+02	.2400E+03	.7082E−14	.2035E−14
ALOG(x) against Taylor series expansion of ALOG(1 + y)	1−.1526E−04	1+.1526E−04	.1417E−13	.5197E−14

Total Error

The final calculation of log(x) is done by adding the following terms in the order below to achieve maximum precision:

```
log(x) = n*(log(2) - factor) +
         (((c7*t2 + c5)*t2 + c3)*t2)*t +
         t +
         t +
         (log(c)/2) + (factor/2)*n +
         (log(c)/2) + (factor/2)*n
```

The values of c and log(c)/2 for each subinterval are stored in a table. Factor is the nearest floating-point value with 8 bits of precision to log(2). Thus, the single precision representation of log(2) − factor is accurate to 56 bits of precision. The sum log(c) + factor*n is split into two equal parts to provide extra precision during the accumulation of the sum of terms.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/x.

Example of ALOG Called From FORTRAN

Source Code:

```
      PROGRAM ALOG_EXAMPLE
C
      x=100.0
      PRINT *, 'The natural logarithm of x is:'
      PRINT *, ALOG(x)
      END
```

Output:

```
The natural logarithm of x is:
4.605170185988
```

ALOG10

ALOG10 computes the common logarithm function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RALOG10 and ALOG10, the call-by-value entry point is MLP$VALOG10, and the vector entry point is MLP$ALOG10V.

The input domain is the collection of all valid, positive real quantities. The output range is included in the set of valid real quantities whose absolute value is less than 4095*log(2) base 10.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [1, 2) and n an integer such that x = y*2**n. Log10(x) is evaluated by:

```
log10(x) = log10(y) + n*log10(2)
```

To evaluate log10(y), the interval [1, 2) is divided into 33 subintervals such that on each the abs(t) < 1/129 where t = (y - c)/(y + c). To achieve this, the subintervals are offset by 1/64. The subintervals are:

```
[1, 65/64)
[65/64, 67/64)
   :
[125/64, 127,64)
[127/64, 2)
```

Log10(y) is then computed using the identity:

```
log10(y) = log10(c) + log10((1 + t)/(1 - t))
```

and the center point c is chosen close to the midpoint of the subinterval containing y, except for the first and last subintervals, where the center points are 1 and 2, respectively. By selecting these center points, it ensures that abs(t) < 1/129.

Log10((1 + t)/(1 - t)) is approximated with a 7th degree minimax polynomial of the form:

```
C1*t + c3*t**3 + c5 + t**5 + c7**t**7
```

The coefficients are:

```
c1 = .8685889638065
c3 = .2895296546022
c5 = .1737177925653
c7 = .1240928374639
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function ALOG10 was tested against ALOG10(11x/10) − ALOG10(11/10). Groups of 2,000 arguments were chosen randomly from the interval [.3162E+00,.9000E+00]. Statistics on relative error were observed: maximum relative error was .3011E−13, root mean square relative error was .8125E−14.

Total Error

The final calculation of log10(x) is done by adding the following terms in the order below to achieve maximum precision:

```
log10(x) = n*(log10(2) - factor) +
           (((c7*t2 + c5)*t2 + c3)*t2 + (c1 - 1))*t +
           t +
           (log10(c) + factor*n)
```

The values of c and log10(c) for each subinterval are stored in a table. Factor is the nearest floating-point value with 8 bits of precision to log10(2). Thus, the single precision representation of log10(2) − factor is accurate to 56 bits of precision. The leading coefficient of the approximation is split into 1 and (c1 − 1) to provide extra precision to the minimax polynomial approximation.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/x.

Example of ALOG10 Called From FORTRAN

Source Code:

```
      PROGRAM ALOG10_EXAMPLE
C
      x=100.0
      PRINT *, 'The common logarithm of x is:'
      PRINT *, ALOG10(x)
      END
```

Output:

```
The common logarithm of x is:
2.
```

AMOD

AMOD returns the remainder of the ratio of two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RAMOD and AMOD, and the call-by-value entry point is MLP$VAMOD.

The input domain is the collection of all valid real pairs (x,y) such that x/y is a valid real quantity, and y is not equal to 0. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

y is equal to zero.

x/y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Given the argument pair (x,y), the formula used for the AMOD computation is:

```
x - aint(x/y)*y
```

The quotient x/y is added to a special floating-point zero that forces truncation, to get n = aint(x/y); then the product of n and y is formed in double precision and subtracted from x in double precision. The most significant word of the result is returned. If the result is nonzero, it has the sign of x.

Example of AMOD Called From FORTRAN

Source Code:

```
      PROGRAM AMOD_EXAMPLE
C
      EXTERNAL AMOD
      x=750.0
      y=140.0
      PRINT *, 'The AMOD of x and y is:'
      PRINT *, AMOD(x,y)
      END
```

Output:

```
The AMOD of x and y is:
50.
```

Example of AMOD Called From Pascal

Source Code:

```
program AMOD_EXAMPLE (output);
var x, y, z : REAL;

begin
    x := 750.0;
    y := 140.0;
    z := AMOD (x, y);
    writeln ( ' The AMOD of x and y is ', z :1:1);
end.
```

Output:

```
The AMOD of x and y is 50.0
```

ANINT

ANINT returns the nearest whole number to an argument. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RANINT and ANINT, and the call-by-value entry point is MLP$VANINT.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it, and the result is added to a special floating-point zero that forces truncation. If the argument is < 0, $-.5$ is added to it, and the result is added to a special floating-point zero that forces truncation.

Example of ANINT Called From FORTRAN

Source Code:

```
      PROGRAM ANINT_EXAMPLE
C
      EXTERNAL ANINT
      x=1234.1234
      y=12.12
      PRINT *, 'The nearest whole number to x is:'
      PRINT *, ANINT(x)
      PRINT *, 'The nearest whole number to y is:'
      PRINT *, ANINT(y)
      END
```

Output:

```
The nearest whole number to x is:
1234.
The nearest whole number to y is:
12.
```

ASIN

ASIN computes the inverse sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RASIN and ASIN, the call-by-value entry point is MLP$VASIN, and the vector entry point is MLP$ASINV.

The input domain is the collection of all valid real quantities in the interval [-1.0,1.0]. The output range is included in the set of valid real quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),  x < -.5
arcos(x)  = pi - arcos(-x),  x < -.5
arcsin(x) = x + x**3*s*((w + z -j)*w + a + m/(e - x**2)),
    where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),  -.5 <= x < .5
arcsin(x) = pi/2 - arcos(x),  .5 <= x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,  .5 <= x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
       w = (x**2 - c)*z + k
       z = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =   3.173 170 078 537 13
e =   1.160 394 629 739 02
m =  50.319 055 960 798 3
c =  -2.369 588 855 612 88
i =   8.226 467 970 799 17
j = -35.629 481 597 455 5
k =  37.459 230 925 758 2
a = 349.319 357 025 144
s =     .746 926 199 335 419*10**-3
```

The approximation of arcsin(-.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

```
a.  If ACOS entry, go to step g.

b.  If |x| ≥ .5, go to step h.

c.  n = 0 (Loop counter).
    q = x
    y = x**2
    u = 0, if ASIN entry.
    u = pi/2, if ACOS entry.

d.  z = (y + r)*y + i
    w = (y - c )*z + k
    p = q + s*q*y*((w + z - j)*w + a + m/(e - y))
    p = u - p
    Y1 = p/2**n

e.  If ASIN entry, go to step k.

f.  If x is in (-.5,1.0), return.
    XF = 2*u - (Y1)
    Return.

g.  If |x| < .5, go to step c.

h.  If x = 1.o or -1.0, go to step 1.
    If x is invalid, go to step m.
        n = 0 (Loop counter).
        y = 1.0 - |x|, and normalize y.

i.  h = 4*y - 2*y**2
    n = n + 1.0
    If 2*y ≤ 2 - sqrt(3) = .267949192431, y = h, and go to step i.

j.  q = 1.0 - h, and normalize q.
    y = q**2
    u = pi/2
    Go to step d.

k.  Y1 = u - (Y1), and normalize Y1.
    Affix sign of x to Y1 = XF.
    Return.

l.  XF = pi/2, if x = 1.0.
    XF = -pi/2, if x = -1.0.
    If ASIN entry, return.
    XF = 0, if x = 1.0.
    XF = pi, if x = -1.0.
    Return.

m.  Return.
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of relative error of the ASIN approximation over (–.5,.5) is 1.996*E–15.

The function ASIN was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-3 shows a summary of these statistics.

Table 8-3. Relative Error of ASIN

Interval From	Interval To	Maximum	Root Mean Square
–.1250E+00	.1250E+00	.7101E–14	.2763E–14
.7500E+00	.1000E+01	.8378E–14	.3462E–14

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/(1.0 – x**2)**.5.

Example of ASIN Called From FORTRAN

Source Code:

```
      PROGRAM ASIN_EXAMPLE
      x=0.5
      PRINT *, 'The inverse sine of x is:'
      PRINT *, ASIN(x)
      END
```

Output:

```
The inverse sine of x is:
.5235987755983
```

ATAN

ATAN computes the inverse tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RATAN and ATAN, the call-by-value entry point is MLP$VATAN, and the vector entry point is MLP$ATANV.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid real quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument x is transformed into an argument y in the interval [0,1/16] by the range reduction formulas:

```
arctan(u) = -arctan(-u), if u < 0
arctan(u) =  pi/4 + (pi/4 - arctan(1/u)), if u >= 1.0
arctan(u) =  arctan(k/16) + arctan((u - k/16)/(1.0 +  u*k/16)),
             if 0 <= u < 1.0, and k is the greatest integer not
             exceeding 16*u.
```

Finally, arctan(y) (for y in [0,1/16]) is computed by the polynomial approximation:

```
arctan(y) = y + a(1)*y**3 + a(2)*y**5 + a(3)*y**7 + a(4)*y**9
```

where:

```
a(1) = -.333 333 333 333 128 45
a(2) =  .199 999 995 801 446 4
a(3) = -.142 854 130 508 745 0
a(4) =  .110 228 161 612 614 9
```

The coefficients of this polynomial are those of the minimax polynomial approximation of degree 3 to the function f over (0,1/4), where f(u**2 = (arctan(u) - u)/u**3.[1]

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

1. Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-4 shows a summary of these statistics.

Table 8-4. Relative Error of ATAN

Test	Interval From	Interval To	Maximum	Root Mean Square
ATAN(x) against truncated Taylor series	−.6250E−01	.6250E−01	.7102E−14	.3647E−14
2*ATAN(x) against ATAN(2x/(1 − x*x))	.2679E+00	.4142E+00	.1355E−13	.4023E−14
	.4142E+00	.1000E+01	.1763E−13	.5931E−14
ATAN(x) against ATAN(1/16) + ATAN((x − 1/16)/(1 + x/16))	.6250E−01	.2679E+00	.7117E−14	.2605E−14

Effect of Argument Error

If a small error e occurs in the argument, the error in the result y is given approximately by e/(1+y**2).

Example of ATAN Called From FORTRAN

Source Code:

```
      PROGRAM ATAN_EXAMPLE
C
      x=0.5
      PRINT *, 'The inverse tangent of x is:'
      PRINT *, ATAN(x)
      END
```

Output:

```
The inverse tangent of x is:
.4636476090008
```

ATANH

ATANH computes the inverse hyperbolic tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RATANH and ATANH, the call-by-value entry point is MLP$VATANH, and the vector entry point is MLP$ATANHV.

The input domain is the collection of all valid real quantities whose absolute value is less than 1.0. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument range can be reduced to the interval [0,1.0] by the identity atanh(−x) = −atanh(x). The expression atanh(x) = .5*ln((1.0 + x)/(1.0 − x)) is formed by using the definition tanh(x) = (e**x − e**−x)/(e**x + e**−x).

The argument range of the log can be reduced to the interval [.75,1.5] by using the property ln(a*b) = ln(a) + ln(b), and extracting the appropriate multiple of ln(2):

```
atanh(x) = .5*n*ln(2) + .5*ln(2**-(n)*(1.0 + x)/(1.0 - x))
```

The argument range is reduced to the interval [−.2,.2] by writing the argument of log in the form (1.0 + y)/(1.0 − y), and substituting atanh(y):

$$\text{atanh}(x) = .5*n*\ln(2) + \text{atanh}\left[\frac{2**-n*(1.0 + x) - (1.0 - x)}{2**-n*(1.0 + x) + (1.0 - x)}\right]$$

The value of n such that 2**−n*(1.0 + x)/(1.0 − x) is in the interval [.75,1.5] is the same as the value of n such that 2**−n*(1.0 + x)/(.75*(1.0 − x)) is in the interval [1.0,2.0]. If .75*(1.0 − x) is written as a*2**m, where a is in interval [1.0,2.0], then 2**(−n − m)*(1.0 + x)/a must be in interval [1.0,2.0]. If (1.0 + x) ≥ a, then −n − m = 0 and n = −m. If (1.0 + x) < a, then −n − m = 1.0 and n = 1.0 − m.

The function atanh(z) in the interval [−.2,.2] is approximated by z + z**3*p/q, where p and q are 4th order even polynomials. For atanh(z), the coefficients of p and q were derived from the (7th order odd)/(4th order even) minimax (relative error) rational form in the interval [−.2,.2].

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 1.0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

For abs(x) < .2, n equals zero, and the expected bound of the error is 4.8E−15.

For abs(x) ≥ .5, the term n*(ln(2)/2) dominates. This term is computed as n*(ln(2)/2 − .125) − n*.125 − n*.125 because the rounding error in representing ln(2)/2 is large; the above form makes the rounding error relatively small. Since n*.125 is exact and the dominating form, the two additions in (other)n*.125 + n*.125 dominate the error, and the expected relative error is 8.3E−15 in this region.

For .2 ≤ abs(x) < .5, n equals one, and the term z = (.5*(1.0 + x) − (1.0 − x))/(.5*(1.0 + x) + (1.0 − x)) may be relatively large. For abs(x) < 0.25, the subtraction 1.0 − x = .5 − x + .5 loses two bits of the original argument. Also, z is negative in this range, and some cancellation occurs in the final combination of terms, costing about one unit in the last place (ulp). The expected upper bound in the region .2 < abs(x) < 0.25 is 19.4E−15.

A group of 10,000 arguments was chosen randomly from the interval [−1.0,1.0]. The maximum relative error of these arguments was found to be .3304E−13.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is 1.0/(1.0 − x**2), and that of relative error is x/((1.0 − x**2)*atanh(x)). This increases from 1 at 0 and becomes arbitrarily large near 1.0.

Example of ATANH Called From FORTRAN

Source Code:

```
      PROGRAM ATANH_EXAMPLE
C
      x=0.5
      PRINT *, 'The inverse hyperbolic tangent of x is:'
      PRINT *, ATANH(x)
      END
```

Output:

```
 The inverse hyperbolic tangent of x is:
 .5493061443341
```

ATAN2

ATAN2 computes the inverse tangent function of the ratio of two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RATAN2 and ATAN2, and the call-by-value entry point is MLP$VATAN2.

The ATAN2 vector math function is divided into three routines having three separate entry points defined as follows:

```
ATAN2(scalar,vector) = MLP$ATAN2SV
ATAN2(vector,scalar) = MLP$ATAN2VS
ATAN2(vector,vector) = MLP$ATAN2VV
```

The input domain is the collection of all valid real pairs (x,y) such that both quantities are not equal to zero. The output range is included in the set of valid real quantities greater than −pi and less than or equal to pi.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

x/y is infinite (positive or negative) and y is not equal to zero.

x is not equal to zero and y is infinite (positive or negative).

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite and y does not equal zero, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The function ATAN2(y,x) is defined to be the angle, in the interval [−pi,pi], subtended at the origin by the point (x,y) and the first coordinate axis.

The argument (y,x) is reduced to the first quadrant by the range reductions:

```
atan2(y,x) = -atan2(-y,x), y < 0
atan2(y,x) = pi - atan2(y,-x), x < 0, y > 0
```

The argument (y,x) is then reduced to the sector:

```
(u,v): u ≥ 0,  v < u, and  v ≥ 0
```

by the range reduction:

```
atan2(y,x) = pi/2 - atan2(x,y), x ≥ 0 or y ≥ 0
```

The routine calls ATAN to evaluate atan2(y,x) as arctan(y/x).[2]

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

x/y is infinite (positive or negative) and y is not equal to zero.

x is not equal to zero and y is infinite (positive or negative).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

2. Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-4 shows a summary of these statistics.

Effect of Argument Error

If small errors e(x) and e(y) occur in x and y, respectively, the error in the result is given approximately by (y*e(x) - x*e(y))/(x**2 + y**2).

Example of ATAN2 Called From FORTRAN

Source Code:

```
      PROGRAM ATAN2_EXAMPLE
C
      x=0.5
      y=0.6
      PRINT *, 'The inverse tangent of the ratio of x,y is:'
      PRINT *, ATAN2(x,y)
      END
```

Output:

```
The inverse tangent of the ratio of x,y is:
.6947382761967
```

CABS

CABS computes the absolute value of an argument. It accepts a complex argument and returns a real result.

The call-by-reference entry points are MLP$RCABS and CABS, and the call-by-value entry point is MLP$VCABS.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 is a valid real quantity. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is positive infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The algorithm used is:

```
a.  u = max(|x|,|y|).
    v = min(|x|,|y|).

b.  If u is zero, return zero to the calling program.

c.  r   = v/u
    w   = 1.0 + r**2

    where t = w**.5 = (1.0 + r**2)**.5 is computed inline using the same
    algorithm as used in SQRT.

d.  Return u*t to the calling program.
```

Formulas used are:

```
|x + i*y| = sqrt(x + i*y)
          = max(|x|,|y|)*(1 + r**2)**.5
          = u*t
```

where r = min(|x|,|y|)/max(|x|,|y|)

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval of complex numbers ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be .1401E−13.

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result u is given by e(u) = (x*e(x) + y*e(y))/u.

Example of CABS Called From FORTRAN

Source Code:

```
      PROGRAM CABS_EXAMPLE
C
      COMPLEX xi
      xi=(-40.0, -1)
      PRINT *, 'The CABS of xi is:'
      PRINT *, CABS(xi)
      END
```

Output:

```
The CABS of xi is:
40.01249804748
```

NOTE

CABS accepts a complex argument and returns a real result.

CCOS

CCOS computes the complex cosine function. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCCOS and CCOS, the call-by-value entry point is MLP$VCCOS, and the vector entry point is MLP$CCOSV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y; |x| is less than 2**47 and |y| is less than 4095*log(2). The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The imaginary part is greater than or equal to 4095*log(2).

The imaginary part is less than or equal to −4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

```
cos(x + i*y) = cos(x)*cosh(y) - i*sin(x)*sinh(y)
```

The routine evaluates COSSIN inline to simultaneously compute the sine and cosine of the real part of the argument. The routine evaluates HYPERB inline to simultaneously compute the hyperbolic sine and hyperbolic cosine of the imaginary part of the argument. See the descriptions of routines COSSIN and HYPERB in chapter 9, Auxiliary Routines, for detailed information.

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The imaginary part is greater than or equal to 4095*log(2).

The imaginary part is less than or equal to −4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

See the descriptions of HYPERB and COSSIN in chapter 9, Auxiliary Routines, for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.276E−13 + 1.241E−13*exp(abs(y)).

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be .7665E−13.

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result is given approximately by −sin(z)*e(z).

Example of CCOS Called From FORTRAN

Source Code:

```
      PROGRAM CCOS_EXAMPLE
C
      COMPLEX xi
      xi=(-40.0, -1)
      PRINT *, 'The complex cosine of xi is:'
      PRINT *,CCOS(xi)
      END
```

Output:

```
 The complex cosine of xi is:
 (-1.029139207557,-.875657875595)
```

CEXP

CEXP computes the complex exponential function. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCEXP and CEXP, the call-by-value entry point is MLP$VCEXP, and the vector entry point is MLP$CEXPV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y; x is less than 4095*log(2) and x is greater than −4097*log(2), and |y| is less than 2**47. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The real part is greater than or equal to 4095*log(2) or less than or equal to −4097*log(2).

The absolute value of the imaginary part is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

```
exp(x + i*y) = exp(x)*cos(y) + i*exp(x)*sin(y)
```

The routine evaluates COSSIN inline to compute cos(y) and sin(y), and calls EXP to compute exp(x).

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The real part is greater than or equal to 4095*log(2) or less than or equal to −4097*log(2).

The absolute value of the imaginary part is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

See the descriptions of EXP in this chapter and COSSIN in chapter 9, Auxiliary Routines, for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.378E−13 + 1.378E−13*exp(abs(x)). If the real part of the argument is large, the error in the routine will be significant.

The function CEXP was tested. A group of 10,000 arguments was chosen randomly from given intervals. Statistics on maximum relative error were observed. Table 8−5 shows a summary of these statistics.

Table 8-5. Relative Error of CEXP

Interval	Maximum
([−1.0,1.0],[−1.0,1.0])	.5462E−13
([1.0,.6700E+03],[1.0,.11E+15])	.9182E−13

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result w is given approximately by w*e(z).

Example of CEXP Called From FORTRAN

Source Code:

```
      PROGRAM CEXP_EXAMPLE
C
      COMPLEX xi
      xi=(-4.0, -1)
      PRINT *, 'The CEXP of xi is:'
      PRINT *, CEXP(xi)
      END
```

Output:

```
The CEXP of xi is:
(.009895981925031,-.01541207869309)
```

CLOG

CLOG computes the complex natural logarithm function. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCLOG and CLOG, the call-by-value entry point is MLP$VCLOG, and the vector entry point is MLP$CLOGV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 is a valid, positive real quantity. The output range is included in the set of valid complex quantities z, such that the real part of z is a valid real quantity, and the imaginary part is greater than −pi and less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

Both the real part and the imaginary part are zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
log(z) = log(|z|) + i*arg(z)
```

where |z| is the modulus of z. The routine calls CABS to evaluate the absolute value of z and calls ALOG to compute the logarithm. Then the routine calls ATAN2 to evaluate the function arg(z). When z is nonzero, and in-range, arg(z) is in the interval [−pi,pi].

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

Both the real part and the imaginary part are zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be .4346E−12.

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result is given approximately by e(z)/z.

Example of CLOG Called From FORTRAN

Source Code:

```
      PROGRAM CLOG_EXAMPLE
C
      COMPLEX xi
      xi=(-4.0, -1)
      PRINT *, 'The CLOG of xi is:'
      PRINT *, CLOG(xi)
      END
```

Output:

```
The CLOG of xi is:
(1.416606672028,-2.896613990463)
```

NOTE

One of the real or imaginary parts for CLOG must be nonzero.

CONJG

CONJG returns the conjugate of an argument. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCONJG and CONJG, and the call-by-value entry point is MLP$VCONJG.

The input domain is the collection of all valid complex quantities. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

No errors are generated by CONJG. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with its imaginary part negated.

Example of CONJG Called From FORTRAN

Source Code:

```
      PROGRAM CONJG_EXAMPLE
C
      EXTERNAL CONJG
      COMPLEX xi
      xi=(-40000.0, -1)
      PRINT *, 'The conjugate of xi is:'
      PRINT *, CONJG(xi)
      END
```

Output:

```
The conjugate of xi is:
(-40000.,1.)
```

COS

COS computes the cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOS and COS, the call-by-value entry point is MLP$VCOS, and the vector entry point is MLP$COSV.

The input domain is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, then COS(x) or SIN(x) is calculated by using the periodic properties of the cosine and sine functions to reduce the task to finding a cosine or sine of an equivalent angle y within [−pi/4, pi/4] as follows:

```
If N + K is even
then
     Z = sin(y)
else
     Z = cos(y)
If MOD(N + K, 4) is 0 or 1 (that is, the second last bit of N + K is even)
then
     S = 0
else
     S = mask(1)
```

where K is 0, 1, or 2 according to whether the SIN of a positive angle, the COS of any angle, or the SIN of a negative angle is to be calculated. N is the nearest integer to 2/pi*x, and y is the nearest single precision floating-point number to x − n*pi/2. The argument x is the absolute value of the angle. The desired SIN or COS is the exclusive or of S and Z.

Once the angle has been reduced to the range [−pi/4, pi/4], the following approximations are used to calculate either the cosine or the sine of the angle, providing 48 bits of precision.

If the cosine of the angle is required, the approximation used is

```
cosine(y) = 1   y*y*P(y*y)
```

where y is the angle and P(w) is the quintic polynomial:

```
P(w) = P0 + P1*w  + P2*w**2 + P3 + w**3 + P4*w**4 + P5*w**5
```

such that P(y*y) is a minimax polynomial approximation to the function (1 – cos(y))/y**2.

The coefficients are:

```
P5 = -2.07006230562462946 2E-9
P4 =  2.755636997406588778E-7
P3 = -2.480158521206426671E-5
P2 =  1.388888888727866775E-3
P1 = -4.166666666666468116E-2
P0 =  5.000000000000000000E-1
```

If the sine of the angle is required, the approximation used is

```
sine(y) = y - y*y*y*Q(y*y)
```

where y is the angle and Q(w) is the quintic polynomial:

```
Q(w) = Q0 + Q1*w  + Q2*w**2 + Q3*w**3 + Q4*w**4 + Q5*w**5
```

such that Q(y*y) is a minimax polynomial approximation to the function (y – sin(y))/y**3.

The coefficients are:

```
Q5 = -1.591814257033005283E-10
Q4 =  2.505113204973767698E-8
Q3 = -2.755731610365754733E-6
Q2 =  1.984126983676100911E-4
Q1 = -8.333333333330950363E-3
Q0 =  1.666666666666666463E-1
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function COS was tested against 4*COS(x/3)**3 - 3*COS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199E+02,.2356E+02]. Statistics on relative error were observed: maximum relative error was .1404E−13, and root mean square relative error was .3245E−14.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e*cos(x) for sin(x) and −e*sin(x) for cos(x).

Example of COS Called From FORTRAN

Source Code:

```
      PROGRAM COS_EXAMPLE
C
      x=0.5
      PRINT *, 'The cosine of x is:'
      PRINT *, COS(x)
      END
```

Output:

```
The cosine of x is:
.8775825618904
```

Example of COS Called From Pascal

Source Code:

```
program COS_EXAMPLE (output);
var x, y : REAL;

begin
    x := 0.5;
    y := COS (x);
    writeln (' The cosine of x is ', y :1:13);
end.
```

Output:

```
The cosine of x is 0.8775825618904
```

COSD

COSD computes the cosine function for an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOSD and COSD, the call-by-value entry point is MLP$VCOSD, and the vector entry point is MLP$COSDV.

The input domain is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [−45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x ± 360 degrees) =  sin(x)
sin(x ± 180 degrees) = -sin(x)
sin(x +  90 degrees) =  cos(x)
sin(x -  90 degrees) = -cos(x)
cos(x ± 360 degrees) =  cos(x)
cos(x ± 180 degrees) = -cos(x)
cos(x +  90 degrees) = -sin(x)
cos(x -  90 degrees) =  sin(x)
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The reduction to (−45,+45) is exact; the constant pi/180 has relative error 1.37E−15, and multiplication by this constant has a relative error 5.33E−15, and a total error of 6.7E−15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is, at most, 5.26E−15 plus the maximum error in SINCOS over (−pi/4,+pi/4).

A group of 10,000 arguments was chosen at random from the interval [0,360]. The maximum relative error of these arguments was found to be .7105E−14 for COSD and .1403E−13 for SIND.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (−45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

Example of COSD Called From FORTRAN

Source Code:

```
      PROGRAM COSD_EXAMPLE
C
      x=180.0
      PRINT *, 'The COSD of x is:'
      PRINT *, COSD(x)
      END
```

Output:

```
The COSD of x is:
-1.
```

COSH

COSH computes the hyperbolic cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOSH and COSH, the call-by-value entry point is MLP$VCOSH, and the vector entry point is MLP$COSHV.

The input domain is the collection of all valid real quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid real quantities greater than or equal to 1.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formula used to compute cosh(x) is:

```
cosh(x) = (exp(x) + exp(-x))/2
```

The routine calls EXP to compute exp(x) and computes 1.0/exp(x) to obtain exp(−x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-6 shows a summary of these statistics.

Table 8-6. Relative Error of COSH

Test	Interval From	Interval To	Maximum	Root Mean Square
COSH(x) against Taylor series expansion of COSH(x)	0.0000E+00	.5000E+00	.1382E−13	.6875E−14
COSH(x) against c*(COSH(x + 1) + COSH(x − 1))	.3000E+01	.2838E+04	.2296E−13	.8260E−14

Effect of Argument Error

If a small error e occurs in the argument x, the resulting error in cosh(x) is given approximately by sinh(x)*e.

Example of COSH Called From FORTRAN

Source Code:

```
      PROGRAM COSH_EXAMPLE
C
      x=180.0
      PRINT *, 'The COSH of x is:'
      PRINT *, COSH(x)
      END
```

Output:

```
The COSH of x is:
7.446921003909E+77
```

COTAN

COTAN computes the trigonometric circular cotangent of an argument in radians. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOTAN and COTAN, the call-by-value entry point is MLP$VCOTAN, and the vector entry point is MLP$COTANV.

The input domain is the collection of all valid real quantities whose absolute value is greater than 0 and less than 2**47. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is 0.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The evaluation is reduced to the interval [-.5,.5] by using the identities:

```
1.  cotan(x) = cotan(x + k*pi/2), if k is even

2.  cotan(x) = -1.0/cotan(x + pi/2)
```

in the form:

```
3.  cotan(x)=1/tan(x)=1/tan((pi/2)*(x*2/pi + k)), if k is even

4.  cotan(x)=1/tan(x)=tan((pi/2)*(x*2/pi + 1.0))/-1.0
```

In effect, the original algorithm for TAN(x) is used to find COTAN(x). The result for COTAN(x) is the reciprocal of TAN(x).

An approximation of tan(pi/2*y) is used. The argument is reduced to the interval [-.5,.5] by subtracting a multiple of pi/2 from x in double precision.

The rational form is used to compute the tangent of the reduced value. The function tan((pi/2)*y) is approximated with a rational form (7th order odd)/(6th order even), which has minimax relative error in the interval [-.5,.5]. The rational form is normalized to make the last numerator coefficient 1 + e, where e is chosen to minimize rounding error in the leading coefficients.

Identity 4 is used if the integer subtracted is odd. The result is negated and inverted by dividing -P/Q instead of Q/P.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is 0.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function COTAN was tested against (COTAN(x/2)**2−1)/(2*COTAN(x/2)). Groups of 2,000 arguments were chosen randomly from the interval (.1885E+02, .1963E+02). Statistics on relative error were observed: maximum relative error was .2297E−13, and root mean square relative error was .7847E−14.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is sec(x)**2, and that of relative error is x/(sin(x)*cos(x)), which is at least 2x and can be arbitrarily large near a multiple of pi/2.

Example of COTAN Called From FORTRAN

Source Code:

```
      PROGRAM COTAN_EXAMPLE
C
      x=180.0
      PRINT *, 'The COTAN of x is:'
      PRINT *, COTAN(x)
      END
```

Output:

```
The COTAN of x is:
.746998814414
```

CSIN

CSIN computes the complex sine function. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCSIN and CSIN, the call-by-value entry point is MLP$VCSIN, and the vector entry point is MLP$CSINV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y; |x | is less than 2**47, and |y | is less than 4095*log(2). The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The absolute value of the imaginary part is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

```
sin(x + i*y) = sin(x)*cosh(y) + i*cos(x)*sinh(y)
```

The routine evaluates COSSIN inline to simultaneously compute sine and cosine, and evaluates HYPERB inline to simultaneously compute hyperbolic sine and hyperbolic cosine. See the descriptions of routines COSSIN and HYPERB in chapter 9, Auxiliary Routines, for detailed information.

Vector Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The absolute value of the imaginary part is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.276E−13 + 1.297E−13*exp(abs(y)). See the description of HYPERB and COSSIN for details in chapter 9, Auxiliary Routines.

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result is given approximately by cos(z)*e(z).

Example of CSIN Called From FORTRAN

Source Code:

```
      PROGRAM CSIN_EXAMPLE
C
      COMPLEX xi
      xi=(-40.0, -1)
      PRINT *, 'The CSIN of xi is:'
      PRINT *, CSIN(xi)
      END
```

Output:

```
 The CSIN of xi is:
 (-1.149769688682,.7837864061402)
```

CSQRT

CSQRT computes the complex square root function that maps to the right half of the complex plane. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCSQRT and CSQRT, the call-by-value entry point is MLP$VCSQRT, and the vector entry point is MLP$CSQRTV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 + |x| is a valid real quantity. If the argument is zero, zero is returned. The output range is included in the set of valid complex quantities z such that the real part of z is nonnegative and the imaginary part of z is a valid complex quantity.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is positive infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

For this computation, values returned by the routine will lie in the right half of the complex plane.

Call-By-Value Routine

Let x + i*y be the argument. The formulas used for computation are:

```
u = (.5*(|x| + |(x,y)|))**.5
v = .5*(y/u)
```

If x is nonnegative, then csqrt(x,y) = u + i*v. If x is negative, then csqrt(x,y) = sign(y)*(v + i*u).

The result of this routine always lies in the first or fourth quadrant of the complex plane. The routine takes complex quantities lying on the axis of the negative reals, to the axis of the positive imaginaries.

Vector Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function CSQRT was tested. A group of 10,000 arguments was chosen randomly from given intervals. Statistics on maximum relative error were observed. Table 8-7 shows a summary of these statistics.

Table 8-7. Relative Error of CSQRT

Interval	Maximum
([0,0],[100,100])	.1600E−13
([0,0],[1.0E+100,1.0E+100])	.1499E−13

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result w = u + i*v is given approximately by e(z)/(2*w**0.5) = (e(x) + i*e(y))/2(u + i*v)**0.5.

Example of CSQRT Called From FORTRAN

Source Code:

```
      PROGRAM CSQRT_EXAMPLE
C
      COMPLEX xi
      xi=(-40.0, -1)
      PRINT *, 'The CSQRT of xi is:'
      PRINT *, CSQRT(xi)
      END
```

Output:

```
The CSQRT of xi is:
(.07905076686887,-6.325049329748)
```

DABS

DABS computes the absolute value of an argument. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDABS and DABS, and the call-by-value entry point is MLP$VDABS.

The input domain is the collection of all valid double precision quantities. The output range is included in the set of valid, nonnegative double precision quantities.

Call-By-Reference Routine

No errors are generated in DABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with the sign bits of both its upper and lower words forced positive.

Example of DABS Called From FORTRAN

Source Code:

```
      PROGRAM DABS_EXAMPLE
C
      EXTERNAL DABS
      DOUBLE PRECISION x
      x=-1000.1234d0
      PRINT *, 'The DABS of x is:'
      PRINT *, DABS(x)
      END
```

Output:

```
The DABS of x is:
1000.1234
```

DACOS

DACOS computes the inverse cosine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDACOS and DACOS, and the call-by-value entry point is MLP$VDACOS.

The input domain is the collection of all valid double precision quantities in the interval [−1.0,1.0]. The output range is included in the set of valid, nonnegative double precision quantities less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The following identities are used to move the interval of approximation to [0,sqrt(.5)]:

```
arcsin(-x)  = -arcsin(x)
arccos(x)   = pi/2-arcsin(x)
arcsin(x)   = arccos(sqrt(1.0 - x**2)), if x ≥ 0
arccos(x)   = arcsin(sqrt(1.0 - x**2)), if x ≥ 0
```

The reduced value is called y. If y <= .09375, no further reduction is performed. If not, the closest entry to y in a table of values (z, arcsin(z), sqrt(1.0 - x**2), z = .14, .39, .52, .64) is found, and the following formula used is:

```
arcsin(x) = arcsin(z) + arcsin(w)
```

where w = x(sqrt(1.0 - z**2) - z*sqrt(1.0 - x**2). The value of w is in (-.0792, .0848).

The arcsin of the reduced argument is then found using a 15th order odd polynomial with quotient:

```
x + x**3(c(3) + x**2(c(5) + x**2(c(7) + x**2(c(11) + x**2(c(13) +
x**2(c(15) + a/(b - x**2))))))
```

where all constants and arithmetic operations before c(11) are double precision and the rest are single precision. The addition of c(11) has the form single + single = double. The polynomial is derived from a minimax rational form (denominator is (b - x**2)) for which the critical points have been modified slightly to make c(11) fit in one word.

To this value, arcsin(z) is added from a table if the last reduction above was done and the sum is conditionally negated. Then 0, -pi/2, + pi/2, or pi is added to complete the unfolding.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The region of worst error is (.9895,.9966). In this region, the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs.

The function DACOS was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-8 shows a summary of these statistics.

Table 8-8. Relative Error of DACOS

Interval From	Interval To	Maximum	Root Mean Square
−.1250D+00	.1250D+00	.2794D−27	.2343D−27
−.1000D+01	−.7500D+00	.3339D−27	.2853D−27
.7500D+00	.1000D+01	.7573D−28	.2257D−28

Effect of Argument Error

If a small error e occurs in the argument x, the resulting error in DACOS is approximately -e/(1.0 - x**2)**.5. The amplification of the relative error is approximately x/(f(x)*(1.0 - x**2)**.5), where f(x) is DACOS. The error is attenuated for x > -.44 but can become serious near -1.0. If the argument is generated as 1.0 - y or y - 1.0, then the following identities can be used to get the full significance of y:

```
asin(x)  = acos(sqrt(1.0 - x**2))
acos(x)  = asin(sqrt(1.0 - x**2))
asin(-x) = -asin(x)
acos(-x) = pi + asin(x)
```

Example of DACOS Called From FORTRAN

Source Code:

```
      PROGRAM DACOS_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DACOS of x is:'
      PRINT *, DACOS(x)
      END
```

Output:

```
The DACOS of x is:
1.0471975511965977461542144 6
```

DASIN

DASIN computes the inverse sine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDASIN and DASIN, the call-by-value entry point is MLP$VDASIN, and the vector entry point is MLP$DASINV.

The input domain is the collection of all valid double precision quantities in the interval [−1.0,1.0]. The output range is included in the set of valid double precision quantities in the interval [−pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The following identities are used to move the interval of approximation to [0,sqrt(.5)]:

```
arcsin(-x) = -arcsin(x)
arccos(x)  =  pi/2-arcsin(x)
arcsin(x)  =  arccos(sqrt(1.0 - x**2)), if x ≥ 0
arccos(x)  =  arcsin(sqrt(1.0 - x**2)), if x ≥ 0
```

The reduced value is called y. If y <= .09375, no further reduction is performed. If not, the closest entry to y in a table of values (z, arcsin(z), sqrt(1.0 - x**2), z = .14, .39, .52, .64) is found, and the formula used is:

```
arcsin(x) = arcsin(z) + arcsin(w)
```

where w = x(sqrt(1.0 - z**2) - z*sqrt(1.0 - x**2). The value of w is in (-.0792, .0848).

The arcsin of the reduced argument is then found using a 15th order odd polynomial with quotient:

```
x + x**3(c(3) + x**2(c(5) + x**2(c(7) + x**2(c(11) + x**2(c(13) +
x**2(c(15) + a/(b   x**2)))))))
```

where all constants and arithmetic operations before c(11) are double precision and the rest are single precision. The addition of c(11) has the form single + single = double. The polynomial is derived from a minimax rational form (denominator is (b - x**2)) for which the critical points have been perturbed slightly to make c(11) fit in one word.

To this value, arcsin(z) is added from a table if the last reduction above was done and the sum is conditionally negated. Then 0, -pi/2, + pi/2, or pi is added to complete the unfolding.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The region of worst error is (.09375,.1446). In this region, the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs, the worst case being .1451−.0629. For DASIN, the polynomial range was extended to cover the region (.0821,.09375), where the worst error occurs.

The function DASIN was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-9 shows a summary of these statistics.

Table 8-9. Relative Error of DASIN

Interval From	Interval To	Maximum	Root Mean Square
−.1250D+00	.1250D+00	.1017D−27	.2246D−28
.7500D+00	.1000D+01	.4761D−27	.3575D−27

Effect of Argument Error

If a small error e occurs in the argument x, the resulting errors in DASIN are approximately e/(1 - x**2)**.5. The amplification of the relative error is approximately x/(f(x)*(1 - x**2)**.5), where f(x) is DASIN. The error is attenuated for abs(x) < .75 but can become serious near −1.0 or +1.0. If the argument is generated as 1 - y or y - 1, then the following identities can be used to get the full significance of y:

```
asin(x)  = acos(sqrt(1.0 - x**2))
acos(x)  = asin(sqrt(1.0 - x**2))
asin(-x) = -asin(x)
acos(-x) = pi + asin(x)
```

Example of DASIN Called From FORTRAN

Source Code:

```
      PROGRAM DASIN_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DASIN of x is:'
      PRINT *, DASIN(x)
      END
```

Output:

```
The DASIN of x is:
.52359877559829887307710723l
```

DATAN

DATAN computes the inverse tangent function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDATAN and DATAN, the call-by-value entry point is MLP$VDATAN, and the vector entry point is MLP$DATANV.

The input domain is the collection of all valid double precision quantities. The output range is included in the set of valid double precision quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Register pair (X4,X5) holds the absolute value of the argument.

B4 = (X9) = sign mask for the argument. (B4 holds a mask for the result's sign.)

If $|x| < 1.0$, then:

```
B3 = (XA) = 0.
B7 = (XB) = 0.  (B7 will hold the closest multiple of pi/2 to the absolute
value of the result.)
Branch to DATCOM at label DTN to complete processing.
```

If $|x| \geq 1.0$, then:

```
B3 = (XA) = 1 in high order bit.
B7 = (XB) = 1.0.
Branch to DATCOM at label DATCOM to complete processing.
```

At labels DATCOM and DTN:

```
(X9) = B4 = mask MS = sign of final result.
(XA) = B3 = mask MI.
(XB) = B7 = closest multiple of pi/2 to the absolute value of the result.
```

At label DATCOM:

```
Register pair (X7,X8) = DU.
Register pair (X4,X5) = DV.
```

At label DTN:

```
Register pair (X7,X8) = DU.
```

Label ATNU is the start of an 18-word table containing atan(n/8) ($0 \leq n \leq 8$) in double precision. Label DATCOM corresponds to step a, and label DTN corresponds to step b.

Constants used in the algorithm are:

```
d3  =  -.333 333 333 333 333 333 333 333 285 915
d5  =   .199 999 999 999 999 999 999 673 046 526
d7  =  -.142 857 142 857 142 856 280 180 055 289
d9  =   .111 111 111 111 109 972 932 035 508 119
c11 =  -.090 909 090 908 247 503
c13 =   .001 351 201 845 778 152
a   =  -.085 666 743 757 593 089
b   = -1.133 579 709 202 919 6
```

where d3, d5, d7, and d9 are double precision constants, and c11, c13, a, and b are real constants. Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. Boolean operations have B subscripts.

The algorithm used is:

```
a.  DQ = DU/DV computed in double precision.

b.  (DQ = DA-DU at DTN)  (Note that 0 ≤ DQ ≤ 1.0.)

c.  n = nearest multiple of 1/8 to DQ.

d.  If n = 0, go to step f.

e.  DA = (DQ - n/8)/(1.0 + n/8*DA), computed in double precision.

f.  Z  = 0
    DC = 0
    If (DA)(u) = 0, go to step i.

g.  XX = DA(u)*DA(u)
    DC = XX*(d)(d3 +(d) XX*(d)(d5 +(d) XX*(d) (d7 +(d) XX*(d)(d9 +(d)
         XX*(d)(d11 +(d) XX*(u)(c13 +(u) a/(b -(u) XX)))))))

h.  w = DA +(d) DC*DA

i.  DB = 0
    If (XB)not= 0  DB = ATN(9)*2*(XB)

j.  BBAR = (B7*pi/2) - (B)B3 (upper and lower)

k.  CBAR = BBAR + (D)ATN(n/8).  ATN(n/8) is obtained as a double precision
    quantity from a table of precomputed values.

l.  Result = (CBAR + (D) w) - (B) (B3 - (B)B4).
```

At the end of processing, register pair (XE,XF) contains the DATAN result.

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of relative error in the algorithm is 1.622E−29.

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-10 shows a summary of these statistics.

Table 8-10. Relative Error of DATAN

Test	Interval From	Interval To	Maximum	Root Mean Square
DATAN(x) against truncated Taylor series	−.6250D−01	.6250D−01	.2556D−28	.1343D−28
2*DATAN(x) against DATAN(2x/(1 − x*x))	.2697D+00	.4142D+00	.4821D−28	.2027D−28
	.4142D+00	.1000D+01	.5992D−28	.2449D−28
DATAN(x) against DATAN(1/16) + DATAN((x − 1/16)/(1 + x/16))	.6250D−01	.2679D+00	.3388D−28	.1557D−28

Total Error

Most of the errors can be traced back to errors in double precision addition.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given by e/(1.0 + x**2).

Example of DATAN Called From FORTRAN

Source Code:

```
      PROGRAM DATAN_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DATAN of x is:'
      PRINT *, DATAN(x)
      END
```

Output:

```
The DATAN of x is:
.46364760900080611621425623I
```

DATAN2

DATAN2 computes the inverse tangent function of the ratio of two arguments. It accepts two double precision arguments and returns a double precision result.

The call-by-reference entry points are MLP$RDATAN2 and DATAN2, and the call-by-value entry point is MLP$VDATAN2.

The DATAN2 vector math function is divided into three routines having three separate entry points defined as follows:

```
DTAN2(scalar,vector) = MLP$DATAN2SV
DTAN2(vector,scalar) = MLP$DATAN2VS
DTAN2(vector,vector) = MLP$DATAN2VV
```

The input domain is the collection of all valid double precision pairs (x,y) such that both quantities are not zero. The output range is included in the set of double precision quantities greater than −pi and less than or equal to pi.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Register pair (X4,X5) holds the absolute value of the first argument. Register pair (X7,X8) holds the absolute value of the second argument.

```
B4 = (X9) = sign mask of the first word of the first argument.
B3 = (XA) = complement of the sign mask of the first word of the second
argument.
B7 = (XB) = closest multiple of pi/2 to the result value.
```

If (X4) > (X7), then:

```
B7 = (XB) = 1.0.
Branch to label DATCOM to complete processing.
```

If (X4) ≤ (X7), then:

```
Exchange (X7) and (X4) and (X8) and (X5).
Complement contents of B3.
B7 = (XB) = 0, if the first word of the second argument is positive.
B7 = (XB) = 2, if the first word of the second argument is negative.
Branch to label DATCOM to complete processing.
```

At label DATCOM:

```
(X9) = B4 = mask MS = sign of the final result.
(XA) = B3 = mask MI.
(XB) = B7 = closest multiple of pi/2 to the absolute value of the result.
Register pair (X7,X8) = DU = smaller of DU and DB = min(x,y).
Register pair (X4,X5) = DV = larger of DU and DV = max(x,y).
```

At label DATCOM10:

```
Register pair (X7,X8) = DQ = DU/DV, which is < 1.0.
```

ATNU is the start of an 18-word table containing atan(n/8) ($0 \leq n \leq 8$) in double precision. Label DATCOM corresponds to step a (on the following page).

Constants used in the algorithm are:

```
d3  =  -.333 333 333 333 333 333 333 333 285 915
d5  =    .199 999 999 999 999 999 999 673 046 526
d7  =  -.142 857 142 857 142 856 280 180 055 289
d9  =    .111 111 111 111 109 972 932 035 508 119
c11 =  -.090 909 090 908 247 503
c13 =    .001 351 201 845 778 152
a   =  -.085 666 743 757 593 089
b   = -1.133 579 709 202 919 6
```

where d3, d5, d7, and d9 are double precision constants, and c11, c13, a, and b are real constants. Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. Boolean operations have B subscripts.

The algorithm used is:

```
a.  DQ = DU/DV in double precision.

b.  If both DU and DV are zero, error exit occurs.

c.  n = nearest multiple of 1/8 to DQ.

d.  If n = 0, go to step f.

e.  DA = (DQ - n/8)/(1 + n/8*DA), computed in double precision.

f.  Z  = 0
    DC = 0
    If (DA)(u) = 0, go to step i.

g.  XX = DA(u)*DA(u)
    DC = XX*(d)(d3 +(d) XX*(d)(d5 +(d) XX*(d) (d7 +(d) XX*(d)(d9 +(d)
         XX*(d)(d11 +(d) XX*(u)(c13 +(u) a/(b -(u) XX))))))

h.  w = DA + (d) DC*DA

i.  DB = 0
    If (XB) not= 0  DB = ATN(9)*2*(XB)

j.  BBAR = (B7*pi/2) - (B)B3 (upper and lower)

k.  CBAR = BBAR + (D)ATN(n/8).  ATN(n/8) is obtained as a double precision
    quantity from a table of precomputed values.

l.  Result = (CBAR + (D) w) - (B) (B3 - (B)B4).
```

At the end of processing, register pair (XE,XF) contains DATAN2 result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is infinite.

x and y are infinite.

x and y are equal to 0.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of relative error in the algorithm is 1.622E−29.

Effect of Argument Error

If small errors e(x) and e(y) occur in the arguments x and y, respectively, the error in the result is given approximately by:

```
(x*e(y) - y*e(x))/(x**2 + y**2)
```

Example of DATAN2 Called From FORTRAN

Source Code:

```
      PROGRAM DATAN2_EXAMPLE
C
      DOUBLE PRECISION x, y
      x=0.5d0
      y=5.0d0
      PRINT *, 'The DATAN2 of x,y is:'
      PRINT *, DATAN2(x,y)
      END
```

Output:

```
The DATAN2 of x,y is:
.09966865249116202737844611 99
```

DCOS

DCOS computes the cosine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDCOS and DCOS, the call-by-value entry point is MLP$VDCOS, and the vector entry point is MLP$DCOSV.

The input domain is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x − n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [−pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [−pi/4,pi/4], the sine polynomial approximation is:

```
p(x) =  a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 + a(11)x**11 +
        a(13)x**13** + a(15)x**15 + a(17)x**17 + a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) =  b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 + b(10)x**10 +
        b(12)x**12 + b(14)x**14 + b(16)x**16 + b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =  .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =  .833 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =  .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =  .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =  .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =  .194 362 013 130 224*10**-19
b(0)  =  .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =  .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =  .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =  .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =  .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =  .408 023 947 777 860*10**-18
```

These polynomials are evaluated from right to left in double precision. The sign flag is used to give the result the correct sign before returning to the calling program.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of the error of approximation of p(x) to sin(x) over (−pi/4,pi/4) is .2570E−28, and of q(x) to cos(x) is .3786E−28.

The function DCOS was tested against 4*DCOS(x/3)**3 - 3*DCOS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199D+02,.2356D+02]. Statistics on relative error were observed: maximum relative error was .2057D−23; root mean square relative error was .4606D−25.

Effect of Argument Error

If a small error e occurs in the argument x, the resulting error in cos is given approximately by −e*sin(x). If the error e becomes significant, the addition formulas for sin and cos should be used to compute the error in the result.

Example of DCOS Called From FORTRAN

Source Code:

```
      PROGRAM DCOS_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DCOS of x is:'
      PRINT *, DCOS(x)
      END
```

Output:

```
The DCOS of x is:
.8775825618903727161162815 83
```

DCOSH

DCOSH computes the hyperbolic cosine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDCOSH and DCOSH, the call-by-value entry point is MLP$VDCOSH, and the vector entry point is MLP$DCOSHV.

The input domain is the collection of all valid double precision quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid double precision quantities greater than or equal to 1.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formulas used for computation are:

```
u       = exp(x)*.5
v       = exp(-x)*.5
cosh(x) = u + v
```

The routine calls DEXP to compute exp(x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-11 shows a summary of these statistics.

Table 8-11. Relative Error of DCOSH

Test	Interval From	Interval To	Maximum	Root Mean Square
DCOSH(x) against Taylor series expansion of DCOSH(x)	0.0000D+00	.5000D+00	.2524D−28	.1739D−28
DCOSH(x) against c*(DCOSH(x + 1) + DCOSH(x − 1))	.3000D+01	.2838D+04	.1023D−27	.4548D−28

Effect of Argument Error

If a small error e occurs in the argument x, the error in cosh(x) is approximately sinh(x)*e.

Example of DCOSH Called From FORTRAN

Source Code:

```
      PROGRAM DCOSH_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DCOSH of x is:'
      PRINT *, DCOSH(x)
      END
```

Output:

```
The DCOSH of x is:
1.12762596520638078522622516
```

DDIM

DDIM computes the positive difference between two arguments. It accepts two double precision arguments and returns a double precision result.

The call-by-reference entry points are MLP$RDDIM and DDIM, and the call-by-value entry point is MLP$VDDIM.

The input domain is the collection of all valid double precision pairs (x,y) such that x − y is a valid double precision quantity. The output range is included in the set of valid, nonnegative double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x − y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit of the difference is extended across another word to form a mask. The boolean product of the mask's complement and the upper and lower word of the difference is formed.

Given arguments (x,y):

```
result = x - y if x > y
result = 0 if x ≤ y.
```

Example of DDIM Called From FORTRAN

Source Code:

```
      PROGRAM DDIM_EXAMPLE
C
      EXTERNAL DDIM
      DOUBLE PRECISION x,y
      x=999999.99d0
      y=99.0d0
      PRINT *, 'The DDIM of x,y is:'
      PRINT *,DDIM(x,y)
      END
```

Output:

```
The DDIM of x,y is:
999900.99
```

DEXP

DEXP computes the exponential function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDEXP and DEXP, the call-by-value entry point is MLP$VDEXP, and the vector entry point is MLP$DEXPV.

The input domain is the collection of all valid double precision quantities whose value is greater than or equal to −4097*log(2) and less than or equal to 4095*log(2). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than −4097*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The argument reduction performed is:

```
x = argument
y = x - n*log(2)
```

where y = is in [−1/2 log(2), 1/2 log(2)] and n is an integer.

Constants used in the algorithm are:

```
1.0/log(2)
log(2) (in double precision)
d3  =    .166 666 666 666 666 666 666 666 666 709
d5  =    .833 333 333 333 333 333 333 331 234 953*10**-2
d7  =    .198 412 698 412 698 412 700 466 386 658*10**-3
d9  =    .275 573 192 239 858 897 408 325 908 796*10**-5
pc  =   -.474 970 880 178 988*10**-10
pa  =    .566 228 284 957 811*10**-7
pb  = 272.110 632 903 710
c11 =    .250 521 083 854 439*10**-7
```

Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. An operand with a u or l subscript denotes the first or second word, respectively, of the double precision pair of words containing the operand.

On input, the argument is in register pair X2-X3, and on output, the result is in register pair XE-XF.

The algorithm used is:

```
a.  x = argument.  If x = 0, set DEXP = 1.0.  Return.

b.  If x not= 0,
    n = nearest integer to x/log(2),
    y = x - n*log(2).
    Then y is in [-1/2*log(2),1/2*log(2)].

c.  q = (y)(u)*(u)(y)(u)

d.  p =  q*(d)(d3 +(d) q*(d)(d5 +(d) q*(d)(d7 +(d) q*(d)(d9 +(d)
         q*(d)(c11 +(d) q*(d)(pa/(pb - q) + pc)))))

e.  s = (y)(u) +(d) (y)(u)*(d)p

f.  Compute hm = sqrt(1.0 + s**2).
    hi  = 3*q + ((s)(u))**2 in real.
    hj  = hi + hi
    hk  = 2*(1.0 + hj)
    hl  = ((y)(u)*(u)(y)(u) - hj)/hk - hi
    hm  = hj +(u) (hk -(u) hl)*(u)(hl/hk)
        (hm now carries cosh - 1.0 in single precision.)

g.  DS = s + (d)(((y)(l) + (r)(y)(l)*(u)hm) + (r)((s)(l) +
       (r)((y)(u)* (l)(p)(u) + (r)(y)(u)*(r)(p)(l))))
       (DS now contains sinh(y) in double precision.)

h.  DC =  hm +(d) (DS*DS - 2*hm - hm*hm)/(2(1.0 + hm)) computed in
          double precision.

i.  DX = DS + DC

j.  Clean up DS, DC, DX with (X7) = n.
    Register pair XA-XB = DS = sinh(y).
    Register pair X8-X9 = DC = cosh(y) - 1.0.
    Register pair X4-X5 = DX = exp(y).

k.  Increase the exponents of exp(y) by n.

l.  Return.
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than −4097*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-12 shows a summary of these statistics.

Table 8-12. Relative Error of DEXP

Test	Interval From	Interval To	Maximum	Root Mean Square
DEXP(x − 2.8125) against DEXP(x) / DEXP(2.8125)	−.3466D+01	−.2772D+04	.9240D−28	.2956D−28
DEXP(x − .0625) against DEXP(x) / DEXP(.0625)	−.2841D+00	.3466D+00	.6449D−28	.1680D−28
DEXP(x − 2.8125) against DEXP(x) / DEXP(2.8125)	.6931D+01	.2838D+04	.9262D−28	.2907D−28

Effect of Argument Error

If a small error e occurs in the argument the error in the result y is given approximately by y*e.

Example of DEXP Called From FORTRAN

Source Code:

```
      PROGRAM DEXP_EXAMPLE
C
      DOUBLE PRECISION x
      x=3.0d0
      PRINT *, 'The DEXP of x is:'
      PRINT *,DEXP(x)
      END
```

Output:

```
The DEXP of x is:
20.085536923187667740928529 7
```

DIM

DIM computes the positive difference between two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RDIM and DIM, and the call-by-value entry point is MLP$VDIM.

The input domain is the collection of all valid real quantities (x,y), such that x − y is a valid real quantity. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x − y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit is extended across another word to form a mask. The boolean product of the mask's complement and the difference is formed.

Given arguments (x,y):

```
result = x - y if x > y
result = 0 if x ≤ y
```

Example of DIM Called From FORTRAN

Source Code:

```
      PROGRAM DIM_EXAMPLE
C
      EXTERNAL DIM
      x=30.0
      y=3000.0
      PRINT *, 'The positive difference between y and x is:  ', DIM(y,x)
      END
```

Output:

```
The positive difference between y and x is:  2970.
```

DINT

DINT returns the integer part of an argument after truncation. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDINT and DINT, and the call-by-value entry point is MLP$VDINT.

The input domain for this function is the collection of all valid double precision quantities. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument is added to a special floating-point zero with an exponent value that forces the argument's fraction bits to be shifted off when it is added to the argument. The result is returned.

Example of DINT Called From FORTRAN

Source Code:

```
      PROGRAM DINT_EXAMPLE
C
      EXTERNAL DINT
      DOUBLE PRECISION x
      x=333.333d0
      PRINT *, 'The integer part of double precision x is:'
      PRINT *,DINT(x)
      END
```

Output:

```
The integer part of double precision x is:
333.
```

DLOG

DLOG computes the natural logarithm function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDLOG and DLOG, the call-by-value entry point is MLP$VDLOG, and the vector entry point is MLP$DLOGV.

The input domain for this function is the collection of all valid, positive double precision quantities. The output range is included in the set of double precision quantities whose absolute value is less than 4095*log(2).

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is put into the form x = 2**k*w, where k is an integer, and 2**-1/2 ≤ w ≤ 2**1/2. Then log(x) is computed from:

```
log(x) = k*log(2) + log(w)
```

and k*log(2) is computed in double precision. A polynomial approximation u is evaluated in single precision using:

```
u = c(1)*t + c(3)*t**3 + c(5)*t**5 + c(7)*t**7
```

where t = (w - 1.0)/(1.0 + w)

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

The coefficients c(1), c(3), c(5), and c(7) are:

```
c(1) = 1.999 999 993 734 000
c(3) =  .666 669 486 638 944
c(5) =  .399 657 811 051 126
c(7) =  .301 005 922 238 712
```

This approximates log with a relative error of absolute value at most 3.133*10**-8 over (2**-1/2,2**-1/2). Newton's rule for finding roots[3] is then applied in two stages to the function exp(x) - w to yield the final approximation to log(w). The two stages are algebraically combined to yield the final approximation v:

```
v = u - (1.0 - x*exp(-u)) - (1.0 - x*exp(-u - (1.0 - x*exp(-u))))
```

z is made to be less than 1.0 by writing z = 1.0 - x*exp(-u), and v is computed using:

```
v = u - z(u) - z(l) - (z(u))**2*(.5 + z(u)/3)
```

where z = z(u) + z(l). This formula is obtained by neglecting terms that are not significant for double precision; exp(-u) is evaluated in double precision by the polynomial of degree 17. If entry was made at MLP$VDLOG10, after k*log(2) + log(w) has been evaluated, the result is multiplied by log(e) base 10 in double precision.

3. For a discussion of Newton's rule for finding roots, refer to any calculus text (for example, Calculus and Analytic Geometry by G. B. Thomas).

Error Analysis

The maximum absolute value of the error of approximation of the algorithm to log(x) is 1.555E−29 over the interval (2**(−.5),2**.5).

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-13 shows a summary of these statistics.

Table 8-13. Relative Error of DLOG

Test	Interval From	Interval To	Maximum	Root Mean Square
DLOG(x*x) against 2*DLOG(x)	.1600D+02	.2400D+03	.4479D−28	.1528D−28
DLOG(x) against DLOG(17x/16) − DLOG(17/16)	.7071D+00	.9375D+00	.9041D−27	.1478D−27

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/x.

Example of DLOG Called From FORTRAN

Source Code:

```
      PROGRAM DLOG_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The natural logarithm of x is:'
      PRINT *, DLOG(x)
      end
```

Output:

```
The natural logarithm of x is:
-.69314718055994530941723212 1
```

DLOG10

DLOG10 computes the common logarithm function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDLOG10 and DLOG10, the call-by-value entry point is MLP$VDLOG10, and the vector entry point is MLP$DLOG10V.

The input domain for this function is the collection of all valid, positive double precision quantities. The output range is included in the set of double precision quantities whose absolute value is less than 4095*log(2) base 10.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is put into the form x = 2**k*w, where k is an integer, and 2**-1/2 ≤ w ≤ 2**1/2. Then log(x) is computed from:

```
log(x) = k*log(2) + log(w)
```

and k*log(2) is computed in double precision. A polynomial approximation u is evaluated in single precision using:

```
u = c(1)*t + c(3)*t**3 + c(5)*t**5 + c(7)*t**7
```

where t = (w − 1.0)/(1.0 + w)

The coefficients c(1), c(3), c(5), and c(7) are:

```
c(1) = 1.999 999 993 734 000
c(3) =  .666 669 486 638 944
c(5) =  .399 657 811 051 126
c(7) =  .301 005 922 238 712
```

This approximates log with a relative error absolute value at most 3.133*10**−8 over (2**−1/2,2**−1/2). Newton's rule for finding roots[4] is then applied in two stages to the function exp(x) − w to yield the final approximation to log(w). The two stages are algebraically combined to yield the final approximation v:

```
v = u - (1.0 - x*exp(-u)) - (1.0 - x*exp(-u - (1.0 - x*exp(-u))))
```

z is made to be less than 1.0 by writing z = 1.0 − x*exp(−u), and v is computed using:

```
v = u - z(u) - z(l) - (z(u))**2*(.5 + z(u)/3)
```

where z = z(u) + z(l). This formula is obtained by neglecting terms that are not significant for double precision; exp(−u) is evaluated in double precision by the polynomial of degree 17. If entry was made at MLP$VDLOG10, after k*log(2) + log(w) has been evaluated, the result is multiplied by log(e) base 10 in double precision.

4. For a discussion of Newton's rule for finding roots, refer to any calculus text (for example, Calculus and Analytic Geometry by G. B. Thomas).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function DLOG10 was tested against DLOG10(11x/10) - DLOG10(11/10). Groups of 2000 arguments were chosen randomly from the interval [.3162D+00,.9000D+00]. Statistics on relative error were observed: maximum relative error was .5417D−27; root mean square relative error was .8117D−28.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/x.

Example of DLOG10 Called From FORTRAN

Source Code:

```
      PROGRAM DLOG10_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The common logarithm of x is:'
      PRINT *, DLOG10(x)
      END
```

Output:

```
The common logarithm of x is:
-.30102999566398119521373889 5
```

DMOD

DMOD returns the remainder of the ratio of two arguments. It accepts two double precision arguments and returns a double precision result.

The call-by-reference entry points are MLP$RDMOD and DMOD, and the call-by-value entry point is MLP$VDMOD.

The input domain for this function is the collection of all valid double precision pairs (x,y), where y is nonzero and x/y is a valid quantity. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

y is equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The function computed by DMOD(x,y) is:

```
x - (x/y)*y
```

where parentheses denote truncation. The result of x/y is found and then added to a special floating-point zero that forces truncation.

Example of DMOD Called From FORTRAN

Source Code:

```
      PROGRAM DMOD_EXAMPLE
C
      EXTERNAL DMOD
      DOUBLE PRECISION x, y
      y=750.0d0
      x=140.0d0
      PRINT *, 'The remainder of the ratio of y and x is:'
      PRINT *, DMOD(y,x)
      END
```

Output:

```
The remainder of the ratio of y and x is:
50.
```

DNINT

DNINT returns the nearest whole number to an argument. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDNINT and DNINT, and the call-by-value entry point is MLP$VDNINT.

The input domain for this function is the collection of all valid double precision quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it and the result is added to a special floating-point zero that forces truncation. If the argument is < 0, $-.5$ is added to it and the result is treated as above.

Example of DNINT Called From FORTRAN

Source Code:

```
      PROGRAM DNINT_EXAMPLE
C
      EXTERNAL DNINT
      DOUBLE PRECISION x
      x=99.99d0
      PRINT *, 'The DNINT of x is:'
      PRINT *, DNINT(x)
      END
```

Output:

```
The DNINT of x is:
100.
```

DPROD

DPROD computes the product of two arguments. It accepts two real arguments and returns a double precision result.

The call-by-reference entry points are MLP$RDPROD and DPROD, and the call-by-value entry point is MLP$VDPROD.

The input domain for this function is the collection of all valid real pairs (x,y) such that x*y is a valid double precision quantity. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Given argument pair (x,y), the result of x*y is found.

Example of DPROD Called From FORTRAN

Source Code:

```
      PROGRAM DPROD_EXAMPLE
      EXTERNAL DPROD
C     Accepts two real arguments.  Returns a double precision result.
      x=140.0
      y=750.0
      PRINT *, 'The DPROD of x and y is:'
      PRINT *, DPROD(x,y)
      END
```

Output:

```
The DPROD of x and y is:
105000.
```

DSIGN

DSIGN transfers the sign of the second argument to the sign of the first. It accepts two double precision arguments and returns a double precision result.

The call-by-reference entry points are MLP$RDSIGN and DSIGN, and the call-by-value entry point is MLP$VDSIGN.

The input domain for this function is the collection of all valid double precision pairs (x,y). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

No errors are generated by DSIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the second argument is isolated by a mask with all other bits zero. The sign bits of the upper and lower words of the first argument are cleared by a boolean AND mask and replaced by the sign of the second argument by a boolean inclusive OR with the complement of the mask.

Given arguments (x,y):

```
result =  |x| if y is nonnegative
result = -|x| if y is negative
```

Example of DSIGN Called From FORTRAN

Source Code:

```
      PROGRAM DSIGN_EXAMPLE
C
      EXTERNAL DSIGN
      DOUBLE PRECISION x, y
      x=-140.0d0
      y=750.0d0
      PRINT *, 'The DSIGN of x,y is:'
      PRINT *, DSIGN(x,y)
      END
```

Output:

```
The DSIGN of x,y is:
140.
```

DSIN

DSIN computes the sine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDSIN and DSIN, the call-by-value entry point is MLP$VDSIN, and the vector entry point is MLP$DSINV.

The input domain for this function is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x − n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [−pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [−pi/4,pi/4], the sine polynomial approximation is:

```
p(x) = a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 + a(11)x**11 +
       a(13)x**13** + a(15)x**15 + a(17)x**17 + a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) = b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 + b(10)x**10 +
       b(12)x**12 + b(14)x**14 + b(16)x**16 + b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =  .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =  .833 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =  .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =  .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =  .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =  .194 362 013 130 224*10**-19
b(0)  =  .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =  .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =  .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =  .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =  .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =  .408 023 947 777 860*10**-18
```

These polynomials are evaluated from right to left in double precision. The sign flag is used to give the result the correct sign before returning to the calling program.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The maximum absolute value of the error of approximation of p(x) to sin(x) over (-pi/4,pi/4) is .2570E-28, and of q(x) to cos(x) is .3786E-28.

The function DSIN was tested against the 3*DSIN(x/3) - 4*DSIN(x/3)**3. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-14 shows a summary of these statistics.

Table 8-14. Relative Error of DSIN

Interval From	Interval To	Maximum	Root Mean Square
0.0000D+00	.1571D+01	.5153D-28	.1254D-28
.1885D+02	.2042D+02	.2764D-23	.6188D-25

Effect of Argument Error

If a small error e occurs in the argument x, the resulting error in sin is given approximately by e*cos(x). If the error e becomes significant, the addition formulas for sin and cos should be used to compute the error in the result.

Example of DSIN Called From FORTRAN

Source Code:

```
      PROGRAM DSIN_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DSIN of x is:'
      PRINT *, DSIN(x)
      END
```

Output:

```
The DSIN of x is:
.47942553860420300027328793 5
```

DSINH

DSINH computes the hyperbolic sine function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDSINH and DSINH, the call-by-value entry point is MLP$VDSINH, and the vector entry point is MLP$SINHV.

The input domain for this function is the collection of all valid double precision quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result is returned to the calling program.

Call-By-Value Routine

Most of the computation is performed in routine DEULER, and the constants used are listed there. The argument reduction performed in DEULER is:

```
x = argument
y = reduced argument
y = x - n*log(2)
```

where n is an integer, and y is in the interval [-1/2*log(2),1/2*log(2)].

The formula used for computation is:

```
sinh(y + n*log(2)) = (cosh(y) + sinh(y))*2**(n-1.0) - (cosh(y) -
                     sinh(y))*2**(-n-1.0)
```

where

```
cosh(y) = DC, and sinh(y) = DS as computed in routine DEULER.
```

On input, the argument is in register pair (X2,X3), and on output, the result is in register pair (XE,XF).

See the description of routine DEULER in chapter 9, Auxiliary Routines, for detailed information.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-15 shows a summary of these statistics.

Table 8-15. Relative Error of DSINH

Test	Interval From	Interval To	Maximum	Root Mean Square
DSINH(x) against Taylor series expansion of DSINH(x)	0.0000D+00	.5000D+00	.1184D−27	.3084D−28
DDINH(x) against c*(DSINH(x + 1) + DSINH(x − 1))	.3000D+01	.2838D+04	.1178D−27	.4582D−28

Effect of Argument Error

If a small error e occurs in the argument x, the error in sinh(x) is approximately cosh(x)*e.

Example of DSINH Called From FORTRAN

Source Code:

```
      PROGRAM DSINH_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DSINH of x is:'
      PRINT *, DSINH(x)
      END
```

Output:

```
The DSINH of x is:
.52109530549374736 1622425626
```

DSQRT

DSQRT computes the square root. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDSQRT and DSQRT, the call-by-value entry point is MLP$VDSQRT, and the vector entry point is MLP$DSQRTV.

The input domain for this function is the collection of all valid, nonnegative double precision quantities. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

An initial approximation to sqrt(y) is obtained by evaluating, inline, the sqrt of y(u) in single precision.

One Heron's iteration is performed in double precision using y and the initial approximation of sqrt(y), giving the double precision result.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The algorithm error is at most 2.05E−31, and is always positive.

The function DSQRT was tested against DSQRT(x*x) − x. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-16 shows a summary of these statistics.

Table 8-16. Relative Error of DSQRT

Interval From	Interval To	Maximum	Root Mean Square
.1000D+01	.1414D+01	.0000D+00	.0000D+00
.7071D+00	.1000D+01	.1785D−28	.9981D-29

Effect of Argument Error

For a small error e in the argument y, the amplification of absolute error is e/2*sqrt(y)).

Example of DSQRT Called From FORTRAN

Source Code:

```
      PROGRAM DSQRT_EXAMPLE
C
      DOUBLE PRECISION x
      x=49.0d0
      PRINT *, 'The DSQRT of x is:'
      PRINT *, DSQRT(x)
      END
```

Output:

```
The DSQRT of x is:
7.
```

DTAN

DTAN is a function that computes the tangent function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDTAN and DTAN, the call-by-value entry point is MLP$VDTAN, and the vector entry point is MLP$DTANV.

The input domain for this function is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument reduction is performed in two steps:

1. A pi/2 reduction is performed first. If the argument is outside the interval [−pi/4,pi/4], a signed integer multiple n of pi/2 is computed such that, after adding it to the argument, the result z falls in the interval [−pi/4,pi/4].

2. A 1/8 reduction is performed next. A signed integer m, which is a multiple of 1/8, is subtracted from z such that the result is in the interval [−1/16,1/16]. A small number e(m) is also subtracted from z. The value of e(m) is constant such that the tangent of m/8 + e(m) can be represented to double precision accuracy in a single precision word. The lower word is zero. Therefore, the original argument y is reduced to x as follows:

   ```
   x = y - (n*pi/2) - (m/8 + e(m))
   ```

The following quantities are computed from the reduced argument x and from the range reduction values. The functions U and L represent "upper of" and "lower of" functions.

```
t = tan(m/8 + e(m))
r = L(U(x)**2)/2U(x) + L(x)
a = L(U(x)**2) + 2L(x)U(x)
b = U(U(x)**2)
```

Since:

```
tan(x) = tan(sqrt(x**2))
       = tan(sqrt(U(U(x)**2 + L(U(x)**2) + 2L(x)U(x)))
       = tan(sqrt(b + a))
       = tan(sqrt(b) + a/2b)
       = tan(sqrt(b) + r)
```

Then s = sqrt(b) = U(x) − L(U(x)**2)/2U(x)

The value of the original argument y is:

```
tan(y) = tan(x + n*pi/2 + m/8 + e(m))
```

The effect of the n*pi/2 term on the final result is:

```
tan(y) = tan(x + m/8 + e(m)), if n is even
tan(y) = 1/tan(x + m/8 + e(m)), if n is odd
```

Applying the tangent addition formula gives:

```
tan(x + m/8 + e(m)) = tan(s + r + (m/8 + e(m))

         tan(s) + tan(r) + t - tan(s)*tan(r)*t
   =   ---------------------------------------------
         1.0 - tan(s)*tan(r) - tan(r)*t - t*tan(s)


         tan(s) + r + t - tan(s)*r*t
   =   ---------------------------------
         1.0 - tan(s)*r - r*t - t*tan(s)
```

Tan(s) is computed by using the general polynomial form:

```
x + x**3/3 + x**5*2/315 ...
```

After Chebyshev is applied to the coefficients, the form is:

```
tan(s) = s + s*(c(1)s**2 + c(2)s**4 + c(3)s**6 + c(4)s**8 +
         (a/(b - s**2))s**10)
```

where a = .0218 ... and b = 2.467 ...

The quotient is inverted if n is odd.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The algorithm error has a negligible effect on the total error. The worst relative error of the algorithm is 1.032E−29. There is a negligible error introduced by the pi/2 range reduction except for points close to nonzero multiples of pi/2. Near pi/2, the pi/2 reduction relative error is bounded by 2**(n−155) where n is the number of bits of precision to which the argument represents pi/2. At larger multiples of pi/2, similar problems occur.

The function DTAN was tested against 2*DTAN(x/2)/(1 − DTAN(x/2)**2). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-17 shows a summary of these statistics.

Table 8-17. Relative Error of DTAN

Interval From	Interval To	Maximum	Root Mean Square
.0000D+00	.7854D+00	.1946D−27	.4491D−28
.1885D+02	.1963D+02	.1729D−27	.4480D−28
.2749D+01	.3534D+01	.2008D−27	.5363D−28

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is e*sec(x)**2.

Example of DTAN Called From FORTRAN

Source Code:

```
      PROGRAM DTAN_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DTAN of x is:'
      PRINT *, DTAN(x)
      END
```

Output:

```
The DTAN of x is:
.546302489843790513255179466
```

DTANH

DTANH computes the hyperbolic tangent function. It accepts a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RDTANH and DTANH, the call-by-value entry point is MLP$VDTANH, and the vector entry point is MLP$DTANHV.

The input domain for this function is the collection of all valid double precision quantities. The output range is included in the set of valid quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Most of the computation is performed in routine DEULER, and the constants used are listed there. The argument reduction performed is:

1. For argument in [−47*log(2),47*log(2)] but not in [−1/2*log(2),1/2*log(2)]:

```
x = argument
y = reduced argument
y = 2x - n*log(2)
```

   where n is an integer, and y is in [−1/2*log(2),1/2*log(2)]

```
tanh(x) = u/v where

   u = 1.0 - 2**-n - 2**-n*(DC - DS)
   v = 1.0 - 2**-n + 2**-n*(DC - DS)
```

2. For argument in [−1/2*log(2),1/2*log(2)]:

```
   x = argument
   y = reduced argument
   y = x
tanh(x) = DS(2*+DC)
```

3. For argument outside [−47*log(2),47*log(2)]:

```
   x = argument
   y = reduced argument
tanh(x) = 1.0 - 2((1.0 + DC - DS)*2**-n - ((1.0 + DC - DS)*2**-n)**2)
```

In steps 1, 2, and 3, DC = cosh(y) − 1.0 and DS = sinh(y), where DC + DS are computed in DEULER.

On input, the argument is in register pair (X2−X3), and on output, the result is in register pair (XE−XF).

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function DTANH was tested against (DTANH(x − 1/8) + DTANH(1/8))/(1 + DTANH(x − 1/8)*DTANH(1/8)). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-18 shows a summary of these statistics.

Table 8-18. Relative Error of DTANH

Interval From	Interval To	Maximum	Root Mean Square
.1250D+00	.5493D+00	.9403D−28	.2612D−28
.6743D+00	.3431D+02	.3282D−27	.2348D−28

Algorithm Error

The algorithm error is insignificant. It is predominated by the error in the sinh expression in DEULER, but by various folding actions, the error is reduced even further.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given by e*sech(x)**2.

Example of DTANH Called From FORTRAN

Source Code:

```
      PROGRAM DTANH_EXAMPLE
C
      DOUBLE PRECISION x
      x=0.5d0
      PRINT *, 'The DTANH of x is:'
      PRINT *, DTANH(x)
      END
```

Output:

```
 The DTANH of x is:
 .46211715726000975850231848 4
```

DTOD

DTOD performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. DTOD also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RDTOD and DTOD, and the call-by-value entry point is MLP$VDTOD.

The DTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOD(scalar,vector) = MLP$DTODV
DTOD(vector,scalar) = MLP$DVTOD
DTOD(vector,vector) = MLP$DVTODV
```

The input domain for this function is the collection of all valid double precision pairs (x,y), where x is positive and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, positive double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0.
```

Upon entry, the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function DTOD was tested. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-19 shows a summary of these statistics.

Table 8-19. Relative Error DTOD

Test	Interval From	Interval To	Maximum	Root Mean Square
	x interval			
x**y against x**2**(y/2)	.1000D−01	.1000D+02	.5172D−25	.9207D−26
	y interval			
	−.6167D+03	.6167D+03		
x**2**1.5 against	.1000D+01	.8053+411	.1133D−24	.4805D−25
x**2*x	.5000D+00	.1000D+01	.1143D−27	.3978D−28
x**1.0 against x	.5000D+00	.1000D+01	.7133D−28	.3195D−28

Effect of Argument Error

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

```
r*(log(b)*e(p) + p*e(b)/b)
```

Example of DTOD Called From FORTRAN

Source Code:

```
      PROGRAM DTOD_EXAMPLE
C
      DOUBLE PRECISION x, y, DTOD
      x=20.0d0
      y=140.0d0
      PRINT *, 'The DTOD of x and y is:'
      PRINT *, DTOD(x,y)
      END
```

Output:

```
The DTOD of x and y is:
1.39379657490816394634598238E+182
```

DTOI

DTOI performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. DTOI also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RDTOI and DTOI, and the call-by-value entry point is MLP$VDTOI.

The DTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOI(scalar,vector) = MLP$DTOIV
DTOI(vector,scalar) = MLP$DVTOI
DTOI(vector,vector) = MLP$DVTOIV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a double precision quantity and y is an integer quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

An x represents the base, and a y represents the exponent. If y is nonnegative and has the binary representation 000...0i(n)i(n−1)...i(1)i(0), where each i(j)($0 \leq j \leq n$) is 0 or 1, then:

```
y = i(n)*2**n + i(n-1)*2**(n-1) + ...  + i(1)*2**1 + i(0)*2**0
```

and n = (log(2)y) = greatest integer not exceeding log(2)y. Then:

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1].
```

The numbers x = x**0, x**2**0, x**2, x**4, ..., x**2n are generated by successive squarings, and the coefficients i(n), ..., i(0) are obtained as the sign bits of successive circular left shifts of y within the computer. A running product is formed during the computation so that smaller powers of x and earlier coefficients i(j) can be discarded. Thus, the computation becomes an iteration of the algorithm:

```
x**y = 1, if y = 0 and x not= 0.
     = (x**2)**(y/2), if y > 0 and y is even.
     = x*(x**2)**((y - 1)/2), if y > 0 and y is odd.
```

Upon entry, if the exponent y is negative, the following steps are performed with R(k) representing the running product after k iterations:

1. y is replaced by −y.
2. y is shifted right (end-off) by 1.

   This effectively divides y by 2 and the final multiplications are completed after the running product, R(n−1) is replaced by 1/R(n−1) in the case of exponent overflow for very large negative exponents.
3. The algorithm continues as if the exponent was positive with the above formula for (n−1) iterations.
4. Either of the following two methods produces the final result R(n):

   a. If the final multiplication (depending on i/n and the last bit of the power) R(n−1) ** 2 * (x ** i(j)) gives exponent overflow, then the running product after (n−1) iterations is inverted and the result is:

      ```
      R(n) = (1/(R(n-1)) * (1/(x ** i(j))), j = n
      ```

   b. If there is no exponent overflow in the final multiplication, the result is:

      ```
      R(n) = (1/(R(n-1) ** 2 * (x ** i(j)))
      ```

In the routine, double precision quantities a = a(u)*a(l) and b = b(u)*b(l) are multiplied according to:

```
a*b = (a*b)(u)*(a*b)(l)
```

where:

```
(a*b)(u) = (((a(u)*b(l)) + (a(l)*b(u))) + (a(u)*(l)b(u))) + (a(u)*b(u))
```

and

```
(a*b)(l) = (((a(u)*b(l)) + (a(l)*b(u))) + (a(u)*(l)b(u))) + (l)(a(u)*b(u))
```

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Effect of Argument Error

If a small error e occurs in the base b, the error in the result will be given approximately by n*b**(n−1)*e, where n is the exponent given to the routine.

Example of DTOI Called From FORTRAN

Source Code:

```
      PROGRAM DTOI_EXAMPLE
C
      INTEGER i
      DOUBLE PRECISION d, dtoi
      i=2
      d=10.0d0
      PRINT *, 'The DTOI of d and i is:'
      PRINT *, DTOI(d,i)
      END
```

Output:

```
The DTOI of i and d is:
100.
```

DTOX

DTOX performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. DTOX also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RDTOX and DTOX, and the call-by-value entry point is MLP$VDTOX.

The DTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOX(scalar,vector) = MLP$DTOXV
DTOX(vector,scalar) = MLP$DVTOX
DTOX(vector,vector) = MLP$DVTOXV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a nonnegative double precision quantity and y is a real quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, positive double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

See the description of function DTOD.

Effect of Argument Error

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

```
r*(e(p)*log(b) + p*e(b)/b)
```

Example of DTOX Called From FORTRAN

Source Code:

```
      PROGRAM DTOX_EXAMPLE
C
      REAL x
      DOUBLE PRECISION d, dtox
      x=2.0
      d=10.0d0
      PRINT *, 'The DTOX of d and x is:'
      PRINT *, DTOX(d,x)
      END
```

Output:

```
The DTOX of d and x is:
100.
```

DTOZ

DTOZ performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. DTOZ also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RDTOZ and DTOZ, and the call-by-value entry point is MLP$VDTOZ.

The DTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOZ(scalar,vector) = MLP$DTOZV
DTOZ(vector,scalar) = MLP$DVTOZ
DTOZ(vector,vector) = MLP$DVTOZV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a double precision quantity and y is a complex quantity. If x is equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y is not equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

If the base is real and the exponent is complex, then:

```
base**exponent = x + i*y
```

Upon entry, the double precision base, x, is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y is not equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(b) occurs in the base b and a small error e(z) occurs in the exponent z, the error in the result w is given approximately by:

```
w*(e(z)*log(b) + z*e(b)/b)
```

Example of DTOZ Called From FORTRAN

Source Code:

```
      PROGRAM DTOZ_EXAMPLE
C

      COMPLEX zeta, dtoz
      DOUBLE PRECISION d
      zeta = (5.0, -1)
      d=10.0d0
      PRINT *, 'The DTOZ of d and zeta is:'
      PRINT *, DTOZ(d,zeta)
      END
```

Output:

```
The DTOZ of d and zeta is:
(-66820.15101903, -74398.03369575)
```

ERF

ERF computes the error function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RERF and ERF, the call-by-value entry point is MLP$VERF, and the vector entry point is MLP$ERFV.

The input domain for this function is the collection of all valid real quantities. The output range is included in the set of real quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The routine calculates the smaller of erf(abs(x)) and erfc(abs(x)). The final value, which is the sum of a signed function and a constant, is computed by using the identities:

```
erf(-x) = -erf(x)
erf(x)  = 1.0 - erfc(x)
```

The forms used in ERF (y = ABS(x)) are given in table 8-20.

Table 8-20. Forms Used in ERF

Range	ERF	ERFC
[−INF,−5.625]	−1.0	+2.0
(−5.625,−.477)	−1.0+p2(y)	+2.0−p2(y)
[−.477,0)	−p1(y)	+1.0+p1(y)
[0,+.477]	+p1(y)	+1.0−p1(y)
[.477,5.625)	+1.0−p2(y)	p2(y)
[5.625,8.0)	+1.0	p2(y)
[8.0,53.0]	+1.0	p3(y)
(53.0,+INF)	+1.0	underflow
+INF	+1.0	0.0

The constants .477 and 53.0 are inverse erf(.5) and inverse erfc(2**−975), which are approximately .47693627620447 and 53.0374219959898.

The function p1 is a (5th order odd)/(8th order even) rational form. The functions p2 and p3 are exp(−x**2)*(rational form), where p2 is (7th order)/(8th order) and p3 is (4th order)/(5th order). Since exp(−x**2) is ill-conditioned for large x, exp(−x**2) is calculated by exp(u + e) = exp(u) + e*exp(u), where u = −x**2 upper and e = −x**2 lower.[5]

5. The coefficients for p2 and p3 are from Hart, Cheney, Lawson, et al., Computer Approximations, New York, 1968, John Wiley and Sons.

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function ERF was tested against 1 – e**(–x**2)*p(x)/q(x)'. A group of 10,000 arguments was chosen randomly from the interval (0.0,8.0). The maximum relative error of these arguments was found to be .2050E–13.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(–x**2) and that of relative error is (2/sqrt(pi))*x*exp(–x**2)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC when $x < .53$. For $x > .53$, the relative error for ERFC is amplified by approximately 2x.

Example of ERF Called From FORTRAN

Source Code:

```
      PROGRAM ERF_EXAMPLE
C
      REAL x
      x=100000.0
      PRINT *, 'The error function of x is:'
      PRINT *, ERF(x)
      END
```

Output:

```
The error function of x is:
1.
```

ERFC

ERFC computes the complementary error function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RERFC and ERFC, the call-by-value entry point is MLP$VERFC, and the vector entry point is MLP$ERFCV.

The input domain for this function is the collection of all valid real quantities less than 53.037, but not equal to infinity. The output range is included in the set of valid, nonnegative real quantities less than or equal to 2.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is greater than 53.037, but not equal to infinity.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The routine calculates the smaller of erf(abs(x)) and erfc(abs(x)). The final value, which is the sum of a signed function and a constant, is computed by using the identities:

```
erf(-x) = -erf(x)
erf(x)  = 1.0 - erfc(x)
```

The forms used are given in table 8-20.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is greater than 53.037, but not equal to infinity.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function ERFC was tested against e**(−x**2)*p(x)/q(x)'. A group of 10,000 arguments was chosen randomly from the interval (0.0,8.0). The maximum relative error of these arguments was found to be .9531E−11.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(−x**2) and that of relative error is (2/sqrt(pi))*x*exp(−x**2)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC when x < .53. For x > .53, the relative error for ERFC is amplified by approximately 2x.

Example of ERFC Called From FORTRAN

Source Code:

```
      PROGRAM ERFC_EXAMPLE
C
      REAL x
      x=53.036
      PRINT *, 'The complementary error function of x is:'
      PRINT *, ERFC(x)
      END
```

Output:

```
The complementary error function of x is:
2.727387727515E-1224
```

EXP

EXP computes the exponential function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$REXP and EXP, the call-by-value entry point is MLP$VEXP, and the vector entry point is MLP$EXPV.

The input domain for this function is the collection of all valid real quantities whose value is greater than or equal to −4097*log(2) and less than or equal to 4095*log(2). The output range is included in the set of valid positive real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than −4097*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

If x is valid, EXP(x) is calculated by reducing it to the simpler task of approximating e**g*2**(NL/32). This reduction is derived as follows:

```
exp(x) = e**(g + (32*NH + NL)*(ln(2)/32))
       = e**(g + NH*ln(2) + (NL/32)*ln(2))
       = e**g*2**NH*2**(NL/32)
       = (e**g*2**(NL/32))*2**NH
```

where

```
-   n is the nearest integer to 32*x/ln(2).
-   g is a real number such that x = g + n*(ln(2)/32).  Thus,
    abs(g) is less than or equal to ln(2)/64.
-   NH is floor(n/32).
-   NL is greater than or equal to 0, less than or equal to 31,
    and is the integer such that n = 32*NH + NL.
```

The reduction:

```
e**g*2**(NL/32)
```

is approximated to 48 bits of precision using the following minimax approximation:

```
Z = Q(NL, g) + Qbias(NL)
```

where for each of the 32 values of NL, Qbias(NL) is a number that is represented exactly in binary floating-point and which is slightly less than 2**(−1/64)*2**(NL/32), which is the minimum value of e**g*2**(NL/32).

Q(NL, g) denotes the 32 quintic polynomials in g which approximate e**g*2**(NL/32) − Qbias(NL) with the lowest maximum relative error for abs(g) ≤ ln(2)/64. Z is evaluated with almost no error since the low bits of Q(NL, g), which may be inaccurate due to truncation errors, are insignificant with respect to Qbias(NL). Thus, Z*2**NH, which is evaluated simply by adding NH to the exponent of Z, is an accurate approximation to EXP(x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than −4097*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-21 shows a summary of these statistics.

Table 8-21. Relative Error EXP

Test	Interval From	Interval To	Maximum	Root Mean Square
EXP(x - 2.8125) against EXP(x)/EXP(2.8125)	−.3466E+01	−.2805E+04	.7335E−14	.3766E−14
EXP(x - .0625) against EXP(x)/EXP(.0625)	−.2841E+00	.3466E+00	.7557E−14	.3945E−14
EXP(x - 2.8125) against EXP(x)/EXP(2.8125)	.6931E+01	.2838E+04	.7384E−14	.3850E−14

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result y is given be y*e.

Example of EXP Called From FORTRAN

Source Code:

```
      PROGRAM EXP_EXAMPLE
C
      REAL x
      x=1000.0
      PRINT *, 'The EXP of x is:'
      PRINT *, EXP(x)
      END
```

Output:

```
The EXP of x is:
1.970071114017E+434
```

EXTB

EXTB extracts bits from the first argument, x, as specified by the second and third arguments, i1 and i2; that is, EXTB(x, i1, i2) extracts bits from x starting with position i1 with length of i2. It accepts any type except character for argument x and accepts integer for arguments i1 and i2. The result is boolean.

If x is of type double precision or complex, only the first word is used. The result is returned with a zero-filled second word. The following example FORTRAN program uses function EXTB with double precision arguments to illustrate the zero-filled second word.

Source Code:

```
      PROGRAM EXTB_EXAMPLE
C
      EXTERNAL EXTB
      DOUBLE PRECISION d1,d2
      BOOLEAN x(2),y(2)
      EQUIVALENCE (x(1),d1),(y(1),d2)
      x(1)=Z"1234567890ABCDEF"
      x(2)=Z"FEDCBA0987654321"
      y(1)=Z"1111111111111111"
      y(2)=Z"2222222222222222"
      d2=EXTB(d1,0,32)
      PRINT *,x(1),x(2)
      PRINT *,y(1),y(2)
      END
```

Output:

```
 Z"1234567890ABCDEF" Z"FEDCBA0987654321"
 Z"12345678" Z"0"
```

Argument x must be byte aligned and be at least 64 bits in length. The argument used is the leftmost 64 bits of x. Argument i1 indicates the first bit to be extracted numbering from bit 0 on the left. Argument i2 indicates the number of bits to be extracted. The extracted bits occupy the rightmost bits of the result, with 0 bits as fill on the left.

The call-by-reference entry points are MLP$REXTB and EXTB, and the call-by-value entry point is MLP$VEXTB.

The input domain for this function is such that i1 is greater than or equal to 0 and less than 64; i2 is greater than or equal to 0; and i1 + i2 is less than or equal to 64. If i2 = 0, the result is 0 (all 0 bits). The data type of argument x is not significant to the processing of this function. The output range is included in the set of valid boolean quantities.

Call-By-Reference Routine

The arguments i1 and i2 are checked upon entry. They are invalid if:

i1 is less than zero.

i2 is less than zero.

i1 is greater than or equal to 64.

i1 + i2 is greater than 64.

If the arguments are invalid, a diagnostic message is displayed. If the arguments are valid, the call-by-value routine is branched to, and the result of the function is returned to the calling program.

Call-By-Value Routine

The extracted bits from the first argument, x, as specified by the second and third arguments, i1 and i2, are returned. The leftmost 64 bits of x are used.

Example of EXTB Called From FORTRAN

Source Code:

```
      PROGRAM EXTB_EXAMPLE
C
      EXTERNAL EXTB
      REAL x
      INTEGER i1, i2
      x=Z"4321FEDCBA987654"
      i1=1
      i2=48
      PRINT *, 'The EXTB of x is:'
      PRINT *, EXTB(x,i1,i2)
      END
```

Output:

```
The EXTB of x is:
Z"4321FEDCBA98"
```

IABS

IABS computes the absolute value of an argument. It accepts an integer argument and returns an integer result.

The call-by-reference entry points are MLP$RIABS and IABS, and the call-by-value entry point is MLP$VIABS.

The input domain for this function is the collection of all valid integer quantities. The output range is included in the set of valid, nonnegative integer quantities.

Call-By-Reference Routine

No errors are generated by IABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the argument is extended throughout a word to form a mask. The argument is subtracted from the exclusive OR of the mask and the argument to form the result.

Example of IABS Called From FORTRAN

Source Code:

```
      PROGRAM IABS_EXAMPLE
C
      EXTERNAL IABS
      INTEGER i
      i=-40.0
      PRINT *, 'The absolute value of i is:'
      PRINT *, IABS(i)
      END
```

Output:

```
The absolute value of i is:
40
```

IDIM

IDIM computes the positive difference between two arguments. It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RIDIM and IDIM, and the call-by-value entry point is MLP$VIDIM.

The input domain for this function is the collection of all valid integer pairs (x,y) such that x - y is less than 2**63. The output range is included in the set of valid, nonnegative integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

x - y is greater than or equal to 2**63.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit is extended across another word to form a mask. The boolean product of the mask's complement and the difference is formed.

Given arguments (x,y):

```
result = x - y if x > y
result = 0 if x ≤ y.
```

Example of IDIM Called From FORTRAN

Source Code:

```
      PROGRAM IDIM_EXAMPLE
C
      EXTERNAL IDIM
      INTEGER i1,i2
      i1=1988
      i2=1929
      PRINT *, 'The IDIM of i1,i2 is:'
      PRINT *, IDIM(i1,i2)
      END
```

Output:

```
The IDIM of i1,i2 is:
59
```

IDNINT

IDNINT returns the nearest integer to an argument. It accepts a double precision argument and returns an integer result.

The call-by-reference entry points are MLP$RIDNINT and IDNINT, and the call-by-value entry point is MLP$VIDNINT.

The input domain for this function is the collection of all valid double precision quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it, and the result is added to a special floating-point zero that forces truncation. If the argument is < 0, $-.5$ is added to it, and the result is added to a special floating-point zero that forces truncation.

If the value of the argument is not in the range [-2**63 - 2**15,2**63 - 2**15], then the high order bits of the resulting integer are lost (the result is truncated in its leftmost position).

Example of IDNINT Called From FORTRAN

Source Code:

```
      PROGRAM IDNINT_EXAMPLE
C
      EXTERNAL IDNINT
      DOUBLE PRECISION x
      x=999.999d0
      PRINT *, 'The nearest integer to x is:'
      PRINT *, IDNINT(x)
      END
```

Output:

```
The nearest integer to x is:
1000
```

INSB

INSB inserts bits from the first argument, x, into a copy of the fourth argument, y, as specified by the second and third arguments, i1 and i2; that is, INSB(x, i1, i2, y) inserts bits from x starting with position i1 with length of i2 into a copy of y. It accepts any type except character for arguments x and y, and accepts integer for arguments i1 and i2. The result is boolean.

If x or y is of type double precision or complex, only the first word is used. The result is returned with a zero-filled second word; however, for double precision the first 4 bytes of the first word are duplicated in the second word. This duplication preserves the exponent in the second word. The following FORTRAN example uses function INSB with double precision arguments to illustrate the zero-filled second word and the duplication of the exponent 1111 in the second word.

Source Code:

```
      PROGRAM INSB_EXAMPLE
C
      EXTERNAL INSB
      DOUBLE PRECISION d1,d2,d3
      BOOLEAN x(2),y(2),z(2)
      EQUIVALENCE (x(1),d1),(y(1),d2),(z(1),d3)
      x(1)=Z"1234567890ABCDEF"
      x(2)=Z"FEDCBA0987654321"
      y(1)=Z"1111111111111111"
      y(2)=Z"2222222222222222"
      d3=insb(d1,16,16,d2)
      PRINT *,x(1),x(2),y(1),y(2)
      PRINT *,z(1),z(2)
      END
```

Output:

```
 Z"1234567890ABCDEF" Z"FEDCBA0987654321" Z"1111111111111111" Z"2222222222222222"
 Z"1111CDEF11111111" Z"1111000000000000"
```

Arguments x and y must be byte aligned and be at least 64 bits in length. The argument used is the leftmost 64 bits of each x and y. Argument i1 indicates first bit position in y for insertion. Argument i2 indicates the rightmost number of bits taken from x to be inserted into y.

The call-by-reference entry points are MLP$RINSB and INSB, and the call-by-value entry point is MLP$VINSB.

The input domain for this function is such that i1 is greater than or equal to 0 and less than 64; i2 is greater than or equal to 0; and i1 + i2 is less than or equal to 64. If i2 = 0, the result is the value of y. The data type of arguments x and y is not significant to the processing of this function. The output range is included in the set of valid boolean quantities.

Call-By-Reference Routine

The arguments i1 and i2 are checked upon entry. They are invalid if:

i1 is less than zero.

i2 is less than zero.

i1 is greater than or equal to 64.

i1 + i2 is greater than 64.

If the arguments are invalid, a diagnostic message is displayed. If the arguments are valid, the call-by-value routine is branched to, and the result of the function is returned to the calling program.

Call-By-Value Routine

The inserted bits from the first argument, x, into a copy of the fourth argument, y, as specified by the second and third arguments, i1 and i2, are returned. The leftmost 64 bits of x and y are used.

Example of INSB Called From FORTRAN

Source Code:

```
      PROGRAM INSB_EXAMPLE
C
      EXTERNAL INSB
      REAL x,y
      INTEGER i1, i2
      x=Z"4321FEDCBA987654"
      y=Z"0"
      i1=0
      i2=48
      PRINT *, 'The inserted bits from x, as specified by '
      PRINT *, '  i1 and i2, into a copy of y are: '
      PRINT *, INSB(x,i1,i2,y)
      END
```

Output:

```
The inserted bits from x, as specified by
  i1 and i2, into a copy of y are:
Z"FEDCBA9876540000"
```

ISIGN

ISIGN transfers the sign of one argument to another argument. It accepts two integer arguments and returns an integer result. The result is a copy of the first argument with the sign of the second argument.

The call-by-reference entry points are MLP$RISIGN and ISIGN, and the call-by-value entry point is MLP$VISIGN.

The input domain for this function is the collection of all valid integer quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

No errors are generated by ISIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The exclusive OR of the first argument, along with the second argument, is shifted to extend its sign bit across a word to produce a mask. The mask is then subtracted from the exclusive OR of the mask and argument to form the result.

Example of ISIGN Called From FORTRAN

Source Code:

```
      PROGRAM ISIGN_EXAMPLE
C
      EXTERNAL ISIGN
      INTEGER i1, i2
      i1=-140
      i2=750
      PRINT *, 'The ISIGN of i1, i2 is:'
      PRINT *, ISIGN(i1,i2)
      END
```

Output:

```
The ISIGN of i1, i2 is:
140
```

ITOD

ITOD performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. ITOD also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RITOD and ITOD, and the call-by-value entry point is MLP$VITOD.

The input domain for this function is the collection of all valid pairs (x,y), where x is a nonnegative integer quantity and y is a double precision quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

y is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0.
```

Upon entry, the integer argument is converted to double precision, and the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Error Analysis

See the description of function DTOD.

Effect of Argument Error

If a small error e occurs in the exponent, the error in the result r is given approximately by r*e*log(b), where b is the base.

Example of ITOD Called From FORTRAN

Source Code:

```
      PROGRAM ITOD_EXAMPLE
C
      INTEGER i
      DOUBLE PRECISION d, d1, itod
      i=2
      d=10.0d0
      d1=ITOD(i,d)
      PRINT *, 'The ITOD of i and d is:'
      PRINT *, d1
      END
```

Output:

```
The ITOD of i and d is:
1024.
```

ITOI

ITOI performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. ITOI also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RITOI and ITOI, and the call-by-value entry point is MLP$VITOI.

The input domain for this function is the collection of all valid integer pairs (x,y) such that the absolute value of x**y is less than 2**63. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

x is zero and y is zero or negative.

If the argument pair is invalid, zero is returned, and a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The arguments are checked to determine whether the exponentiation conforms to a special case. If it does, the proper value is immediately returned, or if the special case is an error condition, a hardware exception condition is forced. The special cases are:

```
 0**0 = error
 0**J = error if J < 0
 1**J = 1
-1**J = +1 or -1  (J even or odd)
 I**0 = 1
 I**J = 0 if J < 0
```

If the exponentiation does not fit any special case, the algorithm listed below is used for the computation.

An x represents the base and a y represents the exponent. If x has binary representation 000... .000i(n)i(n−1) ...i(i)i(0), where each i(j)(0 ≤ j ≤ n) is 0 or 1, then:

```
y = i(0)*2**0 + i(1)*2**1 + ... + i(n)*2**n
n = (log(2)y) = greatest integer not exceeding log(2)y
```

Then:

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1]
```

The numbers x = x**0, x**2**0, x**2, x**4, ..., x**(2)**n are generated during the computation by successive squarings, and the coefficients i(0), ..., i(n) are obtained as sign bits of successive right shifts of y within the computer. A running product is formed during the computation so that smaller powers of x can be discarded. The computation then becomes an iteration of the algorithm:

```
x**y = 1, if y = 1, and x not= 0
     = (x*x)**(y/2), if y > 0 and y is even
     = (x*x)**((y-1)/2)*x, if y > 0 and y is odd
```

Example of ITOI Called From FORTRAN

Source Code:

```
      PROGRAM ITOI_EXAMPLE
C
      INTEGER i1, i2, ix
      i1=2
      i2=8
      ix=ITOI(i1,i2)
      PRINT *, 'The ITOI of i1 and i2 is:'
      PRINT *, ix
      END
```

Output:

```
The ITOI of i1 and i2 is:
256
```

ITOX

ITOX performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ITOX also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RITOX and ITOX, and the call-by-value entry point is MLP$VITOX.

The input domain for this function is the collection of all valid pairs (x,y), where x is a nonnegative integer quantity, y is a real quantity, and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

y is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x ≥ 1
```

Upon entry, x is converted to real, and the routine calls XTOX to compute the result. Zero is returned if the base is zero and the exponent is positive.

Error Analysis

See the description of function XTOX.

Effect of Argument Error

If a small error e occurs in the exponent x, the error in the result r is given approximately by r*e*log(n), where n is the base.

Example of ITOX Called From FORTRAN

Source Code:

```
      PROGRAM ITOX_EXAMPLE
C
      INTEGER i
      REAL x, r, itox
      i=2
      x=8.8
      r=ITOX(i,x)
      PRINT *, 'The ITOX of i and x is:'
      PRINT *,r
      END
```

Output:

```
The ITOX of i and x is:
445.7218884076
```

ITOZ

ITOZ performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ITOZ also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RITOZ and ITOZ, and the call-by-value entry point is MLP$VITOZ.

The ITOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
ITOZ(scalar,vector) = MLP$ITOZV
ITOZ(vector,scalar) = MLP$IVTOZ
ITOZ(vector,vector) = MLP$IVTOZV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a nonnegative nonzero integer quantity and y is a complex quantity. If x is equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is infinite.

x is equal to zero, and the real part of y is zero or negative, or the imaginary part of y is not equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

If n is a positive integer, and x and y are real, then:

```
n**(x + i*y) = exp(x*log(n))*cos(y*log(n)) + i*exp(x*log(n))*sin(y*log(n))
```

Upon entry, n is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is infinite.

x is equal to zero, and the real part of y is zero or negative, or the imaginary part of y is not equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(z) = e(x) + i*e(y) occurs in the exponent z, the error in the result w is given approximately by w*log(n)*e(z).

Example of ITOZ Called From FORTRAN

Source Code:

```
      PROGRAM ITOZ_EXAMPLE
C
      INTEGER i
      COMPLEX z, zeta, itoz
      i = 50
      z = (5.0, -1)
      zeta = ITOZ(i,z)
      PRINT *, 'The ITOZ of i and z is:'
      PRINT *, zeta
      END
```

Output:

```
The ITOZ of i and z is:
(-224253443.769,217638790.1035)
```

MOD

MOD computes the remainder of the ratio of two arguments. It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RMOD and MOD, and the call-by-value entry point is MLP$VMOD.

The input domain for this function is the collection of all valid integer pairs (x,y), where x is an integer quantity and y is a nonzero integer quantity. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

Upon entry, the argument pair (x,y) is checked. It is invalid if:

y is equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned.

Call-By-Value Routine

Upon entry, the arguments x and y are converted to real, the quotient x/y is formed, and the result is multiplied by y and then subtracted from x.

Example of MOD Called From C

Source Code:

```
main()
{
   int i = 83;
   int j = 8;
   int k;
/* Use the left bit-shift operator (<<) to left justify the address
   16 bits.  This is necessary because the MOD Math Library function
   expects left-justified addresses.
*/
   k = MOD((int)(&i)<<16,(int) (&j)<<16);
   printf (" The Mod of 83 and 8 is: %d", k);
   exit (0);
}
```

Output:

```
The Mod of 83 and 8 is:
3
```

Example of MOD Called From FORTRAN

Source Code:

```
      PROGRAM MOD_EXAMPLE
C
      INTEGER i1, i2
      i1=83
      i2=8
      PRINT *, 'The MOD of i1 and i2 is:'
      PRINT *, MOD(i1,i2)
```

Output:

```
The MOD of i1 and i2 is:
3
```

NINT

NINT finds the nearest integer to an argument. It accepts a real argument and returns an integer result.

The call-by-reference entry points are MLP$RNINT and NINT, and the call-by-value entry point is MLP$VNINT.

The input domain for this function is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it, or if the argument is < 0, $-.5$ is added to it. This sum is converted from floating-point to integer and returned.

Example of NINT Called From FORTRAN

Source Code:

```
      PROGRAM NINT_EXAMPLE
C
      EXTERNAL NINT
      INTEGER i1,i2
      REAL x,y
      x=100.1234
      y=12.12
      i1=NINT(x)
      i2=NINT(y)
      PRINT *, 'The nearest integers to x and y are:'
      PRINT *, NINT(x)
      PRINT *, NINT(y)
      END
```

Output:

```
The nearest integers to x and y are:
100
12
```

RANF

RANF generates the next random number in a series of random numbers. It accepts a dummy argument and returns a real result.

The call-by-reference entry points are MLP$RRANF and RANF, and the call-by-value entry point is MLP$VRANF.

There is no input domain to this function. The output range is included in the set of positive real quantities less than 1.0.

Call-By-Reference Routine

No errors are generated in RANF. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

RANF uses the multiplicative congruential method modulo 2**48. The formula is:

```
x(n + 1.0) = a*x(n) (mod 2**48)
```

The library holds a random seed (mlv$initial_seed) and a multiplier (mlv$random_multiplier). The random seed can be changed to any valid seed value prior to calling RANF by use of the function RANSET (described later in this chapter). Upon entry at RANF, the random seed is multiplied in double precision by mlv$random_multiplier to generate a 96-bit product, which is the new seed partially normalized by one bit. This result is then denormalized. The lower 48 bits are formed with an exponent that yields a result between 0 and 1.0 to become the new random seed (mlv$random_seed). The current seed for the task is updated with the newly formed unnormalized seed. The seed is used to generate subsequent random numbers. The default initial value of mlv$initial_seed is 40002BC68CFE166D hexadecimal. The new random seed is normalized and returned as the random number.

The multiplier (mlv$random_multiplier) is constant and has a value of 40302875A2E7B175 hexadecimal. This multiplier passes the Coveyou-MacPherson test, the auto-correlation test with lag ≤ 100, the pair triplet test, and other statistical tests for randomness.[6]

6. Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota.

Example of RANF Called From Ada

Source Code:

```
package RANDOM_LIBRARY is

function RANF return FLOAT;
pragma INTERFACE (MATH_LIBRARY, RANF);

procedure RANGET (RESULT : in out FLOAT);
pragma INTERFACE (MATH_LIBRARY, RANGET);

procedure RANSET (VALUE : in out FLOAT);
pragma INTERFACE (MATH_LIBRARY, RANSET);

end RANDOM_LIBRARY;

with RANDOM_LIBRARY; use RANDOM_LIBRARY;
with TEXT_IO; use TEXT_IO;

procedure RANDOM is

    x1 : FLOAT;
    x2 : FLOAT;

    package FLT_IO is new FLOAT_IO (FLOAT);
    use FLT_IO;

begin

    PUT_LINE ("Begin");
    x1 := 0.7777;
    PUT ("Call RANSET with : "); PUT (x1); NEW_LINE;
    RANSET (x1);
    RANGET (x2);
    PUT ("RANGET returned : "); PUT (x2); NEW_LINE;
    x1 := RANF;
    x2 := RANF;
    PUT ("RANF returned : "); PUT (x1); NEW_LINE;
    PUT ("RANF returned : "); PUT (x2); NEW_LINE;
    PUT_LINE ("End");

end RANDOM;
```

Output:

```
Begin
Call RANSET with :  7.777000000000E01
RANGET returned :  7.777000000000E01
RANF returned :  8.022426980171E-01
RANF returned :  5.003749989168E-02
End
```

Example of RANF Called From C

Source Code:

```
/*  This C program uses the RANF function to compute 10 random numbers
    between 0 and 1.
*/

#define MAX 10

main()
{

  int count = 0;     /* loop counter                             */

  int random_number; /* Random number generated by RANF. */

  int ran_num_add;

  for (count=0; count < MAX; ++count)
  {


     random_number = RANF();

     printf("Random number %d is %f.\n", count, random_number);


   }
```

Output:

```
Random number 0 is 0.580114.
Random number 1 is 0.950513.
Random number 2 is 0.786371.
Random number 3 is 0.297620.
Random number 4 is 0.453700.
Random number 5 is 0.006262.
Random number 6 is 0.275736.
Random number 7 is 0.305651.
Random number 8 is 0.689101.
Random number 9 is 0.382662.

-- Program exit code value was 10.
```

RANGET

RANGET is a callable program procedure that returns the current random number seed of a task. It accepts a real argument.

The call-by-reference entry points are MLP$RRANGET and RANGET. There is no call-by-value routine for RANGET.

The result is returned through parameter n and is a positive real quantity in the interval (0,1.0).

Call-By-Reference Routine

RANGET returns the current seed, between 0 and 1, of the random number generator. The value returned might not be normalized. This seed can be used to restart the random sequence at exactly the same point. The current seed is mlv$random_seed.

Call-By-Value Routine

There are no call-by-value entry points for RANGET.

Example of RANGET

See the example Ada program in the RANF description in this chapter for an example of a RANGET call.

RANSET

RANSET is a callable program procedure that sets the seed of the random number generator. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RRANSET and RANSET. There is no call-by-value routine.

The input domain for this procedure is the collection of all possible full word bit patterns. There is no output.

Call-By-Reference Routine

RANSET uses the value passed to it to form a valid seed for the random number generator. If the argument is zero, the seed is set to its initial value (mlv$initial_seed) at load time. Otherwise, the value passed has its exponent set to 4000 hexadecimal, and the coefficient is made odd. This value is then saved and becomes the new seed (mlv$random_seed) for the task.

Example of RANSET

See the example Ada program in the RANF description in this chapter for an example of a RANSET call.

SIGN

SIGN transfers the sign from one argument to another argument. It accepts two real arguments and returns a real result. The result is a copy of the first argument with the sign of the second argument.

The call-by-reference entry points are MLP$RSIGN and SIGN, and the call-by-value entry point is MLP$VSIGN.

The input domain for this function is the collection of all valid real quantities. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

No errors are generated by SIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the second argument is inserted into the sign bit of the first argument.

Example of SIGN Called From FORTRAN

Source Code:

```
      PROGRAM SIGN_EXAMPLE
C
      EXTERNAL SIGN
      REAL x, y
      x=-180.0
      y=90.0
      PRINT *, 'The SIGN of x, y is:'
      PRINT *, SIGN(x,y)
      END
```

Output:

```
The SIGN of x, y is:
180.
```

SIN

SIN computes the sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSIN and SIN, the call-by-value entry point is MLP$VSIN, and the vector entry point is MLP$SINV.

The input domain for this function is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

See the description of function COS.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function SIN was tested against 3*SIN(x/3) - 4*SIN(x/3)**3. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-22 shows a summary of these statistics.

Table 8-22. Relative Error of SIN

Interval From	Interval To	Maximum	Root Mean Square
0.0000E+00	.1571E+01	.8305E-14	.2874E-14
.1885E+02	.2042E+02	.1355E-13	.3168E-14

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e*cos(x) for sin(x) and −e*sin(x) for cos(x).

Example of SIN Called From FORTRAN

Source Code:

```
      PROGRAM SIN_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The SIN of x is:'
      PRINT *, SIN(x)
      END
```

Output:

```
The SIN of x is:
.4794255386042
```

SIND

SIND computes the sine function of an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSIND and SIND, the call-by-value entry point is MLP$VSIND, and the vector entry point is MLP$SINDV.

The input domain for this function is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [−1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [−45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x ± 360 degrees) =  sin(x)
sin(x ± 180 degrees) = -sin(x)
sin(x +  90 degrees) =  cos(x)
sin(x -  90 degrees) = -cos(x)
cos(x ± 360 degrees) =  cos(x)
cos(x ± 180 degrees) = -cos(x)
cos(x +  90 degrees) = -sin(x)
cos(x -  90 degrees) =  sin(x)
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The reduction to (−45,+45) is exact; the constant pi/180 has relative error 1.37E−15, and multiplication by this constant has a relative error 5.33E−15, and a total error of 6.7E−15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is at most 5.26E−15 plus the maximum error in SINCOS over (−pi/4,+pi/4). The maximum relative error observed for a group of 10,000 arguments chosen randomly in the interval [0,360] was .1403E−13 for SIND and .7105E−14 for COSD.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (−45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

Example of SIND Called From FORTRAN

Source Code:

```
      PROGRAM SIND_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The SIND of x is:'
      PRINT *, SIND(x)
      END
```

Output:

```
The SIND of x is:
.008726535498374
```

SINH

SINH computes the hyperbolic sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSINH and SINH, the call-by-value entry point is MLP$VSINH, and the vector entry point is MLP$SINHV.

The input domain for this function is the collection of all valid real quantities whose absolute value is less than 4095*log(2). The output range is included in the set of all valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formulas used to compute sinh(x) are:

```
      x  = n*log(2) + a, where |a| ≤ 1/2*log(2)
           and n is an integer
sinh(x) = (cosh(a) + sinh(a))*2**(n-1), when n > 25
sinh(x) = sinh(a), when n = 0, otherwise,
sinh(x) = (c - s)*2**(n-1) + (c + s)*2**(-n-1)
```

where:

```
s = sinh(a) = a + s(3)*a**3*(s(5) + TOP/(BOT - a**2))
c = cosh(a) = 1.0 + a**2*(.5 + a**2*(c(4) + a**2*(c(6) +
              c(10)*a**2*(c(8) + a**2))))
```

Constants used in the algorithm are:

```
s(3)  =    .166 666 666 666 935 58
s(5)  =  -.005 972 995 665 652 368
TOP   =   1.031 539 921 161
BOT   = 72.103 746 707 22
c(4)  =    .041 666 666 666 488 081
c(6)  =    .001 388 888 895 231 804 5
c(8)  = 89.754 738 973 150 22
c(10) =  2.763 250 805 803*10**-7
```

The algorithm used is:

```
a.  u = |x|

b.  n = (u/log(2) + .5) = nearest integer to u/log(2) R
    w = u - n*log(2), where the right-hand expression is evaluated in double
    precision

c.  s = w + w**3(s(3) + w**2(s(5) + TOP/(BOT - w**2)))
    d = w**2(1/2 + w**2(c(4) + w**2(c(6) + w**2(c(8) + w**2)*c(10))))
    a = (1.0 + d - s)*2**(-n-1)
    b = d + s

d.  c  = (1/4 + (1/4 + b))*2**(n-1) + (2**(n-3) + (2**(n-3) - a))
    XF = c with the sign of x

e.  Return
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-23 shows a summary of these statistics.

Table 8-23. Relative Error of SINH

Test	Interval From	Interval To	Maximum	Root Mean Square
SINH(x) against Taylor series expansion of SINH(x)	0.0000E+00	.5000E+00	.3374E−13	.9969E−14
SINH(x) against c*(SINH (x + 1) + SINH(x − 1))	.3000E+01	.2838E+04	.2894E−13	.9979E−14

Effect of Argument Error

If a small error e occurs in the argument x, the resulting error in sinh(x) is given approximately by cosh(x)*e.

Example of SINH Called From FORTRAN

Source Code:

```
      PROGRAM SINH_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The SINH of x is:'
      PRINT *, SINH(x)
      END
```

Output:

```
The SINH of x is:
.5210953054938
```

SQRT

SQRT computes the square root function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSQRT and SQRT, the call-by-value entry point is MLP$VSQRT, and the vector entry point is MLP$SQRTV.

The input domain for this function is the collection of all valid, nonnegative real quantities. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [0.5, 2) and n an integer such that x = y*2**(2*n). Then SQRT(x) is evaluated by:

```
SQRT(x) = SQRT(y)*2**n
```

Then SQRT(y) is approximated to 48 bits of precision by applying one iteration of Heron's rule to an initial approximation which is accurate to at least 24 bits of precision. The initial approximation is computed by dividing the interval [0.5, 2) into the following 64 subintervals:

```
[32/64, 33/64)
  ⋮
[63/64, 64/64)
[32/32, 33/32)
  ⋮
[63/32, 64/32)
```

The coefficients of these 64 minimax approximations are stored in three tables p0, p1, and p2 such that:

```
z1 = p0[i] + p1[i]*y + p2[i]*y**2
```

is the quadratic minimax approximation to the square root of y over the subinterval whose index is i. The required initial approximation is obtained by calculating the index i of the subinterval that contains y and then evaluating the above quadratic polynomial so that z1 approximates SQRT(y) to at least 24 bits of precision.

Using Heron's rule, the computation:

```
twoz2 = z1 + y/z1
```

approximates SQRT(y) to 48 bits precision followed by the computation:

```
SQRT(x) = twoz2*2**(n - 1)
```

which approximates SQRT(x) to 48 bits of precision.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function SQRT was tested in the form SQRT(x*x) − x. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-24 shows a summary of these statistics.

Table 8-24. Relative Error of SQRT

Interval From	Interval To	Maximum	Root Mean Square
.1000E+01	.1414E+01	.7099E−14	.5677E−14
.7071E+00	.1000E+01	.5023E−14	.4106E−14

Effect of Argument Error

For a small error e in the argument y, the amplification of absolute error is e/(2*sqrt(y)).

Example of SQRT Called From FORTRAN

Source Code:

```
      PROGRAM SQRT_EXAMPLE
C
      REAL x, xe
      x=22500.0
      xe=SQRT(x*x)- x
      PRINT *, 'The SQRT of x is:'
      PRINT *, SQRT(x)
      PRINT *, 'The calculated error of the SQRT of x is:'
      PRINT *, xe
      END
```

Output:

```
The SQRT of x is:
150.
The calculated error of the SQRT of x is:
0.
```

SUM1S

SUM1S returns the sum (or number) of 1 bits in a word. (The number of bits in a NOS/VE word is always 64.) It accepts any type of argument except character and logical and returns an integer result. If the argument is of type double precision or complex, only the first word is used.

The call-by-reference entry points are MLP$RSUM1S and SUM1S, and the call-by-value entry point is MLP$VSUM1S.

The input domain for this function is the collection of all valid boolean, real, complex, integer, or double precision quantities. Character and logical arguments are not allowed. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

No errors are generated by SUM1S. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The number of bits in a word is returned. The argument can be any type except character and logical.

Example of SUM1S Called From FORTRAN

Source Code:

```
      PROGRAM SUM1S_EXAMPLE
C
      REAL x
      x=Z"4321FEDCBA987654"
      PRINT *, 'The SUM1S of x is:'
      PRINT *, SUM1S(x)
      END
```

Output:

```
The SUM1S of x is:
33
```

TAN

TAN computes the trigonometric circular tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTAN and TAN, the call-by-value entry point is MLP$VTAN, and the vector entry point is MLP$TANV.

The input domain for this function is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The evaluation is reduced to the interval [-.5,.5] by using the identities:

```
1.  tan(x) = tan(x + k*pi/2), if k is even

2.  tan(x) = -1.0/tan(x + pi/2)
```

in the form:

```
3.  tan(x) = tan((pi/2)*(x*2/pi + k)), if k is even

4.  tan(x) = -1.0/tan((pi/2)*(x*2/pi + 1.0))
```

An approximation of tan(pi/2*y) is used. The argument is reduced to the interval [-.5,.5] by subtracting a multiple of pi/2 from x in double precision.

The rational form is used to compute the tangent of the reduced value. The function tan((pi/2)*y) is approximated with a rational form (7th order odd)/(6th order even), which has minimax relative error in the interval [-.5,.5]. The rational form is normalized to make the last numerator coefficient 1 + e, where e is chosen to minimize rounding error in the leading coefficients.

Identity 4 is used if the integer subtracted is odd. The result is negated and inverted by dividing -Q/P instead of P/Q.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The range reduction, the final add in each part of the rational form, the final multiply in P, and the divide dominate the error. Each of these operations contributes directly to the final error, and each is accurate to about 1/2 ulp.

The function TAN was tested against 2*TAN(x/2)/(1 − TAN(x/2)**2). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-25 shows a summary of these statistics.

Table 8-25. Relative Error of TAN

Interval From	Interval To	Maximum	Root Mean Square
0.0000E+00	.7854E+00	.2177E−13	.5613E−14
.1885E+02	.1963E+02	.1993E−13	.5617E−14
.2749E+01	.3534E+01	.2190E−13	.7286E−14

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is sec(x)**2, and that of relative error is x/(sin(x)*cos(x)), which is at least 2x and can be arbitrarily large near a multiple of pi/2.

Example of TAN Called From FORTRAN

Source Code:

```
      PROGRAM TAN_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The TAN of x is:'
      PRINT *, TAN(x)
      END
```

Output:

```
The TAN of x is:
.54630224898438
```

TAND

TAND computes the trigonometric tangent for an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTAND and TAND, the call-by-value entry point is MLP$VTAND, and the vector entry point is MLP$TANDV.

The input domain for this function is the collection of all valid real arguments whose absolute value is less than 2**47, excluding odd multiples of 90. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [−45,45] by finding the nearest integer n to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The routine calls TAN to compute the tangent, and if the multiple n of 90 is odd, the result is negated and inverted by using the identities:

```
tan(x ± 180 degrees) = tan(x)
tan(x ± 90 degrees)  = -1/tan(x)
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The reduction to (−45,+45) is exact; the constant pi/180 has a relative error of 1.37E−15, and multiplication by this constant has a relative error of 5.33E−15, so the total error is 6.7E−15. The maximum relative error observed for 10,000 arguments chosen randomly in the interval [0,360], was .2130E−13.

Effect of Argument Error

Errors in the argument x are amplified at most by x/(sin(x)*cos(x)). This function has a maximum of pi/2 within (−45,+45) but has poles at all multiples of 90 degrees except zero.

Example of TAND Called From FORTRAN

Source Code:

```
      PROGRAM TAND_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The TAND of x is:'
      PRINT *, TAND(x)
      END
```

Output:

```
The TAND of x is:
.008726867790759
```

TANH

TANH computes the hyperbolic tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTANH and TANH, the call-by-value entry point is MLP$VTANH, and the vector entry point is MLP$TANHV.

The input domain for this function is the collection of all valid real quantities. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument range is reduced to:

```
tanh(x) = 1.0 - 2*(q - p)/((q - p) + 2**n*(q + p))
```

by the identities:

```
tanh(-x) = -tanh(x) for x < 0
tanh(x)  = p(x)/q(x) approximately, in the interval [0,.55]
tanh(x)  = 1.0 - 2/(exp(2*x) + 1.0)
exp(2*x) = (1.0 + tanh(x))/(1.0 - tanh(x))
exp(2*x) = 2**n*exp(2*(x - n*ln(2)/2))
```

where n is chosen to be nint(x*2/ln(2)) and p and q are evaluated on x − n*ln(2)/2. This choice of n minimizes abs(x − n*ln(2)/2).

When abs(x) ≤ .55 = atanh(.5), the approximation p(x)/q(x) is used. When abs(x) > .55, the above range reduction is used. For abs(x) > 17.1, tanh(x) = sign(1.0,x).

The approximation p/q is a minimax (relative error) rational form (5th order odd)/(6th order even). The range reduction is simplified by scaling the coefficients so that (x*2/ln(2) − n) can be used instead of (x − n*ln(2)/2). The coefficients are further scaled by an amount sufficient to reduce truncation error in the leading coefficients without otherwise affecting accuracy.

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The algorithm error due to finite approximation and coefficient truncation is 1.7E−15. For abs(x) < .55, the form p(x)/q(x) is used. The final operations z = x*2/ln(2) and tanh(z*(p0+small))/(q0+small) dominate the error. For abs(x) > 1.25 the final subtraction (1.0 − small) dominates.

For .55 ≤ abs(x) ≤ 1.25, the final operation is 1−R, where R becomes smaller as x approaches 1.25. Thus, the worst relative error is near .55, namely, (contribution from R) + (error in final sum), where R = 2*(q − p)/((q − p) + 4*(q + p)).

The function TANH was tested against (TANH(x − 1/8) + TANH(1/8))/(1 + TANH(x − 1/8)*TANH(1/8)). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-26 shows a summary of these statistics.

Table 8-26. Relative Error of TANH

Interval From	Interval To	Maximum	Root Mean Square
.1250E+00	.5493E+00	.4091E−13	.1085E−13
.6743E+00	.1768E+02	.2842E−13	.3730E−14

Effect of Argument Error

For small errors in the argument x, the amplification of the absolute error is 1/cosh**(x) and of relative error is x/(sinh(x)*cosh(x)). Both have maximum values of 1.0 at zero and approach zero as x gets large.

Example of TANH Called From FORTRAN

Source Code:

```
      PROGRAM TANH_EXAMPLE
C
      REAL x
      x=0.5
      PRINT *, 'The TANH of x is:'
      PRINT *, TANH(x)
      END
```

Output:

```
The TANH of x is:
.46211715726
```

XTOD

XTOD performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. XTOD also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RXTOD and XTOD, and the call-by-value entry point is MLP$VXTOD.

The XTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOD(scalar,vector) = MLP$XTODV
XTOD(vector,scalar) = MLP$XVTOD
XTOD(vector,vector) = MLP$XVTODV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a nonnegative real quantity and y is a double precision quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result if valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the argument x is converted to double precision, and all operations are carried out in double precision. The routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

See the description of function DTOD.

Effect of Argument Error

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

```
r*(e(p)*log(b) + p*e(b)/b)
```

Example of XTOD Called From FORTRAN

Source Code:

```
      PROGRAM XTOD_EXAMPLE
C
      REAL x
      DOUBLE PRECISION y, z, XTOD
      x=20.0
      y=140.0d0
      z=XTOD(x,y)
      PRINT *, 'The XTOD of x and y is:'
      PRINT *, z
      END
```

Output:

```
The XTOD of x and y is:
1.3937965749081639463459823 8E+182
```

XTOI

XTOI performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. XTOI also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RXTOI and XTOI, and the call-by-value entry point is MLP$VXTOI.

The XTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOI(scalar,vector) = MLP$XTOIV
XTOI(vector,scalar) = MLP$XVTOI
XTOI(vector,vector) = MLP$XVTOIV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a real quantity and y is an integer quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The arguments are checked to see whether the exponentiation conforms to a special case. If it does, the proper value is immediately returned. If the special case is an error condition, an error message is displayed. The special cases are:

```
x indefinite = error
x infinite   = error
0**0         = error
x**i         = 1.0 if i = 0 and x > 0
x**i         = 1.0/x**-i if i < 0
x = 0        = error if i < 0
```

If the exponentiation is not a special case, the following algorithm is used.

Starting with the second most significant bit, the binary representation of i is scanned from left to right. The result is initialized to x. For each scanned bit, the result is squared. If the scanned bit is 1, the result is multiplied by x.

Effect of Argument Error

If a small error e occurs in the base b, the error in the result will be given approximately by n*b**(n−1)*e, where n is the exponent (integer argument of the function).

Example of XTOI Called From FORTRAN

Source Code:

```
      PROGRAM XTOI_EXAMPLE
C
      INTEGER i
      REAL x, XTOI
      i=3
      x=10.0
      PRINT *, 'The XTOI of x and i is:'
      PRINT *, XTOI(x,i)
      END
```

Output:

```
The XTOI of x and i is:
1000.
```

XTOX

XTOX performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. XTOX also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RXTOX and XTOX, and the call-by-value entry point is MLP$VXTOX.

The XTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOX(scalar,vector) = MLP$XTOXV
XTOX(vector,scalar) = MLP$XVTOX
XTOX(vector,vector) = MLP$XVTOXV
```

The input domain for this function is the collection of all valid real pairs (x,y), where x is a nonnegative quantity and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the routine calls ALOG to compute log(x), and EXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

The function XTOX was tested. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 8-27 shows a summary of these statistics.

Table 8-27. Relative Error of XTOX

Test	Interval From	Interval To	Maximum	Root Mean Square
	x interval			
x**y against x**2**(y/2)	.1000E−01	.1000E+02	.3547E−12	.6352E−13
	y interval			
	−.6167E+03	.6167E+03		
x**2**1.5 against	.1000E+01	.8053+411	.1360E−13	.5687E−14
x**2*x	.5000E+00	.1000E+01	.1360E−13	.5715E−14
x**1.0 against x	.5000E+00	.1000E+01	.6802E−14	.3442E−14

Effect of Argument Error

If a small error e(b) occurs in the base b, and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

```
r*(log(b)*e**p + p*(e(b))/b)
```

Example of XTOX Called From FORTRAN

Source Code:

```
      PROGRAM XTOX_EXAMPLE
C
      REAL x, y, XTOX
      x=2.0
      y=10.0
      PRINT *, 'The XTOX of x and y is:'
      PRINT *, XTOX(x,y)
      END
```

Output:

```
The XTOX of x and y is:
1024.
```

XTOZ

XTOZ performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. XTOZ also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RXTOZ and XTOZ, and the call-by-value entry point is MLP$VXTOZ.

The XTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOZ(scalar,vector) = MLP$XTOZV
XTOZ(vector,scalar) = MLP$XVTOZ
XTOZ(vector,vector) = MLP$XVTOZV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a real quantity, y is a complex quantity, and x**y is a valid quantity. If x is zero, the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y does not equal zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the real argument x is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(x) occurs in the base x, and a small error e(z) (e(x) + i*e(y)) occurs in the exponent z, the error in the result w is given approximately by:

```
w*(log(x)* e(z) + z*e(x)/x)
```

Example of XTOZ Called From FORTRAN

Source Code:

```
      PROGRAM XTOZ_EXAMPLE
C
      REAL x
      COMPLEX zeta, omega, xtoz
      x = 5.0
      zeta = (5.0, 0)
      omega = XTOZ (x, zeta)
      PRINT *, 'The XTOZ of x and zeta is:'
      PRINT *, omega
      END
```

Output:

```
The XTOZ of x and zeta is:
(3125.,0.)
```

ZTOD

ZTOD performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ZTOD also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RZTOD and ZTOD, and the call-by-value entry point is MLP$VZTOD.

The ZTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOD(scalar,vector) = MLP$ZTODV
ZTOD(vector,scalar) = MLP$ZVTOD
ZTOD(vector,vector) = MLP$ZVTODV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a complex quantity, y is a double precision quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the double precision argument y is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(z) occurs in the base z and a small error e(e) occurs in the exponent e, the error in the result w is given approximately by:

```
w*(e(e)*log(z) + e*e(z)/z)
```

Example of ZTOD Called From FORTRAN

Source Code:

```
      PROGRAM ZTOD_EXAMPLE
C
      COMPLEX zeta, omega, ztod
      DOUBLE PRECISION y
      zeta = (5.0, -1)
      y=140.0d0
      omega = ZTOD(zeta,y)
      PRINT *, 'The ZTOD of zeta and y is:'
      PRINT *, omega
      END
```

Output:

```
The ZTOD of zeta and y is:
(-8.96804850841 4E+98,-6.662556718066E+98)
```

ZTOI

ZTOI performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ZTOI also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RZTOI and ZTOI, and the call-by-value entry point is MLP$VZTOI.

The ZTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOI(scalar,vector) = MLP$ZTOIV
ZTOI(vector,scalar) = MLP$ZVTOI
ZTOI(vector,vector) = MLP$ZVTOIV
```

The input domain for this function is the collection of all valid pairs (x,y), where x is a complex quantity, y is a integer quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Let x represent the base and y represent the exponent. If y has binary representation 000... .000i(n)i(n−1) ...i(1)i(0), where each i(j)(0 $\leq$ j $\leq$ n) is 0 or 1, then:

```
y = i(0)*2**0 + i(1)*2**1 + ... + i(n)*2**n
n = (log(2)y) = greatest integer not exceeding log(2)y
```

Then:

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1]
```

The numbers x**0, x = x**2**0, x**2, x**4, ..., x**(2)**n are generated during the computation by successive squarings, and the coefficients i(0), ..., i(n) are obtained as sign bits of successive circular right shifts of y within the computer. A running product is formed during the computation so that smaller powers of x can be discarded. The computation then becomes an iteration of the algorithm:

```
x**y = 1, if y = 0 and x is not= 0
     = (x*x)**(y/2), if y > 0 and y is even
     = (x*x)**((y-1)/2)*x, if y > 0 and y is odd
```

Upon entry, if the exponent y is negative, y is replaced by −y and a sign flag is set. x**y is computed according to this algorithm, and if the sign flag was set, the result is reciprocated before being returned to the calling program.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Effect of Argument Error

If a small error e occurs in the base b, the error in the result will be given approximately by n*b**(n−1)*e, where n is the exponent given to the routine.

Example of ZTOI Called From FORTRAN

Source Code:

```
      PROGRAM ZTOI_EXAMPLE
C
      INTEGER i
      COMPLEX zeta, omega, ztoi
      i = 12
      zeta = (2.0, -1)
      omega= ZTOI (zeta,i)
      PRINT *, 'The ZTOI of zeta and i is:'
      PRINT *, omega
      END
```

Output:

```
The ZTOI of zeta and i is:
(11753.,10296.)
```

ZTOX

ZTOX performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ZTOX also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RZTOX and ZTOX, and the call-by-value entry point is MLP$VZTOX.

The ZTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOX(scalar,vector) = MLP$ZTOXV
ZTOX(vector,scalar) = MLP$ZVTOX
ZTOX(vector,vector) = MLP$ZVTOXV
```

The input domain for this function is the collection of all valid argument pairs (x,y), where x is a complex quantity, y is a real quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the real argument is converted to a complex argument, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(z1) occurs in the base z1 and a small error e(z2) occurs in the exponent z2, the error in the result w is given approximately by:

```
w*(e(z2)*log(z1) + z2*e(z1)/z1)
```

Example of ZTOX Called From FORTRAN

Source Code:

```
      PROGRAM ZTOX_EXAMPLE
C
      REAL x
      COMPLEX zeta, omega, ztox
      x = 12.0
      zeta = (2.0, -1)
      omega= ZTOX (zeta,x)
      PRINT *, 'The ZTOX of zeta and x is:'
      PRINT *, ZTOX (zeta,x)
      PRINT *, omega
      END
```

Output:

```
The ZTOX of zeta and x is:
(11753.,10296.)
```

ZTOZ

ZTOZ performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result. ZTOZ also accepts compiler-generated calls (for example, the FORTRAN and Ada compilers provide the exponentiation operator **).

The call-by-reference entry points are MLP$RZTOZ and ZTOZ, and the call-by-value entry point is MLP$VZTOZ.

The ZTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOZ(scalar,vector) = MLP$ZTOZV
ZTOZ(vector,scalar) = MLP$ZVTOZ
ZTOZ(vector,vector) = MLP$ZVTOZV
```

The input domain is the collection of all valid complex pairs (x,y). If the real and imaginary parts of x are equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, and the imaginary part of y does not equal zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0.
```

Upon entry, argument checking is performed. If the arguments are valid, the routine calls CLOG to compute log(x), and CEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 7, Vector Processing, for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval ([−1.0,1.0],[−1.0,1.0]) and ([−1.0,1.0],[−1.0,1.0]). The maximum relative error of these arguments was found to be 1.7431E−11.

Effect of Argument Error

If a small error e(z1) occurs in the base z1 and a small error e(z2) occurs in the exponent z2, the error in the result w is given approximately by:

```
w*(e(z2)*log(z1) + z2*e(z1)/z1)
```

Example of ZTOZ Called From FORTRAN

Source Code:

```
      PROGRAM ZTOZ_EXAMPLE
C
      COMPLEX alpha, zeta, omega, ztoz
      alpha = (12.0, 0)
      zeta = (2.0, -1)
      omega= ZTOZ (alpha, zeta)
      PRINT *, 'The ZTOZ of alpha and zeta is:'
      PRINT *, omega
      END
```

Output:

```
The ZTOZ of alpha and zeta is:
(-114.0508449541,-87.91134605528)
```

Auxiliary Routines 9

Auxiliary Routines 9

The auxiliary routines cannot be called by their Math Library names. As the following list indicates, these routines are algorithmic modules that are called by Math Library functions:

- ACOSIN (called by ACOS and ASIN)
- COSSIN (called by CSIN and CCOS)
- DASNCS (called by DCOS and DSIN)
- DEULER (called by DEXP and DTANH)
- DSNCOS (called by DCOS and DSIN)
- HYPERB (called by COSH and SINH)
- SINCOS (called by SIN and COS)
- SINCSD (called by SIND and COSD)

Most of these routines can be called by their call-by-value entry points from assembler programs, but this is not recommended. These routines are described in this manual for algorithmic and error analysis.

ACOSIN

ACOSIN is an auxiliary routine that computes the inverse sine or inverse cosine function. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for ACOSIN. The call-by-value entry points are MLP$VACOS and MLP$VASIN.

The input domain is the collection of all valid real quantities in the interval [−1.0,1.0]. The output range is included in the set of valid, nonnegative real quantities less than or equal to pi.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),   x < -.5
arcos(x)  = pi - arcos(-x),   x < -.5
arcsin(x) = x + x**3*s*((w + z - j)*w + a + m/(e - x**2)),
  where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),   -.5 ≤ x < .5
arcsin(x) = pi/2 - arcos(x),   .5 ≤ x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,   .5 ≤ x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
        w  = (x**2 - c)*z + k
        z  = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =    3.173 170 078 537 13
e =    1.160 394 629 739 02
m =   50.319 055 960 798 3
c =   -2.369 588 855 612 88
i =    8.226 467 970 799 17
j =  -35.629 481 597 455 5
k =   37.459 230 925 758 2
a =  349.319 357 025 144
s =     .746 926 199 335 419*10**-3
```

The approximation of arcsin (−.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

```
a.  If ACOS entry, go to step g.

b.  If |x | ≥ .5, go to step h.

c.  n = 0  (Loop counter).
    q = x
    y = x**2
    u = 0, if ASIN entry.
    u = pi/2, if ACOS entry.

d.  z  = (y + r)*y + i
    w  = (y - c)*z + k
    p  = q + s*q*y*((w + z - j)*w + a + m/(e - y))
    p  = u - p
    Y1 = p/2**n

e.  If ASIN entry, go to step k.

f.  If x is in (-.5,1.0), return.
    XF = 2*u - (Y1)
    Return.

g.  If |x | < .5, go to step c.

h.  If x = 1.0 or -1.0, go to step l.
    If x is invalid, go to step m.
        n = 0 (Loop counter).
        y = 1.0 - |x|, and normalize y.

i.  h = 4*y - 2*y**2
    n = n + 1.0
    If 2*y ≤ 2 - sqrt(3) = .267949192431, y = h, and go to step i.

j.  q = 1.0 - h, and normalize q.
    y = q**2
    u = pi/2
    Go to step d.

k.  Y1 = u - (Y1), and normalize Y1.
    Affix sign of x to Y1 = XF.
    Return.

l.  XF = pi/2, if x = 1.0.
    XF = -pi/2, if x = -1.0.
    If ASIN entry, return.
    XF = 0, if x = 1.0.
    XF = pi, if x = -1.0.
    Return.

m.  Return.
```

Error Analysis

See the descriptions of functions ACOS and ASIN.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e/(1.0 - x**2)**.5.

COSSIN

COSSIN is an auxiliary routine that accepts calls from other math functions that require simultaneous computation of the sine and cosine of the same argument. COSSIN accepts a real argument and returns two real results.

See the descriptions of functions CSIN and CCOS for additional information.

Call-By-Value Routine

The argument is reduced to the interval [−pi/4,pi/4]. Polynomials p(x) and q(x) of degrees 11 and 12 are used to compute sin(x) and cos(x) over that interval. Upon entry, the argument x is multiplied by 2/pi. Then, the nearest integer n to 2/pi*x is computed. The upper and lower halves of the result are added. The value of y is in the interval (−pi/4,pi/4). y = x − n*pi/2 is computed in double precision as the reduced argument for input to p(y) and q(y). Then sin(x) and cos(x) are computed from these as indicated by the value k = mod(n,4), where k = 0, 1, 2, 3. The formula used to compute sine(x) is:

```
sin(x) = sin(y + n*p/2) = sin(y + k*pi/2)
       = sin(y)*cos*(k*pi/2) + cos(y)*sin(k*pi/2)
```

A similar formula is used for the computation of cosine(x). Depending upon k, either the sine(k = 0,1) or cosine(k = 2,3) of y is evaluated and complemented, if necessary.

The polynomials p(x) and q(x) are:

```
p(x) = s(0)x + s(1) x**3 + s(2)x**5 + s(3)x**7 + s(4)x**9 +
       s(5)x**11

q(x) = c(0) + c(1)x**2 + c(2)x**4 + c(3)x**6 + c(4)x**8 +
       c(5)x**10 + c(6)x**12
```

where the coefficients are:

```
s(0) =  .999 999 999 999 972
s(1) = -.166 666 666 665 404
s(2) =  .833 333 331 696 029*10**-2
s(3) = -.198 426 073 537 90*10**-3
s(4) =  .275 548 564 509 884*10**-5
s(5) = -.247 320 720 952 463*10**-7
c(0) =  .999 999 999 999 996
c(1) = -.499 999 999 999 991
c(2) =  .041 666 666 666 470 5
c(3) = -.138 888 888 888 159*10**-2
c(4) =  .248 015 784 673 257*10**-4
c(5) = -.275 552 187 277 097*10**-6
c(6) =  .206 291 063 476 645*10**-8
```

The coefficients were obtained as follows. The polynomials of degrees 15 and 14, obtained by truncating the Maclaurin series[1] for sin(x) and cos(x), were telescoped to form the polynomials p(x) and q(x) of degrees 11 and 12. The telescoping is done by removing the leading term of the polynomial. This is accomplished by subtracting an appropriate multiple of T(n)(a(x - x(0))) of the same degree n; 2/a is the length of the interval of approximation, and x(0) is its center.

The Chebyshev polynomial of degree n, T(n)(x), is defined by T(n)(x) = cos(n*arccos(x)).[2] The absolute value of x is no greater than one and satisfies the recurrence relation:

```
    T(0)(x) = 1
    T(1)(x) = x
T(n + 1)(x) = 2xT(n)(x) - T(n - 1)(x)
```

where $n \geq 1$.

For $n \geq 1.0$, T(n)(x) is the unique polynomial 2(n - 1.0)*x**n + ... of degree n whose maximum absolute value over the interval [-1.0,1.0] is minimal. This maximum absolute value is one.

The formulas used for the range reduction are:

```
    sin(x) = (-1)**n*sin(y)
    cos(x) = (-1)**n*cos(y)
if x = y + n*pi, n an integer

    sin(x) = cos(x - pi/2)
    cos(x) = -sin(x - pi/2)
if pi/4 ≤ x ≤ pi/2
```

Error Analysis

The maximum absolute error in the approximation of sin(x) by p(x) in the interval (-pi/4,pi/4) is .1893E-14. The maximum absolute error in the approximation of cos(x) by q(x) is .3687E-14.

Effect of Argument Error

Not applicable, since this routine is not called directly by the user's program.

1. For a discussion of Maclaurin series, refer to any calculus text (for example, Calculus and Analytic Geometry by G. B. Thomas).

2. For a discussion of the Chebyshev polynomial, see any analysis text (for example, Introduction to Numerical Analysis by F. B. Hildebrand).

DASNCS

DASNCS is an auxiliary routine that computes the inverse sine or inverse cosine function. It accepts a double precision argument and returns a double precision result.

There are no call-by-reference entry points for DASNCS. The call-by-value entry points are MLP$VDACOS and MLP$VDASIN.

The input domain is the collection of all valid double precision quantities in the interval [−1.0,1.0]. The output range at entry point MLP$VDACOS is included in the set of valid, nonnegative double precision quantities less than or equal to pi. The output range at entry point MLP$VDASIN is included in the set of valid double precision quantities in the interval [−pi/2,pi/2].

Call-By-Value Routine

The following identities are used to move the interval of approximation to [0,sqrt(.5)]:

```
arcsin(-x) = -arcsin(x)
arccos(x)  =  pi/2-arcsin(x)
arcsin(x)  =  arccos(sqrt(1.0 - x**2)), if x ≥ 0
arccos(x)  =  arcsin(sqrt(1.0 - x**2)), if x ≥ 0
```

The reduced value is called y. If y ≤ .09375, no further reduction is performed. If not, the closest entry to y in a table of values (z,arcsin(z),sqrt(1.0 − z**2), z = .14, .39, .52, .64) is found, and the formula used is:

```
arcsin(x) = arcsin(z) + arcsin(w)
```

where w = x*sqrt(1.0 − z**2) − z*sqrt(1.0 − x**2). The value of w is in the open interval (−.0792,.0848).

The arcsin of the reduced argument is then found using a 15th order odd polynomial with quotient:

```
x + x**3(c(3) + x**2(c(5) + x**2(c(7) + x**2(c(11) + x**2(c(13) +
x**2(c(15) + a/(b-x**2)))))))
```

where all constants and arithmetic operations before c(11) are double precision and the rest are single precision. The addition of c(11) has the form single+single=double. The polynomial is derived from a minimax rational form (denominator is (b − x**2)) for which the critical points have been perturbed slightly to make c(11) fit in one word.

To this value, arcsin(z) is added from a table if the last reduction above was done and the sum is conditionally negated. Then 0, −pi/2, +pi/2, or pi is added to complete the unfolding.

Error Analysis

See the descriptions of functions DACOS and DASIN.

Effect of Argument Error

See the descriptions of functions DACOS and DASIN.

DEULER

DEULER is an auxiliary routine that accepts calls from other math functions. It performs computations that are common among these routines.

The input and output ranges are described in the DEXP and DTANH function descriptions.

Call-By-Value Routine

Constants used in the algorithm are:

```
1.0/log(2)
log(2)  (in double precision)
d3  =     .166 666 666 666 666 666 666 666 666 709
d5  =     .833 333 333 333 333 333 333 331 234 953*10**-2
d7  =     .198 412 698 412 698 412 700 466 386 658*10**-3
d9  =     .275 573 192 239 858 897 408 325 908 796*10**-5
pc  =   -.474 970 880 178 988*10**-10
pa  =     .566 228 284 957 811*10**-7
pb  = 272.110 632 903 710
c11 =     .250 521 083 854 439*10**-7
```

Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. An operand with a u or l subscript denotes the first or second word, respectively, of the double precision pair of words containing the operand.

The algorithm used is:

```
a.  n = nearest integer to x/log(2),
    y = x - n*log(2),
    Then y is in [-1/2*log(2),1/2*log(2)].

b.  q = (y)(u)*(u)(y)(u)

c.  p = q*(d)(d3 +(d) q*(d)(d5 +(d) q*(d)(d7 +(d) q*(d)(d9 +(d)
        q*(d)(c11 +(d) q*(d)(pa/(pb - q) + pc))))))

d.  s = (y)(u) + (d)(y)(u)*(d)p

e.  Compute hm = sqrt(1.0 + s**2).
    hi = 3*q + ((s)(u))**2 computed in single precision.
    hj = hi + hi
    hk = 2*(1.0 + hj)
    hl = ((y)(u)*(u)(y)(u) - hj)/hk - hi
    hm = hj + (u)(hk - (u)hl)*(u)(hl/hk)
         (hm now carries cosh-1.0 in single precision.)

f.  DS = s + (d)(((y)(l) + (r)(y)(l)*(u)hm) + (r)((s)(l) +
         (r)((y)(u)* (l)(p)(u) + (r)(y)(u)*(r)(p)(l))))
         (DS now contains sinh(y) in double precision.)

g.  DC = hm + (d)(DS*DS - 2*hm - hm*hm)/(2(1.0 + hm)) computed in double
    precision.

h.  DX = DS + DC

i.  Clean up DS, DC, DX with (X7) = n.
    Register pair XA-XB = DS = sinh(y).
    Register pair X8-X9 = DC = cosh(y) - 1.0.
    Register pair X4-X5 = DX = exp(y).

j.  Return.
```

Error Analysis

See the descriptions of functions DEXP and DTANH.

Effect of Argument Error

See the descriptions of functions DEXP and DTANH.

DSNCOS

DSNCOS is an auxiliary routine that computes the trigonometric sine or trigonometric cosine function. It accepts a double precision argument and returns a double precision result.

There are no call-by-reference entry points for DSNCOS. The call-by-value entry points are MLP$VDCOS and MLP$VDSIN.

The input domain for this routine is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [−1.0,1.0].

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x − n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [−pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [−pi/4,pi/4], the sine polynomial approximation is:

```
p(x) = a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 + a(11)x**11 +
       a(13)x**13** + a(15)x**15 + a(17)x**17 + a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) = b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 + b(10)x**10 +
       b(12)x**12 + b(14)x**14 + b(16)x**16 + b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =   .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =   .833 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =   .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =   .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =   .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =   .194 362 013 130 224*10**-19
b(0)  =   .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =   .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =   .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =   .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =   .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =   .408 023 947 777 860*10**-18
```

These polynomials are evaluated from right to left in double precision. The sign flag is used to give the result the correct sign before returning to the calling program.

Error Analysis

See the descriptions of functions DCOS and DSIN.

Effect of Argument Error

See the descriptions of functions DCOS and DSIN.

HYPERB

HYPERB is an auxiliary routine that accepts calls from other math functions that require the simultaneous hyperbolic sine and hyperbolic cosine of the same argument. HYPERB accepts a real argument and returns two real results.

The entry points and input and output ranges for this routine are described in in the CSIN and CCOS function descriptions.

Call-By-Value Routine

Upon entry, the routine computes e**x = exp(x), where x is the angle passed to HYPERB. The hyperbolic cosine is computed by:

```
cosh(x) = 0.5*(exp(x) + exp(-x))
```

If $|x| \geq .22$, the hyperbolic sine is computed by:

```
sinh(x) = 0.5*(exp(x) - exp(-x))
```

For $|x| < 0.22$, the Maclaurin series[3] for sinh is truncated after the term x**9/9! and the resulting polynomial is taken as the approximation:

```
sinh(x) = x + x**3/3! + x**5/5! + x**7/7! + x**9/9!
```

Error Analysis

See the descriptions of functions COSH and SINH.

Effect of Argument Error

See the descriptions of functions COSH and SINH.

3. For a discussion of Maclaurin series, refer to any calculus text (for example, Calculus and Analytic Geometry by G. B. Thomas).

SINCOS

SINCOS is an auxiliary routine that computes the trigonometric sine and cosine functions. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for SINCOS. The call-by-value entry points are MLP$VCOS and MLP$VSIN.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Value Routine

If x is valid, then COS(x) or SIN(x) is calculated by using the periodic properties of the cosine and sine functions to reduce the task to finding a cosine or sine of an equivalent angle y within [-pi/4, pi/4] as follows:

```
If N + K is even
then
     Z = sin(y)
else
     Z = cos(y)
If MOD(N + K, 4) is 0 or 1 (that is, the second last bit of N + K is even)
then
     S = 0
else
     S = mask(1)
```

where K is 0, 1, or 2 according to whether the SIN of a positive angle, the COS of any angle, or the SIN of a negative angle is to be calculated. N is the nearest integer to 2/pi*x, and y is the nearest single precision floating-point number to x – n*pi/2. The argument x is the absolute value of the angle. The desired SIN or COS is xor(S, Z).

Once the angle has been reduced to the range [-pi/4, pi/4], the following approximations are used to calculate either the cosine or the sine of the angle, providing 48 bits of precision.

If the cosine or the angle is required, the approximation used is

```
cosine(y) = 1 - y*y*P(y*y)
```

where y is the angle and P(w) is the quintic polynomial:

```
P(w) = P0 + P1*w  + P2*w**2 +P3 + w**3 + P4*w**4 + P5*w**5
```

such that P(y*y) is a minimax polynomial approximation to the function (1 – cos(y))/y**2.

The coefficients are:

```
P5 = -2.070062305624629462E-9
P4 =  2.755636997406588778E-7
P3 = -2.480158521206426671E-5
P2 =  1.388888888727866775E-3
P1 = -4.166666666666468116E-2
P0 =  5.000000000000000000E-1
```

If the sine of the angle is required, the approximation used is

```
sine(y) = y - y*y*y*Q(y*y)
```

where y is the angle and Q(w) is the quintic polynomial:

```
Q(w) = Q0 + Q1*w  + Q2*w**2 +Q3*w**3 + Q4*w**4 + Q5*w**5
```

such that Q(y*y) is a minimax polynomial approximation to the function (y − sin(y))/y**3.

The coefficients are:

```
Q5 = -1.5918142570330052831E-10
Q4 =  2.505113204973767698E-8
Q3 = -2.7557316103657547733E-6
Q2 =  1.98412698367610091 1E-4
Q1 = -8.3333333333330950363E-3
Q0 =  1.6666666666666664631E-1
```

Error Analysis

The function SINCOS was tested against 4*COS(x/3)**3 − 3*COS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199E+02,.2356E+02]. Statistics on relative error were observed: maximum relative error was .1404E−13, and root mean square relative error was .3245E−14.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given approximately by e*cos(x) for sin(x) and −e*sin(x) for cos(x).

SINCSD

SINCSD computes the sine and cosine functions for arguments in degrees. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for SINCSD. The call-by-value entry points are MLP$VCOSD and MLP$VSIND.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [−1.0,1.0].

Call-By-Value Routine

The result is put in the interval [−45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x ± 360 degrees) =  sin(x)
sin(x ± 180 degrees) = -sin(x)
sin(x +  90 degrees) =  cos(x)
sin(x -  90 degrees) = -cos(x)
cos(x ± 360 degrees) =  cos(x)
cos(x ± 180 degrees) = -cos(x)
cos(x +  90 degrees) = -sin(x)
cos(x -  90 degrees) =  sin(x)
```

Error Analysis

The reduction to (−45,+45) is exact; the constant pi/180 has relative error 1.37E−15, and multiplication by this constant has a relative error 5.33E−15, and a total error of 6.7E−15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is, at most, 5.26E-15 plus the maximum error in SINCOS over (−pi/4,+pi/4).

A group of 10,000 arguments was chosen at random from the interval [0,360]. The maximum relative error of these arguments was found to be .7105E−14 for COSD and .1403E−13 for SIND.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (−45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

Appendixes

Glossary A

A

Algorithm Error

Error caused by inaccuracies inherent in the mathematical process used to compute the result. It includes error in coefficients used in the algorithm.

Argument

A variable or constant that is passed to a routine and used by that routine to compute a function. The actual value of the variable is passed when a routine is called by value; the address of the variable is passed when the routine is called by reference.

Argument Set

An ordered list of one or more arguments.

Auxiliary Routine

A math routine which is not directly called from program code, but assists in the computation of a Math Library function.

C

Call-by-Address

See call-by-reference.

Call-by-Reference

A method of referencing a subprogram in which the addresses of the arguments are passed. Synonymous with call-by-address.

Call-by-Value

A method of referencing a subprogram in which the values of the arguments are passed.

D

Data Descriptor

Describes data by pointing to one or more contiguous data locations.

Domain

The collection of argument lists for which an entry point (function call) has been designed to return meaningful results without generating an error condition.

Dummy Argument

A variable or constant that is passed to a routine, but is not used by the routine to compute a function.

E

Entry Point

A statement within a math routine at which execution can begin. There may be more than one entry point into a math routine.

Error

The computed value of a function minus the true value.

Exponentiation Routine

A math routine which accepts compiler-generated calls from a source program to perform exponentiation. These calls are generated when a program statement involves exponentiation of certain number types. Exponentiation routines are not called directly using their function names.

External Routine

A predefined subprogram that accepts calls from program code to compute certain mathematical functions.

F

Function Name

A symbolic name that appears in a program and causes a math routine to be executed (for example, ABS).

I

Indefinite Value

A value that results from a mathematical operation that cannot be resolved, such as a division where the dividend and divisor are both zero. Indefinite numbers are nonstandard floating-point numbers with exponents in the range of 7000 hexadecimal to 7FFF hexadecimal or F000 hexadecimal to FFFF hexadecimal.

Infinite Value

A value that results from a computation whose result exceeds the capacity of the computer.

Input Range

A collection of argument sets for which a given math routine will return a valid result.

Intrinsic Function

A compiler-defined FORTRAN procedure that returns a single value.

M

Machine Round-Off Error

Machine round-off error is caused by the finite nature of the computer. Because only a finite number of bits can be represented in each word of memory, some precision is lost.

N

Number Types

A classification of the numbers processed by the math routines. The math routines perform computations on four number types: integer, single precision floating-point, double precision floating-point, and complex floating-point.

O

Output Range

The collection of results obtained by using the arguments in the input domain of each math routine for computation of the function or routine.

Q

Quintic

An algebraic function of the fifth degree. A quintic polynomial is a polynomial equation of the fifth degree.

R

Range

The collection of results obtained by entering members of the domain into an entry point.

Relative Error

The error of a function divided by the true value. The maximum relative error approximates the worst-case behavior of the function in the given interval.

Root Mean Square Relative Error

The square root of the average of the squares of the relative errors of all the arguments.

Routine

A computer subprogram that computes commonly occurring math functions and performs other tasks such as input and output. A method of referencing a subprogram, that is, either by values or by address.

S

Scalar

A constant, variable, array element, or substring of any type.

Stride

The distance measured in terms of array elements between two consecutive elements of the same dimension. For the Math Library, the stride is always equal to one.

U

Units in the Last Place (ulp)

A mathematical concept used to describe the accuracy of an algorithm.

V

Vector

One-dimensional array of up to 512 elements.

Vectorization

The manipulation of object code to reduce execution time taking advantage of the vector processing capabilities of the CYBER 180/990 Series running FORTRAN Version 2.

Related Manuals B

Table B-1 lists all manuals that are referenced in this manual or that contain background information. A complete list of NOS/VE manuals is given in the NOS/VE System Usage manual. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
explain
```

Table B-1 also lists a few VX/VE manuals. Additional VX/VE manuals are listed in the VX/VE Programmer Reference Manual.

Ordering Printed Manuals

You can order Control Data manuals through Control Data sales offices or through:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

Accessing Online Manuals

To access an online manual, log in to NOS/VE and specify the online manual title (listed in table B-1) on the EXPLAIN command. For example, to read the FORTRAN Version 1 Quick Reference online manual, enter:

```
explain manual=fortran
```

Table B-1. Related Manuals

Manual Title	Publication Number	Online Title
Ada for NOS/VE Usage	60498113	ADA
APL for NOS/VE Usage	60485813	
BASIC for NOS/VE Usage	60486313	BASIC
C for NOS/VE Usage	60469830	C
CYBIL Language Definition Usage	60464113	
Debug for NOS/VE Usage	60488213	
FORTRAN for NOS/VE LIB99	60485915	
FORTRAN Version 1 Language Definition Usage	60485913	
FORTRAN Version 1 Quick Reference	L60485918	FORTRAN
FORTRAN Version 2 Language Definition Usage	60487113	
FORTRAN Version 2 Quick Reference	L60487118	VFORTRAN
LISP for NOS/VE Language Definition Usage	60486213	
NOS/VE Diagnostic Messages	60484613	MESSAGES
NOS/VE System Usage	60464014	
Pascal for NOS/VE Usage	60485613	PASCAL
Prolog for NOS/VE Usage	60486713	PROLOG
VX/VE Programmer Reference Manual	60469820	
VX/VE User Guide	60469780	

ASCII Character Set C

Table C-1 gives the ASCII character set with the hexadecimal character code for each ASCII character.

See the appropriate language manual as listed in appendix B for additional ASCII character set tables.

Table C-1. ASCII Character Set and Collating Sequence

Collating Sequence Position	ASCII Code (Hexadecimal)	Graphic or Mnemonic	Name or Meaning
0	00	NULL	Null
1	01	SOH	Start of heading
2	02	STX	Start of text
3	03	ETX	End of text
4	04	EOT	End of transmission
5	05	ENQ	Enquiry
6	06	ACK	Acknowledge
7	07	BEL	Bell
8	08	BS	Backspace
9	09	HT	Horizontal tabulation
10	0A	LF	Line feed
11	0B	VT	Vertical tabulation
12	0C	FF	Form feed
13	0D	CR	Carriage return
14	0E	SO	Shift out
15	0F	SI	Shift in
16	10	DLE	Data link escape
17	11	DC1	Device control 1
18	12	DC2	Device control 2
19	13	DC3	Device control 3
20	14	DC4	Device control 4
21	15	NAK	Negative acknowledge
22	16	SYN	Synchronous idle
23	17	ETB	End of transmission block
24	18	CAN	Cancel
25	19	EM	End of medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	File separator
29	1D	GS	Group separator
30	1E	RS	Record separator
31	1F	US	Unit separator
32	20	SP	Space
33	21	!	Exclamation point
34	22	"	Quotation marks
35	23	#	Number sign
36	24	$	Dollar sign
37	25	%	Percent sign
38	26	&	Ampersand
39	27	'	Apostrophe

(Continued)

Table C-1. ASCII Character Set and Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexadecimal)	Graphic or Mnemonic	Name or Meaning
40	28	(	Opening parenthesis
41	29	)	Closing parenthesis
42	2A	*	Asterisk
43	2B	+	Plus
44	2C	,	Comma
45	2D	-	Hyphen
46	2E	.	Period
47	2F	/	Slant
48	30	0	Zero
49	31	1	One
50	32	2	Two
51	33	3	Three
52	34	4	Four
53	35	5	Five
54	36	6	Six
55	37	7	Seven
56	38	8	Eight
57	39	9	Nine
58	3A	:	Colon
59	3B	;	Semicolon
60	3C	<	Less than
61	3D	=	Equal to
62	3E	>	Greater than
63	3F	?	Question mark
64	40	@	Commercial at
65	41	A	Uppercase A
66	42	B	Uppercase B
67	43	C	Uppercase C
68	44	D	Uppercase D
69	45	E	Uppercase E
70	46	F	Uppercase F
71	47	G	Uppercase G
72	48	H	Uppercase H
73	49	I	Uppercase I
74	4A	J	Uppercase J
75	4B	K	Uppercase K
76	4C	L	Uppercase L
77	4D	M	Uppercase M
78	4E	N	Uppercase N
79	4F	O	Uppercase O

(Continued)

Table C-1. ASCII Character Set and Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexadecimal)	Graphic or Mnemonic	Name or Meaning
80	50	P	Uppercase P
81	51	Q	Uppercase Q
82	52	R	Uppercase R
83	53	S	Uppercase S
84	54	T	Uppercase T
85	55	U	Uppercase U
86	56	V	Uppercase V
87	57	W	Uppercase W
88	58	X	Uppercase X
89	59	Y	Uppercase Y
90	5A	Z	Uppercase Z
91	5B	[	Opening bracket
92	5C	\	Reverse slant
93	5D	]	Closing bracket
94	5E	^	Circumflex
95	5F	_	Underline
96	60	`	Grave accent
97	61	a	Lowercase a
98	62	b	Lowercase b
99	63	c	Lowercase c
100	64	d	Lowercase d
101	65	e	Lowercase e
102	66	f	Lowercase f
103	67	g	Lowercase g
104	68	h	Lowercase h
105	69	i	Lowercase i
106	6A	j	Lowercase j
107	6B	k	Lowercase k
108	6C	l	Lowercase l
109	6D	m	Lowercase m
110	6E	n	Lowercase n
111	6F	o	Lowercase o
112	70	p	Lowercase p
113	71	q	Lowercase q
114	72	r	Lowercase r
115	73	s	Lowercase s
116	74	t	Lowercase t
117	75	u	Lowercase u
118	76	v	Lowercase v
119	77	w	Lowercase w

(Continued)

Table C-1. ASCII Character Set and Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexadecimal)	Graphic or Mnemonic	Name or Meaning
120	78	x	Lowercase x
121	79	y	Lowercase y
122	7A	z	Lowercase z
123	7B	{	Opening brace
124	7C	|	Vertical line
125	7D	}	Closing brace
126	7E	~	Tilde
127	7F	DEL	Delete

ASCII codes 80 through FF hexadecimal (not listed in this table) are ordered as equal to the space (ASCII code 20 hexadecimal).

Bibliography D

Abramowitz, A. and Stegun I., *Handbook of Mathematical Functions*, AMS 55.

Carnahan, Luther, and Wilkes *Applied Numerical Methods* John Wiley & Sons, Inc., 1969.

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

Frankowski, K., *Algorithm and Constants*, Computer Information and Control Science, University of Minnesota.

Hart, Cheney, Lawson, et al. *Computer Approximations*, John Wiley and Sons, 1968.

Hastings, *Approximations for Digital Computers*, Princeton University Press, 1955.

Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, 1956.

James, G., ed., *Mathematics Dictionary*, Van Nostrand Reinhold Company, 1976.

Lanczos, Cornelius *Applied Analysis*, Prentice-Hall.

National Bureau of Standards, *Handbook of Mathematical Functions*, 1964.

Selby, S. M., *Standard Mathematical Tables*, CRC, 1971.

Ralston, A., and Herbert Wilf, ed. *Mathematical Methods for Digital Computers*, John Wiley & Sons, 1967.

Thomas, G. B., *Calculus and Analytic Geometry*, 1972.

Wall, H. S., *Analytic Theory of Continued Fractions*, D. Van Nostrand Co. Inc., 1948.

Wilkinson, J. H., *Rounding Errors in Algebraic Processes*, Prentice-Hall, 1963.

Index

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

X

Z

Comments (continued from other side)

Please fold on dotted line;
seal edges with tape only.

FOLD

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First-Class Mail Permit No. 8241 Minneapolis, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA
Technical Publications
SVL104
P.O. Box 3492
Sunnyvale, CA 94088-3492

We would like your comments on this manual to help us improve it. Please take a few minutes to fill out this form.

Who are you?

- ☐ Manager
- ☐ Systems analyst or programmer
- ☐ Applications programmer
- ☐ Operator
- ☐ Other ______

How do you use this manual?

- ☐ As an overview
- ☐ To learn the product or system
- ☐ For comprehensive reference
- ☐ For quick look-up
- ☐ Other ______

What programming languages do you use? ______

How do you like this manual? Answer the questions that apply.

Yes	Somewhat	No	
☐	☐	☐	Does it tell you what you need to know about the topic?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the order of topics logical?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Are there enough examples?
☐	☐	☐	Are the examples helpful? (☐ Too simple? ☐ Too complex?)
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Is the manual easy to read (print size, page layout, and so on)?
☐	☐	☐	Do you use this manual frequently?

Comments? If applicable, note page and paragraph. Use other side if needed.

Check here if you want a reply: ☐

Name ______ Company ______

Address ______ Date ______

Phone ______

Please send program listing and output if applicable to your comment.

Quick Index

The following quick index summarizes the math functions and their Math Library names, describes each function, and provides a page reference to the complete description in chapter 8.

CONTROL DATA