Contents

Preface ........................................................................................................ v

1 Introduction to MTH$

1.1 Entry Point Names ................................................................. 1-1
1.2 Calling Conventions .............................................................. 1-2
1.3 Algorithms .............................................................................. 1-3
1.4 Condition Handling ............................................................... 1-3
1.5 Complex Numbers ................................................................. 1-3
1.6 Mathematics Routines Not Documented in the MTH$ Reference
   Section ......................................................................................... 1-4
1.7 Examples of Calls to Run-Time Library Mathematics Routines . 1-9
   1.7.1 BASIC Example ......................................................... 1-9
   1.7.2 COBOL Example ....................................................... 1-9
   1.7.3 FORTRAN Examples ................................................. 1-10
   1.7.4 MACRO Examples ................................................... 1-11
   1.7.5 Pascal Examples ....................................................... 1-14
   1.7.6 PL/I Examples ........................................................... 1-15
   1.7.7 Ada Example .............................................................. 1-16

2 Vector Routines in MTH$

2.1 BLAS — Basic Linear Algebra Subroutines Level 1 ................. 2-1
   2.1.1 Using BLAS Level 1 .................................................. 2-5
   2.1.1.1 Memory Overlap ............................................... 2-5
   2.1.1.2 Round-Off Effects ............................................. 2-5
   2.1.1.3 Underflow and Overflow .................................. 2-5
   2.1.1.4 Notational Definitions ..................................... 2-5
   2.2 FOLR — First Order Linear Recurrence Routines ............... 2-6
   2.2.1 FOLR Routine Name Format .................................. 2-6
   2.2.2 Calling a FOLR Routine ......................................... 2-7
   2.3 Vector Versions of Existing Scalar Routines ....................... 2-7
   2.3.1 Exceptions ............................................................. 2-7
   2.3.2 Underflow Detection .............................................. 2-8
   2.3.3 Vector Routine Name Format .................................. 2-8
   2.3.4 Calling a Vector Math Routine ................................. 2-9
   2.4 Fast-Vector Math Routines ............................................. 2-11
   2.4.1 Exception Handling ............................................... 2-12
   2.4.2 Special Restrictions On Input Arguments .................. 2-13
   2.4.3 Accuracy ............................................................... 2-13
   2.4.4 Performance .......................................................... 2-13
### Scalar MTaHs Reference Section

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$\times$ACOS</td>
<td>MTH-3</td>
</tr>
<tr>
<td>MTH$\times$ACOSD</td>
<td>MTH-6</td>
</tr>
<tr>
<td>MTH$\times$ASIN</td>
<td>MTH-9</td>
</tr>
<tr>
<td>MTH$\times$ASIND</td>
<td>MTH-11</td>
</tr>
<tr>
<td>MTH$\times$ATAN</td>
<td>MTH-13</td>
</tr>
<tr>
<td>MTH$\times$ATAND</td>
<td>MTH-15</td>
</tr>
<tr>
<td>MTH$\times$ATAND2</td>
<td>MTH-17</td>
</tr>
<tr>
<td>MTH$\times$ATANH</td>
<td>MTH-19</td>
</tr>
<tr>
<td>MTH$\times$ATANH2</td>
<td>MTH-21</td>
</tr>
<tr>
<td>MTH$\times$ABS</td>
<td>MTH-23</td>
</tr>
<tr>
<td>MTH$\times$CCOS</td>
<td>MTH-26</td>
</tr>
<tr>
<td>MTH$\times$COS</td>
<td>MTH-28</td>
</tr>
<tr>
<td>MTH$\times$COSD</td>
<td>MTH-30</td>
</tr>
<tr>
<td>MTH$\times$COSH</td>
<td>MTH-32</td>
</tr>
<tr>
<td>MTH$\times$CLOG</td>
<td>MTH-34</td>
</tr>
<tr>
<td>MTH$\times$CLOG</td>
<td>MTH-36</td>
</tr>
<tr>
<td>MTH$\times$CMPLX</td>
<td>MTH-39</td>
</tr>
<tr>
<td>MTH$\times$CMPLX</td>
<td>MTH-41</td>
</tr>
<tr>
<td>MTH$\times$CONJG</td>
<td>MTH-43</td>
</tr>
<tr>
<td>MTH$\times$CONJG</td>
<td>MTH-44</td>
</tr>
<tr>
<td>MTH$\times$COS</td>
<td>MTH-46</td>
</tr>
<tr>
<td>MTH$\times$COSD</td>
<td>MTH-48</td>
</tr>
<tr>
<td>MTH$\times$COSH</td>
<td>MTH-50</td>
</tr>
<tr>
<td>MTH$\times$CSIN</td>
<td>MTH-52</td>
</tr>
<tr>
<td>MTH$\times$COS</td>
<td>MTH-53</td>
</tr>
<tr>
<td>MTH$\times$CSQRT</td>
<td>MTH-56</td>
</tr>
<tr>
<td>MTH$\times$CSQRT</td>
<td>MTH-58</td>
</tr>
<tr>
<td>MTH$\times$CVT_x_x</td>
<td>MTH-61</td>
</tr>
<tr>
<td>MTH$\times$CVT_xA_xA</td>
<td>MTH-63</td>
</tr>
<tr>
<td>MTH$\times$EXP</td>
<td>MTH-65</td>
</tr>
<tr>
<td>MTH$\times$HACOS</td>
<td>MTH-68</td>
</tr>
<tr>
<td>MTH$\times$HACOSD</td>
<td>MTH-70</td>
</tr>
<tr>
<td>MTH$\times$HASIN</td>
<td>MTH-72</td>
</tr>
<tr>
<td>MTH$\times$HASIND</td>
<td>MTH-74</td>
</tr>
<tr>
<td>MTH$\times$HATAN</td>
<td>MTH-76</td>
</tr>
<tr>
<td>MTH$\times$HATAND</td>
<td>MTH-78</td>
</tr>
<tr>
<td>MTH$\times$HATAND</td>
<td>MTH-80</td>
</tr>
<tr>
<td>MTH$\times$HATAND2</td>
<td>MTH-82</td>
</tr>
<tr>
<td>MTH$\times$HATANH</td>
<td>MTH-84</td>
</tr>
<tr>
<td>MTH$\times$HCOS</td>
<td>MTH-86</td>
</tr>
<tr>
<td>MTH$\times$HCOSD</td>
<td>MTH-87</td>
</tr>
<tr>
<td>MTH$\times$HCOSH</td>
<td>MTH-88</td>
</tr>
<tr>
<td>MTH$\times$HEXP</td>
<td>MTH-90</td>
</tr>
<tr>
<td>MTH$\times$HLOG</td>
<td>MTH-92</td>
</tr>
<tr>
<td>MTH$\times$HLOG2</td>
<td>MTH-94</td>
</tr>
</tbody>
</table>
MTH$HLOG10 ........................................ MTH-96
MTH$HSIN ........................................... MTH-98
MTH$HSIND .......................................... MTH-99
MTH$HSINH .......................................... MTH-101
MTH$HSQRT ......................................... MTH-103
MTH$HTAN ........................................... MTH-105
MTH$HTANH .......................................... MTH-107
MTH$xIMAG ........................................... MTH-111
MTH$xLOG ............................................ MTH-113
MTH$xLOG2 ........................................... MTH-115
MTH$xLOG10 ......................................... MTH-117
MTH$RANDOM ......................................... MTH-119
MTH$xREAL ........................................... MTH-121
MTH$xSIN ............................................ MTH-123
MTH$xSINCOS ........................................ MTH-125
MTH$xSINCOSD ....................................... MTH-128
MTH$xSIND ........................................... MTH-131
MTH$xSINH ........................................... MTH-133
MTH$xSQRT ........................................... MTH-136
MTH$xTAN ............................................ MTH-138
MTH$xTAND .......................................... MTH-140
MTH$xTANH .......................................... MTH-142
MTH$UMAX ............................................ MTH-144
MTH$UMIN ............................................ MTH-145

Vector MTHS Reference Section

BLAS1$VxAMAX ........................................ MTH-149
BLAS1$VxASUM ....................................... MTH-152
BLAS1$VxAXPY ....................................... MTH-155
BLAS1$VxCOPY ....................................... MTH-160
BLAS1$VxDOTx ....................................... MTH-165
BLAS1$VxNRM2 ....................................... MTH-170
BLAS1$VxROT ......................................... MTH-173
BLAS1$VxROTG ....................................... MTH-178
BLAS1$VxSCAL ....................................... MTH-182
BLAS1$VxSWAP ....................................... MTH-186
MTH$VxFOLRLy_MA_V15 ............................... MTH-190
MTH$VxFOLRLy_z_V8 ................................. MTH-194
MTH$VxFOLRLy_MA_V5 ............................... MTH-198
MTH$VxFOLRLy_z_V2 ................................. MTH-202
A  Additional MTH$ Routines

B  Vector MTH$ Routine Entry Points

Index

Tables

1–1  Additional Mathematics Routines ............................................. 1–4
2–1  Functions of BLAS Level 1 .................................................... 2–3
2–2  Determining the FOLR Routine You Need ................................. 2–7
2–3  Vector Routine Format — Underflow Signaling Enabled ............... 2–8
2–4  Vector Routine Format — Underflow Signaling Disabled ............... 2–8
2–5  Fast-Vector Math Routines .................................................... 2–12
2–6  Input Argument Restrictions ................................................ 2–13
A–1  Additional MTH$ Routines ................................................... A–1
B–1  Vector MTH$ Routines ......................................................... B–1
Preface

This manual provides users of the OpenVMS operating system with detailed usage and reference information on mathematics routines supplied in the MTH$ facility of the Run-Time Library.

Run-Time Library routines can be used only in programs written in languages that produce native code for the VAX hardware. At present, these languages include VAX MACRO and the following compiled high-level languages:

- VAX Ada
- VAX BASIC
- VAX BLISS-32
- VAX C
- VAX COBOL
- VAX COBOL-74
- VAX CORAL
- VAX DIBOL
- VAX FORTRAN
- VAX Pascal
- VAX PL/I
- VAX RPG
- VAX SCAN

Interpreted languages that can also access Run-Time Library routines include VAX DSM and VAX DATATRIEVE.

Intended Audience

This manual is intended for system and application programmers who want to call Run-Time Library routines.

Document Structure

This manual contains two tutorial chapters, two reference sections, and two appendixes:

- Chapter 1 is an introductory chapter that provides guidelines on using the MTH$ scalar routines.
- Chapter 2 provides guidelines on using the MTH$ vector routines.
- The Scalar MTH$ Reference Section provides detailed reference information on each scalar mathematics routine contained in the MTH$ facility of the Run-Time Library. The routines in this section are the same as those provided in VMS Version 5.5.
- The Vector MTH$ Reference Section provides detailed reference information on the BLAS Level 1 (Basic Linear Algebra Subroutines) and FOLR (First Order Linear Recurrence) routines.
Reference information is presented using the documentation format described in the OpenVMS Programming Interfaces: Calling a System Routine. Routine descriptions are in alphabetical order by routine name.

- Appendix A lists supported MTH$ routines not included with the routines in the Scalar MTH$ Reference Section, because they are rarely used.
- Appendix B contains all of the vector MTH$ routines that you can call from VAX MACRO in one table.

Associated Documents

The Run-Time Library routines are documented in a series of reference manuals. A description of how the Run-Time Library routines are accessed is presented in OpenVMS Programming Interfaces: Calling a System Routine. A description of OpenVMS features and functionality available through calls to the MTH$ Run-Time Library appears in OpenVMS Programming Concepts Manual. Descriptions of the other RTL facilities and their corresponding routines are presented in the following books:

- DPML, Digital Portable Mathematics Library
- OpenVMS RTL DECTalk (DTK$) Manual
- OpenVMS RTL Library (LIB$) Manual
- OpenVMS RTL General Purpose (OTS$) Manual
- OpenVMS RTL Parallel Processing (PPL$) Manual
- OpenVMS RTL Screen Management (SMG$) Manual
- OpenVMS RTL String Manipulation (STR$) Manual

Application programmers using any language can refer to the Guide to Creating OpenVMS Modular Procedures for writing modular and reentrant code.

High-level language programmers will find additional information on calling Run-Time Library routines in their language reference manuals. Additional information may also be found in the language user’s guide provided with your OpenVMS language software.

For a complete list and description of the manuals in the OpenVMS documentation set, see Overview of OpenVMS Documentation.

Conventions

In this manual, every use of OpenVMS VAX means the OpenVMS VAX operating system.

The following conventions are also used in this manual:

- \texttt{Ctrl/x} A sequence such as \texttt{Ctrl/x} indicates that you must hold down the key labeled Control while you press another key or a pointing device button.
- \texttt{PF1 x} A sequence such as \texttt{PF1 x} indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
A sequence such as GOLD x indicates that you must first press and release the key defined GOLD, then press and release another key. GOLD key sequences can also have a slash (/), dash (-), or underscore (_) as a delimiter in EVE commands. In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)

A horizontal ellipse in examples indicates one of the following possibilities:

- Additional optional arguments in a statement have been omitted.
- The preceding item or items can be repeated one or more times.
- Additional parameters, values, or other information can be entered.

A vertical ellipse indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.

In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.

In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification, or in the syntax of a substring specification in an assignment statement.)

In format descriptions, braces surround a required choice of options; you must choose one of the options listed.

Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.

Boldface text is also used to show user input in Bookreader versions of the manual.

Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error number), command lines (for example, /PRODUCER=name), and command parameters in text.

Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.

All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.
The Run-Time Library mathematics routines may be called to perform a wide variety of computations including the following:

- Floating-point trigonometric function evaluation
- Exponentiation
- Complex function evaluation
- Complex exponentiation
- Miscellaneous function evaluation

The OTS$ facility provides additional language-independent arithmetic support routines.

This introduction to Run-Time Library mathematics routines includes examples of how to call mathematics routines from BASIC, COBOL, FORTRAN, MACRO, Pascal, PL/I, and Ada.

### 1.1 Entry Point Names

The names of the mathematics routines are formed by adding the MTH$ prefix to the function names.

When function arguments and returned values are of the same data type, the first letter of the name indicates this data type. When function arguments and returned values are of different data types, the first letter indicates the data type of the returned value, and the second letter indicates the data type of the argument(s).

The letters used as data type prefixes are listed below.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Word</td>
</tr>
<tr>
<td>J</td>
<td>Longword</td>
</tr>
<tr>
<td>D</td>
<td>D_floating</td>
</tr>
<tr>
<td>G</td>
<td>G_floating</td>
</tr>
<tr>
<td>H</td>
<td>H_floating</td>
</tr>
<tr>
<td>C</td>
<td>F_floating complex</td>
</tr>
<tr>
<td>CD</td>
<td>D_floating complex</td>
</tr>
<tr>
<td>CG</td>
<td>G_floating complex</td>
</tr>
</tbody>
</table>

Generally, F-floating data types have no letter designation. For example, MTH$SIN returns an F-floating value of the sine of an F-floating argument and MTH$DSIN returns a D-floating value of the sine of a D-floating argument.
Introduction to MTH$  
1.1 Entry Point Names

However, in some of the miscellaneous functions, F-floating data types are referenced by the letter designation A.

1.2 Calling Conventions

For calling conventions specific to the MTH$ vector routines, refer to Chapter 2. All calls to mathematics routines, as described in the FORMAT section of each routine, accept arguments passed by reference. JSB entry points accept arguments passed by value.

All mathematics routines return values in R0 or R0/R1 except those routines for which the values cannot fit in 64 bits. D-floating complex, G-floating complex, and H-floating values are data structures which are larger than 64 bits. Routines returning values that cannot fit in registers R0/R1 return their function values into the first argument in the argument list.

The notation JSB MTH$NAME_Rn, where n is the highest register number referenced, indicates that an equivalent JSB entry point is available. Registers R0:Rn are not preserved.

Routines with JSB entry points accept a single argument in R0:Rm, where m, which is defined in the following table, is dependent on the data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_floating</td>
<td>0</td>
</tr>
<tr>
<td>D_floating</td>
<td>1</td>
</tr>
<tr>
<td>G_floating</td>
<td>1</td>
</tr>
<tr>
<td>H_floating</td>
<td>3</td>
</tr>
</tbody>
</table>

A routine returning one value returns it to registers R0:Rm.

When a routine returns two values (for example, MTH$SIN COS), the first value is returned in R0:Rm and the second value is returned in (R<m+1>:R<2*m+1>). Note that for routines returning a single value, n>=m. For routines returning two values, n>=2*m + 1.

In general, CALL entry points for mathematics routines do the following:
- Disable floating-point underflow
- Enable integer overflow
- Cause no floating-point overflow or other arithmetic traps or faults
- Preserve all other enabled operations across the CALL

JSB entry points execute in the context of the caller with the enable operations as set by the caller. Since the routines do not cause arithmetic traps or faults, their operation is not affected by the setting of the arithmetic trap enables, except as noted.

For more detailed information on CALL and JSB entry points, refer to the OpenVMS Programming Interfaces: Calling a System Routine.
1.3 Algorithms

For those mathematics routines having corresponding algorithms, the complete algorithm can be found in the Description section of the routine description appearing in the MTH$ Reference Section of this manual.

1.4 Condition Handling

Error conditions are indicated by using the VAX signaling mechanism. The VAX signaling mechanism signals all conditions in mathematics routines as SEVERE by calling LIB$SIGNAL. When a SEVERE error is signaled, the default handler causes the image to exit after printing an error message. A user-established condition handler can be written to cause execution to continue at the point of the error by returning SS$_{\text{CONTINUE}}$. A mathematics routine returns to its caller after the contents of R0/R1 have been restored from the mechanism argument vector CHF$\text{L}_\text{MCH}_\text{SAVR0/R1}$. Thus, the user-established handler should correct CHF$\text{L}_\text{MCH}_\text{SAVR0/R1}$ to the desired function value to be returned to the caller of the mathematics routine.

D-floating complex, G-floating complex, and H-floating values cannot be corrected with a user-established condition handler, because R2/R3 is not available in the mechanism argument vector.

Note that it is more reliable to correct R0 and R1 to resemble R0 and R1 of a double-precision floating-point value. A double-precision floating-point value correction works for both single- and double-precision values.

If the correction is not performed, the floating-point reserved operand -0.0 is returned. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Accessing the floating-point reserved operand will cause a reserved operand fault. See the *OpenVMS RTL Library (LIB$) Manual* for a complete description of how to write user condition handlers for SEVERE errors.

A few mathematics routines signal floating underflow if the calling program (JSB or CALL) has enabled floating underflow faults or traps.

All mathematics routines access input arguments and the real and imaginary parts of complex numbers using floating-point instructions. Therefore, a reserved operand fault can occur in any mathematics routine.

1.5 Complex Numbers

A complex number $y$ is defined as an ordered pair of real numbers $r$ and $i$, where $r$ is the real part and $i$ is the imaginary part of the complex number.

$y = (r,i)$

OpenVMS supports three floating-point complex types: F-floating complex, D-floating complex, and G-floating complex. There is no H-floating complex data type.

Run-Time Library mathematics routines that use complex arguments require a pointer to a structure containing two x-floating values to be passed by reference for each argument. The first x-floating value contains $r$, the real part of the complex number. The second x-floating value contains $i$, the imaginary part of the complex number. Similarly, Run-Time Library mathematics routines that return complex function values return two x-floating values. Some Language Independent Support (OTS$) routines also calculate complex functions.
1.5 Complex Numbers

Note that complex functions have no JSB entry points.

1.6 Mathematics Routines Not Documented in the MTH$ Reference Section

The mathematics routines in Table 1–1 are not found in the reference section of this manual. Instead, their entry points and argument information are listed in Appendix A of this manual.

A reserved operand fault can occur for any floating-point input argument in any mathematics routine. Other condition values signaled by each mathematics routine are indicated in the footnotes.

Table 1–1 Additional Mathematics Routines

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Absolute Value Routines</strong></td>
<td></td>
</tr>
<tr>
<td>MTH$ABS</td>
<td>F-floating absolute value</td>
</tr>
<tr>
<td>MTH$DABS</td>
<td>D-floating absolute value</td>
</tr>
<tr>
<td>MTH$GABS</td>
<td>G-floating absolute value</td>
</tr>
<tr>
<td>MTH$HABS</td>
<td>H-floating absolute value¹</td>
</tr>
<tr>
<td>MTH$IIABS</td>
<td>Word absolute value²</td>
</tr>
<tr>
<td>MTH$JIABS</td>
<td>Longword absolute value²</td>
</tr>
</tbody>
</table>

| **Bitwise AND Operator Routines** | |
| MTH$IIAND | Bitwise AND of two word arguments |
| MTH$JIAND | Bitwise AND of two longword arguments |

| **F-floating Conversion Routines** | |
| MTH$DBLE | Convert F-floating to D-floating (exact) |
| MTH$GDBLE | Convert F-floating to G-floating (exact) |
| MTH$IIFIX | Convert F-floating to word (truncated)² |
| MTH$JIFIX | Convert F-floating to longword (truncated)² |

¹Returns value to the first argument; value exceeds 64 bits.
²Integer overflow exceptions can occur.

(continued on next page)
### Table 1–1 (Cont.) Additional Mathematics Routines

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Floating-Point Positive Difference Routines</strong></td>
<td></td>
</tr>
<tr>
<td>MTH$DIM</td>
<td>Positive difference of two F-floating arguments$^3$</td>
</tr>
<tr>
<td>MTH$DDIM</td>
<td>Positive difference of two D-floating arguments$^3$</td>
</tr>
<tr>
<td>MTH$GDIM</td>
<td>Positive difference of two G-floating arguments$^3$</td>
</tr>
<tr>
<td>MTH$HDIM</td>
<td>Positive difference of two H-floating arguments$^1,^3$</td>
</tr>
<tr>
<td>MTH$IIDIM</td>
<td>Positive difference of two word arguments$^2$</td>
</tr>
<tr>
<td>MTH$JIDIM</td>
<td>Positive difference of two longword arguments$^2$</td>
</tr>
</tbody>
</table>

| **Bitwise Exclusive OR Operator Routines** |
| MTH$IIIEOR | Bitwise exclusive OR of two word arguments |
| MTH$JIEOR | Bitwise exclusive OR of two longword arguments |

| **Integer to Floating-Point Conversion Routines** |
| MTH$FLOATI | Convert word to F-floating (exact) |
| MTH$DFLOTI | Convert word to D-floating (exact) |
| MTH$GFLOTI | Convert word to G-floating (exact) |
| MTH$FLOATJ | Convert longword to F-floating (rounded) |
| MTH$DFLOTJ | Convert longword to D-floating (exact) |
| MTH$GFLOTJ | Convert longword to G-floating (exact) |

| **Conversion to Greatest Floating-Point Integer Routines** |
| MTH$FLOOR | Convert F-floating to greatest F-floating integer |
| MTH$DFLOOR | Convert D-floating to greatest D-floating integer |
| MTH$GFLOOR | Convert G-floating to greatest G-floating integer |
| MTH$HFLOOR | Convert H-floating to greatest H-floating integer$^1$ |

---

$^1$Returns value to the first argument; value exceeds 64 bits.

$^2$Integer overflow exceptions can occur.

$^3$Floating-point overflow exceptions can occur.

(continued on next page)
## Introduction to MTH$  
### 1.6 Mathematics Routines Not Documented in the MTH$ Reference Section

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Floating-Point Truncation Routines</strong></td>
<td></td>
</tr>
<tr>
<td>MTH$AINT</td>
<td>Convert F-floating to truncated F-floating</td>
</tr>
<tr>
<td>MTH$IIINT</td>
<td>Convert F-floating to truncated word²</td>
</tr>
<tr>
<td>MTH$JIINT</td>
<td>Convert F-floating to truncated longword²</td>
</tr>
<tr>
<td>MTH$DINT</td>
<td>Convert D-floating to truncated D-floating</td>
</tr>
<tr>
<td>MTH$IIDINT</td>
<td>Convert D-floating to truncated word²</td>
</tr>
<tr>
<td>MTH$JIDINT</td>
<td>Convert D-floating to truncated longword²</td>
</tr>
<tr>
<td>MTH$GINT</td>
<td>Convert G-floating to truncated G-floating</td>
</tr>
<tr>
<td>MTH$IIGINT</td>
<td>Convert G-floating to truncated word²</td>
</tr>
<tr>
<td>MTH$JIGINT</td>
<td>Convert G-floating to truncated longword²</td>
</tr>
<tr>
<td>MTH$HIINT</td>
<td>Convert H-floating to truncated H-floating¹</td>
</tr>
<tr>
<td>MTH$IIHINT</td>
<td>Convert H-floating to truncated word²</td>
</tr>
<tr>
<td>MTH$JIHINT</td>
<td>Convert H-floating to truncated longword²</td>
</tr>
</tbody>
</table>

| **Bitwise Inclusive OR Operator Routines** |
| MTH$IIOR | Bitwise inclusive OR of two word arguments |
| MTH$JIOR | Bitwise inclusive OR of two longword arguments |

| **Maximum Value Routines** |
| MTH$AIMAX0 | F-floating maximum of n word arguments |
| MTH$AJMAX0 | F-floating maximum of n longword arguments |
| MTH$IMAX0 | Word maximum of n word arguments |
| MTH$JMAX0 | Longword maximum of n longword arguments |
| MTH$AMAX1 | F-floating maximum of n F-floating arguments |
| MTH$DMAX1 | D-floating maximum of n D-floating arguments |
| MTH$GMAX1 | G-floating maximum of n G-floating arguments |
| MTH$HMAX1 | H-floating maximum of n H-floating arguments¹ |
| MTH$IMAX1 | Word maximum of n F-floating arguments² |
| MTH$JMAX1 | Longword maximum of n F-floating arguments² |

¹Returns value to the first argument; value exceeds 64 bits.
²Integer overflow exceptions can occur.

(continued on next page)
## Introduction to MTH$  

### 1.6 Mathematics Routines Not Documented in the MTH$ Reference Section

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimum Value Routines</strong></td>
<td></td>
</tr>
<tr>
<td>MTH$AIMINO</td>
<td>F-floating minimum of n word arguments</td>
</tr>
<tr>
<td>MTH$AJMINO</td>
<td>F-floating minimum of n longword arguments</td>
</tr>
<tr>
<td>MTH$IMIN0</td>
<td>Word minimum of n word arguments</td>
</tr>
<tr>
<td>MTH$JMIN0</td>
<td>Longword minimum of n longword arguments</td>
</tr>
<tr>
<td>MTH$AMIN1</td>
<td>F-floating minimum of n F-floating arguments</td>
</tr>
<tr>
<td>MTH$DMIN1</td>
<td>D-floating minimum of n D-floating arguments</td>
</tr>
<tr>
<td>MTH$GMIN1</td>
<td>G-floating minimum of n G-floating arguments</td>
</tr>
<tr>
<td>MTH$HMIN1</td>
<td>H-floating minimum of n H-floating arguments</td>
</tr>
<tr>
<td>MTH$IMIN1</td>
<td>Word minimum of n F-floating arguments</td>
</tr>
<tr>
<td>MTH$JMIN1</td>
<td>Longword minimum of n F-floating arguments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Remainder Routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$AMOD</td>
<td>Remainder of two F-floating arguments, arg1/arg2</td>
</tr>
<tr>
<td>MTH$DMOD</td>
<td>Remainder of two D-floating arguments, arg1/arg2</td>
</tr>
<tr>
<td>MTH$GMOD</td>
<td>Remainder of two G-floating arguments, arg1/arg2</td>
</tr>
<tr>
<td>MTH$HMOD</td>
<td>Remainder of two H-floating arguments, arg1/arg2</td>
</tr>
<tr>
<td>MTH$IMOD</td>
<td>Remainder of two word arguments, arg1/arg2</td>
</tr>
<tr>
<td>MTH$JMOD</td>
<td>Remainder of two longword arguments, arg1/arg2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Floating-Point Conversion to Nearest Value Routines</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$ANINT</td>
<td>Convert F-floating to nearest F-floating integer</td>
</tr>
<tr>
<td>MTH$ININT</td>
<td>Convert F-floating to nearest word integer</td>
</tr>
<tr>
<td>MTH$JNINT</td>
<td>Convert F-floating to nearest longword integer</td>
</tr>
<tr>
<td>MTH$DNINT</td>
<td>Convert D-floating to nearest D-floating integer</td>
</tr>
<tr>
<td>MTH$IDNNT</td>
<td>Convert D-floating to nearest word integer</td>
</tr>
<tr>
<td>MTH$JDNNT</td>
<td>Convert D-floating to nearest longword integer</td>
</tr>
<tr>
<td>MTH$GNINT</td>
<td>Convert G-floating to nearest G-floating integer</td>
</tr>
<tr>
<td>MTH$IGNNT</td>
<td>Convert G-floating to nearest word integer</td>
</tr>
<tr>
<td>MTH$JGNNT</td>
<td>Convert G-floating to nearest longword integer</td>
</tr>
</tbody>
</table>

---

1 Returns value to the first argument; value exceeds 64 bits.
2 Integer overflow exceptions can occur.
3 Floating-point overflow exceptions can occur.
4 Divide-by-zero exceptions can occur.
5 Floating-point underflow exceptions are signaled.
## Introduction to MTH$

### 1.6 Mathematics Routines Not Documented in the MTH$ Reference Section

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$HNINT</td>
<td>Convert H-floating to nearest H-floating integer$^1$</td>
</tr>
<tr>
<td>MTH$IHNNNT</td>
<td>Convert H-floating to nearest word integer$^2$</td>
</tr>
<tr>
<td>MTH$JHNNNT</td>
<td>Convert H-floating to nearest longword integer$^2$</td>
</tr>
</tbody>
</table>

**Bitwise Complement Operator Routines**

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$INOT</td>
<td>Bitwise complement of word argument</td>
</tr>
<tr>
<td>MTH$JNOT</td>
<td>Bitwise complement of longword argument</td>
</tr>
</tbody>
</table>

**Floating-Point Multiplication Routines**

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$DPROD</td>
<td>D-floating product of two F-floating arguments$^3$</td>
</tr>
<tr>
<td>MTH$GPROD</td>
<td>G-floating product of two F-floating arguments</td>
</tr>
</tbody>
</table>

**Bitwise Shift Operator Routines**

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$IISHFT</td>
<td>Bitwise shift of word</td>
</tr>
<tr>
<td>MTH$JISHFT</td>
<td>Bitwise shift of longword</td>
</tr>
</tbody>
</table>

**Floating-Point Sign Function Routines**

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$SGN</td>
<td>F- or D-floating sign function</td>
</tr>
<tr>
<td>MTH$SIGN</td>
<td>F-floating transfer of sign of y to sign of x</td>
</tr>
<tr>
<td>MTH$DSIGN</td>
<td>D-floating transfer of sign of y to sign of x</td>
</tr>
<tr>
<td>MTH$GSIGN</td>
<td>G-floating transfer of sign of y to sign of x</td>
</tr>
<tr>
<td>MTH$HSIGN</td>
<td>H-floating transfer of sign of y to sign of x$^1$</td>
</tr>
<tr>
<td>MTH$IISIGN</td>
<td>Word transfer of sign of y to sign of x</td>
</tr>
<tr>
<td>MTH$JISIGN</td>
<td>Longword transfer of sign of y to sign of x</td>
</tr>
</tbody>
</table>

---

$^1$Returns value to the first argument; value exceeds 64 bits.

$^2$Integer overflow exceptions can occur.

$^3$Floating-point overflow exceptions can occur.

(continued on next page)
1.6 Mathematics Routines Not Documented in the MTH$ Reference Section

Table 1–1 (Cont.) Additional Mathematics Routines

<table>
<thead>
<tr>
<th>Entry Point</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion of Double to Single Floating-Point Routines</td>
<td></td>
</tr>
<tr>
<td>MTH$SNGL</td>
<td>Convert D-floating to F-floating (rounded)(^3)</td>
</tr>
<tr>
<td>MTH$SNGLG</td>
<td>Convert G-floating to F-floating (rounded)(^3,4)</td>
</tr>
</tbody>
</table>

\(^3\)Floating-point overflow exceptions can occur. 
\(^4\)Floating-point underflow exceptions can occur.

1.7 Examples of Calls to Run-Time Library Mathematics Routines

1.7.1 BASIC Example

The following BASIC program uses the H-floating data type. BASIC also supports the D-floating, F-floating, and G-floating data types, but does not support the complex data types.

```basic
10 ! Sample program to demonstrate a call to MTH$HEXP from BASIC.
!-
EXTERNAL SUB MTH$HEXP ( HFLOAT, HFLOAT )
DECLARE HFLOAT X,Y ! X and Y are H-floating
DIGIT$ = '###.#################################
X = '1.2345678901234567891234567892'H
CALL MTH$HEXP (Y,X)
A$ = 'MTH$HEXP of ' + DIGIT$ + ' is ' + DIGIT$
PRINT USING A$, X, Y
END
```

The output from this program is as follows:

MTH$HEXP of 1.234567890123456789123456789200000
is 3.436893084346008004973301321342110

1.7.2 COBOL Example

The following COBOL program uses the F-floating and D-floating data types. COBOL does not support the G-floating and H-floating data types or the complex data types.

This COBOL program calls MTH$EXP and MTH$DEXP.
Introduction to MTH$  
1.7 Examples of Calls to Run-Time Library Mathematics Routines

IDENTIFICATION DIVISION.
PROGRAM-ID. FLOATING_POINT.
* Calls MTH$EXP using a Floating Point data type.
* Calls MTH$DEXP using a Double Floating Point data type.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 FLOAT_PT COMP-1.
01 ANSWER_F COMP-1.
01 DOUBLE_PT COMP-2.
01 ANSWER_D COMP-2.
PROCEDURE DIVISION.
  P0.
    MOVE 12.34 TO FLOAT_PT.
    MOVE 3.456 TO DOUBLE_PT.
    CALL "MTH$EXP" USING BY REFERENCE FLOAT_PT GIVING ANSWER_F.
    DISPLAY " MTH$EXP of ", FLOAT_PT CONVERSION, " is ",
                     ANSWER_F CONVERSION.
    CALL "MTH$DEXP" USING BY REFERENCE DOUBLE_PT GIVING ANSWER_D.
    DISPLAY " MTH$DEXP of ", DOUBLE_PT CONVERSION, " is ",
                     ANSWER_D CONVERSION.
    STOP RUN.

The output from this example program is as follows:
MTH$EXP of 1.234000E+01 is 2.286620E+05
MTH$DEXP of 3.456000000000000E+00 is 3.168996280537917E+01

1.7.3 FORTRAN Examples

The first FORTRAN program below uses the G-floating data type. The second
FORTRAN program below uses the H-floating data type. The third FORTRAN
program below uses the F-floating complex data type. FORTRAN supports the
four floating data types and the three complex data types.

1. C
   This FORTRAN program computes the log base 2 of x, log2(x) in
   G-floating double precision by using the RTL routine MTH$GLOG2.
   C
   Declare X and Y and MTH$GLOG2 as double precision values.
   C
   MTH$GLOG2 will return a double precision value to variable Y.
   C
   REAL*8 X, Y, MTH$GLOG2
   X = 16.0
   Y = MTH$GLOG2(X)
   WRITE (6,1) X, Y
   1 FORMAT (' MTH$GLOG2(',F4.1,') is ',F4.1)
   END

The output generated by the preceding program is as follows:
MTH$GLOG2(16.0) is 4.0
2. C
   C This FORTRAN program computes the log base 2 of x, log2(x) in
   C H-floating precision by using the RTL routine MTH$HL0G2.
   C
   C Declare X and Y and MTH$GL0G2 as REAL*16 values.
   C
   C MTH$HL0G2 will return a REAL*16 value to variable Y.
   C-
   REAL*16 X, Y
   X = 16.12345678901234567890123456789
   CALL MTH$HL0G2(Y, X)
   WRITE (6,1) X, Y
   1 FORMAT (' MTH$HLOG2(',F30.27, ') is ' ,F30.28)
   END

   The output generated by the preceding program is as follows:
   MTH$HLOG2(16.123456789012345678901234568) is 4.0110891785623860194931388310

3. C
   C This FORTRAN example raises a complex base to
   C a NONNEGATIVE integer power using OTS$POWCJ.
   C
   C Declare Z1, Z2, Z3, and OTS$POWCJ as complex values.
   C Then OTS$POWCJ returns the complex result of
   C Z1**Z2: Z3 = OTS$POWCJ(Z1,Z2),
   C where Z1 and Z2 are passed by value.
   C-
   COMPLEX Z1,Z3,OTS$POWCJ
   INTEGER Z2
   C+
   C Generate a complex base.
   C-
   Z1 = (2.0,3.0)
   C+
   C Generate an integer power.
   C-
   Z2 = 2
   C+
   C Compute the complex value of Z1**Z2.
   C-
   Z3 = OTS$POWCJ( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(Z2))
   TYPE 1,Z1,Z2,Z3
   1 FORMAT(' The value of (' ,F10.8, ',',F11.8, ')**',II,7,' is '+ ',F11.8, ',',F12.8, ' •')
   END

   The output generated by the preceding FORTRAN program is as follows:
   The value of (2.00000000, 3.00000000)**2 is (-5.00000000, 12.00000000).

1.7.4 MACRO Examples

MACRO and BLISS support JSB entry points as well as CALLS and CALLG
entry points. Both MACRO and BLISS support the four floating data types and
the three complex data types.

The following MACRO programs show the use of the CALLS and CALLG
instructions, as well as JSB entry points.
1.7 Examples of Calls to Run-Time Library Mathematics Routines

1. .TITLE EXAMPLE_JSB
   ; This example calls MTH$DEXP by using a MACRO JSB command.
   ; The JSB command expects R0/R1 to contain the quadword input value X.
   ; The result of the JSB will be located in R0/R1.
   ;
   \*.EXTERN MTH$DEXP_R6 ;MTH$DEXP is an external routine.
   \*.PSECT DATA, PIC, EXE, NOWRT
   X: \.DOUBLE 2.0 ; X is 2.0
   \.ENTRY EXAMPLE_JSB, \^M>
   MOVQ X, R0 ; X is in registers R0 and R1
   JSB G*MTH$DEXP_R6 ; The result is returned in R0/R1.
   RET
   \.END EXAMPLE_JSB

This MACRO program generates the following output:

R0 <— 732541EC
R1 <— ED6EC6A6

That is, MTH$DEXP(2) is 7.38905609893065022723042746057501

2. .TITLE EXAMPLE_CALLG
   ; This example calls MTH$HEXP by using a MACRO CALLG command.
   ; The CALLG command expects that the address of the return value
   ; Y, the address of the input value X, and the argument count 2 be
   ; stored in memory; this program stores this information in ARGUMENTS.
   ; The result of the CALLG will be located in R0/R1.
   ;
   \*.EXTERN MTH$HEXP
   \*.PSECT DATA, PIC, EXE, WRT
   ARGUMENTS:
   \.LONG 2 ; The CALLG will use two arguments.
   \.ADDRESS Y, X ; The first argument must be the address
                  ; receiving the computed value, while
                  ; the second argument is used to
                  ; compute exp(X).
   X: \.HFLOAT 2 ; X = 2.0
   Y: \.HFLOAT 0 ; Y is the result, initially set to 0.
   \.ENTRY EXAMPLE_G, \^M>
   CALLG ARGUMENTS, G*MTH$HEXP ; CALLG returns the value to Y.
   RET
   \.END EXAMPLE_G

The output generated by this MACRO program is as follows:

address of Y <--- D8B54003
<--- 4DDA4B8D
<--- 3A3BDC3
<--- B6BBA206

That is, MTH$HEXP of 2.0 returns

7.38905609893065022723042746057501

1-12
1.7 Examples of Calls to Run-Time Library Mathematics Routines

3. +

; This example calls MTH$HEXP by using the MACRO CALLS command.
; The CALLS command expects the SP to contain the H-floating address of
; the return value, the address of the input argument X, and the argument
; count 2. The result of the CALLS will be located in registers R0-R3.
;
; EXTRN MTH$HEXP
; MTH$HEXP is an external routine.
.PSECT DATA, PIC, EXE, WRT
Y: .H_FLOATING 0
X: .H_FLOATING 2
; ENTRY EXAMPLE_S, "M<>
MOVAL X, -(SP)
MOVAL Y, -(SP)
CALLS Y, G'MTH$HEXP
; The value is returned to the address of Y
RET
.END EXAMPLE_S

The output generated by this program is as follows:
address of Y <-- D8E64003
 <-- 4DDA4B8D
 <-- 3A3BDCC3
 <-- B68BA206
That is, MTH$HEXP of 2.0 returns
7.38905609893065022723042746057501

4. +

; This example calls MTH$CLOG by using a MACRO CALLG command.
; To compute the complex natural logarithm of Z = (2.0,1.0) register
; R0 is loaded with 2.0, the real part of Z, and register R1 is loaded
; with 1.0, the imaginary part of Z. The CALLG to MTH$CLOG
; returns the value of the natural logarithm of Z in
; registers R0 and R1. R0 gets the real part of Z and R1
; gets the imaginary part.
;
; EXTRN MTH$CLOG
.PSECT DATA, PIC, EXE, NOWRT
ARGS: .LONG 1
.ADDRESS REAL
; The CALLG will use one argument.
; The one argument that the CALLG
; uses is the address of the argument
; of MTH$CLOG.
REAL: .FLOAT 2
IMAG: .FLOAT 1
; ENTRY COMPLEX_EX1, "M<>
CALLG ARGS, G'MTH$CLOG; MTH$CLOG returns the real part of the
; complex natural logarithm in R0 and
; the imaginary part in R1.
RET
.END COMPLEX_EX1

This program generates the following output:
R0 <-- 0210404E
R1 <-- 63383FED
That is, MTH$CLOG(2.0,1.0) is
(0.804719,0.4636476)
1.7 Examples of Calls to Run-Time Library Mathematics Routines

5.

This example calls MTH$CL0G by using a MACRO CALLS command.

```assembly
TITLE COMPLEX_EX2

* This example calls MTH$CL0G by using a MACRO CALLS command.
* To compute the complex natural logarithm of Z = (2.0,1.0) register
* R0 is loaded with 2.0, the real part of Z, and register R1 is loaded
* with 1.0, the imaginary part of Z. The CALLS to MTH$CL0G
* returns the value of the natural logarithm of Z in registers R0
* and R1. R0 gets the real part of Z and R1 gets the imaginary
* part.
*
.EXTRN
MTH$CLOG
.PSECT
DATA, PIC, EXE, NOWRT
REAL: .FLOAT 2 ; real part of Z is 2.0
IMAG: .FLOAT 1 ; imaginary part Z is 1.0
.ENTRY
COMPLEX_EX2, A
MOVAL REAL, -(SP) ; SP <-- address of Z. Real part of Z is
CALLS #1, G MTH$CLOG ; in @(SP) and imaginary part is in
@(@SP)+4.
; MTH$CLOG return the real part of the
; complex natural logarithm in R0 and
; the imaginary part in R1.
RET
.END COMPLEX_EX2
```

This MACRO example program generates the following output:

R0 <--- 0210404E
R1 <--- 63383FED

That is, MTH$CLOG(2.0,1.0) is
(0.8047190,0.4636476)

1.7.5 Pascal Examples

The following Pascal programs use the D-floating and H-floating data types. Pascal also supports the F-floating and G-floating data types. Pascal does not support the complex data types, however.

1. {+}
{ Sample program to demonstrate a call to MTH$DEXP from PASCAL.
{-}
PROGRAM CALL_MTH$DEXP (OUTPUT);
{+}
{ Declare variables used by this program.
{-}
VAR
X : DOUBLE := 3.456; { X,Y are D-floating unless overridden } 
Y : DOUBLE; { with /DOUBLE qualifier on compilation } 
{+}
{ Declare the RTL routine used by this program.
{-}
[EXTERNAL,ASYNCHRONOUS] FUNCTION MTH$DEXP (VAR value : DOUBLE) : DOUBLE; EXTERN;
BEGIN
  Y := MTH$DEXP (X);
  WRITELN ('MTH$DEXP of ', X:5:3, ' is ', Y:20:16);
END.

The output generated by this Pascal program is as follows:

MTH$DEXP of 3.456 is 31.6899656462382318
1.7 Examples of Calls to Run-Time Library Mathematics Routines

2. {+} { Sample program to demonstrate a call to MTH$HEXP from PASCAL. {-}

PROGRAM CALL_MTH$HEXP (OUTPUT);
{+} { Declare variables used by this program. {-}

VAR
    X : QUADRUPLE := 1.2345678901234567891234567892; { X is H-floating }
    Y : QUADRUPLE; { Y is H-floating }
{+} { Declare the RTL routine used by this program. {-}

[EXTERNAL,ASYNCHRONOUS] PROCEDURE MTH$HEXP (VAR h_exp : QUADRUPLE;
    value : QUADRUPLE); EXTERN;
BEGIN
    MTH$HEXP (Y,X);
    WRITELN ('MTH$HEXP of X:30:28, ' is ', Y:35:33);
END.

This Pascal program generates the following output:
MTH$DEXP of 3.456 is 31.6899656462382318

1.7.6 PL/I Examples

The following PL/I programs use the D-floating and H-floating data types to test entry points. PL/I also supports the F-floating and G-floating data types. PL/I does not support the complex data types, however.

1. /*
   * This program tests a MTH$D entry point
   *
   */
   TEST: PROC OPTIONS (MAIN) ;
   DCL (MTH$DEXP)
       ENTRY (FLOAT(53)) RETURNS (FLOAT(53));
   DCL OPERAND FLOAT(53);
   DCL RESULT FLOAT(53);
   /***
      Begin test 
   ***
   OPERAND = 3.456;
   RESULT = MTH$DEXP(OPERAND);
   PUT EDIT ('MTH$DEXP of ', OPERAND, ' is ',
              RESULT)(A(12),F(5,3),A(4),F(20,15));
END TEST;

The output generated by this PL/I program is as follows:
MTH$DEXP of 3.456 is 31.689962805379165
1.7 Examples of Calls to Run-Time Library Mathematics Routines

This program tests a MTH$ entry point. Note that in the PL/I statement below, the /G-float switch is needed to compile both G- and H-floating point MTH$ routines.

```pli
PROC OPTIONS (MAIN) ;
DCL (MTH$HEXP)
ENTRY (FLOAT (113), FLOAT (113)) ;
DCL OPERAND FLOAT (113);
DCL RESULT FLOAT (113);
/*** Begin test ***/
OPERAND = 1.234578901234567891234567892;
CALL MTH$HEXP(RESULT,OPERAND);
PUT EDIT ('MTH$HEXP of ', OPERAND, is ', RESULT) (A(12),F(29,27),A(4),F(29,27));
END TEST;
```

To run this program, use the following DCL commands:

```dcl
$ PLI/G_FLOAT EXAMPLE
$ LINK EXAMPLE
$ RUN EXAMPLE
```

This program generates the following output:

```
MTH$HEXP of 1.234578901234567891234567892 is 3.436930928565989790506225633
```

1.7.7 Ada Example

The following Ada program demonstrates the use of MTH$ routines in a manner that an actual program might use. The program performs the following steps:

- Reads a floating-point number from the terminal
- Calls MTH$SQRT to obtain the square root of the value read
- Calls MTH$JNINT to find the nearest integer of the square root
- Displays the result

This example runs on VAX Ada Version 2.0 or later.

```ada
begin
-- This Ada program calls the MTH$SQRT and MTH$JNINT routines.
-- This Ada program calls the MTH$SQRT and MTH$JNINT routines.
-- with FLOAT_MATH_LIB;
-- Package FLOAT_MATH_LIB is an instantiation of the generic package
-- MATH_LIB for the FLOAT datatype. This package provides the most
-- common mathematical functions (SQRT, SIN, COS, etc.) in an easy
-- to use fashion. An added benefit is that the VAX Ada compiler
-- will use the faster JSB interface for these routines.
with MTH;
-- Package MTH defines all the MTH$ routines. It should be used when
-- package MATH_LIB is not sufficient. All functions are defined here
-- as "valued procedures" for consistency.
with FLOAT_TEXT_IO, INTEGER_TEXT_IO, TEXT_IO;
procedure ADA_EXAMPLE is
  FLOAT_VAL: FLOAT;
  INT_VAL: INTEGER;
begin
-- Prompt for initial value.
  TEXT_IO.PUT ('Enter value: ');
  FLOAT_TEXT_IO.GET (FLOAT_VAL);
  TEXT_IO.NEW_LINE;
```
1.7 Examples of Calls to Run-Time Library Mathematics Routines

-- Take the square root by using the SQRT routine from package
-- FLOAT_MATH_LIB. The compiler will use the JSB interface
to MTH$SQRT.
FLOAT_VAL := FLOAT_MATH_LIB.SQRT (FLOAT_VAL);

-- Find the nearest integer using MTH$JNINT. Argument names are
-- the same as those listed for MTH$JNINT in the reference
-- section of this manual.
MTH.JNINT (F_FLOATING => FLOAT_VAL, RESULT => INT_VAL);

-- Write the result.
TEXT_IO.PUT (*Result is: *);
INTEGER_TEXT_IO.PUT (INT_VAL);
TEXT_IO.NEW_LINE;
end ADA_EXAMPLE;

To run this example program, use the following DCL commands:

$ CREATE/DIR [.ADALIB]
$ ACS CREATE LIB [.ADALIB]
$ ACS SET LIB [.ADALIB]
$ ADA ADA_EXAMPLE
$ ACS LINK ADA_EXAMPLE
$ RUN ADA_EXAMPLE

The preceding Ada example generates the following output:

Enter value: 42.0
Result is: 6

This chapter discusses four sets of routines provided by the RTL MTH$ facility that support vector processing. These routines are as follows:

- Basic Linear Algebra Subroutines (BLAS) Level 1
- First Order Linear Recurrence (FOLR) routines
- Vector versions of existing scalar routines
- Fast-Vector math routines

2.1 BLAS — Basic Linear Algebra Subroutines Level 1

The BLAS Level 1 routines perform operations on vectors, such as copying a vector to another vector, swapping vectors, and so on. These routines help you take advantage of the vector processing speed. BLAS Level 1 routines form an integral part of many mathematical libraries such as LINPACK and EISPACK. Because these routines usually occur in the innermost loops of user code, the Run-Time Library provides versions of the BLAS Level 1 that are tuned to take best advantage of the VAX vector processors.

Two versions of BLAS Level 1 are provided. To use either of these libraries, link in the appropriate shareable image. The libraries are:

- Scalar BLAS — contained in the shareable image BLAS1RTL
- Vector BLAS (routines that take advantage of vectorization) — contained in the shareable image VBLAS1RTL

Note

To call the scalar BLAS from a program that runs on scalar hardware, specify the routine name preceded by BLAS1$ (for example, BLAS1$xCOPY). To call the vector BLAS from a program that runs on vector hardware, specify the routine name preceded by BLAS1$V (for example, BLAS1$VxCOPY).

This manual describes both the scalar and vector versions of BLAS Level 1, but for simplicity the vector prefix (BLAS1$V) is used exclusively. Remember to remove the letter V from the routine prefix when you want to call the scalar version.

1 For more information, see Basic Linear Algebra Subprograms for FORTRAN Usage in ACM Transactions on Mathematical Software, Vol. 5, No. 3, September 1979.
If you are a VAX FORTRAN programmer, do not specify BLAS vector routines explicitly. Specify the FORTRAN intrinsic function name only. The VAX FORTRAN-HPO compiler will then determine whether the vector or scalar version of a BLAS routine should be used. The FORTRAN /BLAS=(NO)INLINE,(NO)MAPPED) qualifier controls how the compiler processes calls to BLAS Level 1. If /NOBLAS is specified, then all BLAS calls are treated as ordinary external routines. The default of INLINE means that calls to BLAS Level 1 routines will be treated as known language constructs, and VAX object code will be generated to compute the corresponding operations at the call site, rather than call a user-supplied routine. If the FORTRAN qualifier /VECTOR or /PARALLEL=AUTO is in effect, the generated code for the loops may use vector instructions or be decomposed to run on multiple processors. If MAPPED is specified, these calls will be treated as calls to the optimized implementations of these routines in the BLAS1$ and BLAS1$V portions of the MTH$ facility. For more information on the FORTRAN /BLAS qualifier, refer to the VAX FORTRAN Performance Guide.

Ten families of routines form BLAS Level 1. (BLAS1$VxCOPY is one family of routines, for example.) These routines operate at the vector-vector operation level. This means that BLAS Level 1 perform operations on one or two vectors. The level of complexity of the computations (in other words, the number of operations being performed in a BLAS Level 1 routine) is of the order $n$ (the length of the vector).

Each family of routines in BLAS Level 1 contains routines coded in single precision, double precision (D and G formats), single precision complex, and double precision complex (D and G formats). BLAS Level 1 can be broadly classified into three groups:

- **BLAS1$VxCOPY, BLAS1$VxSWAP, BLAS1$VxSCAL and BLAS1$VxAXPY**: These routines return vector output(s) for vector inputs. The results of all these routines are independent of the order in which the elements of the vector are processed. The scalar and vector versions of these routines return the same results.

- **BLAS1$VxDOT, BLAS1$VxAMAX, BLAS1$VxASUM, and BLAS1$VxNRM2**: These routines are all reduction operations that return a scalar value. The results of these routines (except BLAS1$VxAMAX) are dependent upon the order in which the elements of the vector are processed. The scalar and vector versions of BLAS1$VxDOT, BLAS1$VxASUM, and BLAS1$VxNRM2 can return different results. The scalar and vector versions of BLAS1$VxAMAX return the same results.

- **BLAS1$VxROTG and BLAS1$VxROT**: These routines are used for a particular application (plane rotations), unlike the routines in the previous two categories. The results of BLAS1$VxROTG and BLAS1$VxROT are independent of the order in which the elements of the vector are processed. The scalar and vector versions of these routines return the same results.

Table 2–1 lists the functions and corresponding routines of BLAS Level 1.
## Table 2-1 Functions of BLAS Level 1

<table>
<thead>
<tr>
<th>Function</th>
<th>Routine</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy a vector to another vector</td>
<td>BLAS1$VSCOPY</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDCOPY</td>
<td>Double (D-floating or G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCCOPY</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZCOPY</td>
<td>Double complex (D-floating or G-floating)</td>
</tr>
<tr>
<td>Swap the elements of two vectors</td>
<td>BLAS1$VSSWAP</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDSWAP</td>
<td>Double (D-floating or G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCSWAP</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZSWAP</td>
<td>Double complex (D-floating or G-floating)</td>
</tr>
<tr>
<td>Scale the elements of a vector</td>
<td>BLAS1$VSSCAL</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDSCAL</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGSCAL</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCSCAL</td>
<td>Single complex with complex scale</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCSSCAL</td>
<td>Single complex with real scale</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZSCAL</td>
<td>Double complex with complex scale (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWSCAL</td>
<td>Double complex with complex scale (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZDSCAL</td>
<td>Double complex with real scale (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWGSCLAL</td>
<td>Double complex with real scale (G-floating)</td>
</tr>
<tr>
<td>Multiply a vector by a scalar and add a vector</td>
<td>BLAS1$VSAXPY</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDAXPY</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGAXPY</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCAXPY</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZAXPY</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWAXPY</td>
<td>Double complex (G-floating)</td>
</tr>
<tr>
<td>Obtain the index of the first element of a vector having the largest absolute value</td>
<td>BLAS1$VISAMAX</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VIDAMAX</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VIGAMAX</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VICAMAX</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VIZAMAX</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VIWAMAX</td>
<td>Double complex (G-floating)</td>
</tr>
</tbody>
</table>

(continued on next page)
## Table 2-1 (Cont.) Functions of BLAS Level 1

<table>
<thead>
<tr>
<th>Function</th>
<th>Routine</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain the sum of the absolute values of the elements of a vector</td>
<td>BLAS1$VSASUM</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDASUM</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGASUM</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VSCASUM</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDZASUM</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGWASUM</td>
<td>Double complex (G-floating)</td>
</tr>
<tr>
<td>Obtain the inner product of two vectors</td>
<td>BLAS1$VSDOT</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDDOT</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGDOT</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCDOTU</td>
<td>Single complex unconjugated</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCDCOTC</td>
<td>Single complex conjugated</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZDOTU</td>
<td>Double complex unconjugated (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZDOTC</td>
<td>Double complex conjugated (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWDOTU</td>
<td>Double complex unconjugated (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWDOTC</td>
<td>Double complex conjugated (G-floating)</td>
</tr>
<tr>
<td>Obtain the Euclidean norm of the vector</td>
<td>BLAS1$VSNRM2</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDNRM2</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGNRM2</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VSCNRM2</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDZNRM2</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGWNRM2</td>
<td>Double complex (G-floating)</td>
</tr>
<tr>
<td>Generate the elements for a Givens plane rotation</td>
<td>BLAS1$VSROTG</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VDROTG</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGROTG</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCROTG</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZROTG</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWROTG</td>
<td>Double complex (G-floating)</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 2-1 (Cont.) Functions of BLAS Level 1

<table>
<thead>
<tr>
<th>Function</th>
<th>Routine</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply a Givens plane</td>
<td>BLAS1$VSROT</td>
<td>Single</td>
</tr>
<tr>
<td>rotation</td>
<td>BLAS1$VDROT</td>
<td>Double (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VGROT</td>
<td>Double (G-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VCSROT</td>
<td>Single complex</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VZDROT</td>
<td>Double complex (D-floating)</td>
</tr>
<tr>
<td></td>
<td>BLAS1$VWGROT</td>
<td>Double complex (G-floating)</td>
</tr>
</tbody>
</table>

For a detailed description of these routines, refer to the Vector MTH$ Reference Section of this manual.

2.1.1 Using BLAS Level 1

The following sections provide some guidelines for using BLAS Level 1.

2.1.1.1 Memory Overlap

The vector BLAS produces unpredictable results when any element of the input argument shares a memory location with an element of the output argument. (An exception is a special case found in the BLAS1$VxCOPY routines.)

The vector BLAS and the scalar BLAS can yield different results when the input argument overlaps the output array.

2.1.1.2 Round-Off Effects

For some of the routines in BLAS Level 1, the final result is independent of the order in which the operations are performed. However, in other cases (for example, some of the reduction operations), efficiency dictates that the order of operations on a vector machine be different from the natural order of operations. Because round-off errors are dependent upon the order in which the operations are performed, some of the routines will not return results that are bit-for-bit identical to the results obtained by performing the operations in natural order.

Where performance can be increased by the use of a backup data type, this has been done. This is the case for BLAS1$VSNRM2, BLAS1$VSCNRM2, BLAS1$VSROTG, and BLAS1$VCROTG. The use of a backup data type can also yield a gain in accuracy over the scalar BLAS.

2.1.1.3 Underflow and Overflow

In accordance with LINPACK convention, underflow, when it occurs, is replaced by a zero. A system message informs you of overflow. Because the order of operations for some routines is different from the natural order, overflow might not occur at the same array element in both the scalar and vector versions of the routines.

2.1.1.4 Notational Definitions

The vector BLAS (except the BLAS1$VxROTG routines) perform operations on vectors. These vectors are defined in terms of three quantities:

- A vector length, specified as n
- An array or a starting element in an array, specified as x
- An increment or spacing parameter to indicate the distance in number of array elements to skip between successive vector elements, specified as incx
Vector Routines in MTH$ 

2.1 BLAS — Basic Linear Algebra Subroutines Level 1

Suppose \( \mathbf{x} \) is a real array of dimension \( \text{ndim} \), \( \mathbf{n} \) is its vector length, and \( \text{incx} \) is the increment used to access the elements of a vector \( \mathbf{X} \). The elements of vector \( \mathbf{X} = \{X_i, i = 1, \ldots, \mathbf{n}\} \), are stored in \( \mathbf{x} \). If \( \text{incx} \) is greater than or equal to 0, then \( X_i \) is stored in the following location:

\[
x(1 + (i - 1) \times \text{incx})
\]

However, if \( \text{incx} \) is less than 0, then \( X_i \) is stored in the following location:

\[
x(1 + (n - i) \times |\text{incx}|)
\]

It therefore follows that the following condition must be satisfied:

\[
\text{ndim} \geq 1 + (n - 1) \times |\text{incx}|
\]

A positive value for \( \text{incx} \) is referred to as forward indexing, and a negative value is referred to as backward indexing. A value of zero implies that all of the elements of the vector are at the same location, \( x_1 \).

Suppose \( \text{ndim} = 20 \) and \( \mathbf{n} = 5 \). In this case, \( \text{incx} = 2 \) implies that \( X_1, X_2, X_3, X_4, \) and \( X_5 \) are located in array elements \( x_1, x_3, x_5, x_7, \) and \( x_9 \).

If, however, \( \text{incx} \) is negative, then \( X_1, X_2, X_3, X_4, \) and \( X_5 \) are located in array elements \( x_9, x_7, x_5, x_3, \) and \( x_1 \). In other words, when \( \text{incx} \) is negative, the subscript of \( \mathbf{x} \) decreases as \( i \) increases.

For some of the routines in BLAS Level 1, \( \text{incx} = 0 \) is not permitted. In the cases where a zero value for \( \text{incx} \) is permitted, it means that \( x_1 \) is broadcast into each element of the vector \( \mathbf{X} \) of length \( \mathbf{n} \).

You can operate on vectors that are embedded in other vectors or matrices by choosing a suitable starting point of the vector. For example, if \( \mathbf{A} \) is an \( \mathbf{n1} \times \mathbf{n2} \) matrix, its \( j \)-th column is referenced with a length of \( \mathbf{n1} \), starting point \( \mathbf{A}(i,1) \), and increment \( 1 \). Similarly, the \( i \)-th row is referenced with a length of \( \mathbf{n2} \), starting point \( \mathbf{A}(i,1) \), and increment \( \mathbf{n1} \).

2.2 FOLR — First Order Linear Recurrence Routines

The MTH$ FOLR routines provide a vectorized algorithm for the linear recurrence relation. A linear recurrence uses the result of a previous pass through a loop as an operand for subsequent passes through the loop and prevents the vectorization of a loop.

The only error checking performed by the FOLR routines is for a reserved operand.

There are four families of FOLR routines in the MTH$ facility. Each family accepts each of four data types (longword integer, F-floating, D-floating, and G-floating). However, all of the arrays you specify in a single FOLR call must be of the same data type.

For a detailed description of these routines, refer to the Vector MTH$ Reference Section of this manual.

2.2.1 FOLR Routine Name Format

The four families of FOLR routines are as follows:

- MTH$\text{VxFOLR}\_y\_\text{MA}_V15
- MTH$\text{VxFOLR}\_y\_z\_V8
- MTH$\text{VxFOLR}\_y\_\text{MA}_V5
2.2 FOLR — First Order Linear Recurrence Routines

- \( \text{MTH}\$VxFOLRLy_\_z\_V2 \)
  
  where:
  
  \( x = \) J for longword integer, F for F-floating, D for D-floating, or G for G-floating
  
  \( y = \) P for a positive recursion element, or N for a negative recursion element
  
  \( z = \) M for multiplication, or A for addition

  The FOLR entry points end with \_Vn, where \( n \) is an integer between 0 and 15 that denotes the vector registers that the FOLR routine uses. For example, \( \text{MTH}\$VxFOLRLy_\_z\_V8 \) uses vector registers V0 through V8.

  To determine which group of routines you should use, match the task in the left column in Table 2-2 that you need the routine to perform with the method of storage that you need the routine to employ. The point where these two tasks meet shows the FOLR routine you should call.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Save each iteration in an array</th>
<th>Save only last result in a variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication AND addition</td>
<td>( \text{MTH}$VxFOLRLy__z_V2 )</td>
<td>( \text{MTH}$VxFOLRLy__z_V8 )</td>
</tr>
<tr>
<td>Multiplication OR addition</td>
<td>( \text{MTH}$VxFOLRLy__z_V2 )</td>
<td>( \text{MTH}$VxFOLRLy__z_V8 )</td>
</tr>
</tbody>
</table>

2.2.2 Calling a FOLR Routine

Save the contents of V0 through Vn before calling a FOLR routine if you need it after the call. The variable \( n \) can be 2, 5, 8, or 15, depending on the FOLR routine entry point. (The OpenVMS Calling Standard specifies that a called procedure may modify all of the vector registers. The FOLR routines modify only the vector registers V0 through Vn.)

The \( \text{MTH}\$ \) FOLR routines assume that all of the arrays are of the same data type.

2.3 Vector Versions of Existing Scalar Routines

Vector forms of many \( \text{MTH}\$ \) routines are provided to support vectorized compiled applications. Vector versions of key F-floating, D-floating, and G-floating scalar routines employ vector hardware, while maintaining identical results with their scalar counterparts. Many of the scalar algorithms have been redesigned to ensure identical results and good performance for both the vector and scalar versions of each routine. All vectorized routines return bit-for-bit identical results as the scalar versions.

You can call the vector \( \text{MTH}\$ \) routines directly if your program is written in VAX MACRO. If you are a FORTRAN programmer, specify the FORTRAN intrinsic function name only. The VAX FORTRAN-HPO compiler will then determine whether the vector or scalar version of a routine should be used.

2.3.1 Exceptions

You should not attempt to recover from an \( \text{MTH}\$ \) vector exception. After an \( \text{MTH}\$ \) vector exception, the vector routines cannot continue execution, and nonexceptional values might not have been computed.
2.3.2 Underflow Detection

In general, if a vector instruction results in the detection of both a floating overflow and a floating underflow, only the overflow will be signaled.

Some scalar routines check to see if a user has enabled underflow detection. For each of those scalar routines, there are two corresponding vector routines: one that always enables underflow checking and one that never enables underflow checking. (In the latter case, underflows produce a result of zero.) The VAX FORTRAN-HPO compiler always chooses the vector version that does not signal underflows, unless the user specifies the appropriate VAX FORTRAN-HPO compiler switch (the /CHECK=UNDERFLOW qualifier). This ensures that the check is performed but does not impair vector performance for those not interested in underflow detection.

2.3.3 Vector Routine Name Format

Use one of the formats in Table 2-3 to call (from VAX MACRO) a vector math routine that enables underflow signaling. (The E in the routine name means enabled underflow signaling.)

<table>
<thead>
<tr>
<th>Format</th>
<th>Type of Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$VxSAMPLE_E_Ry_Vz</td>
<td>Real valued math routine</td>
</tr>
<tr>
<td>MTH$VCxSAMPLE_E_Ry_Vz</td>
<td>Complex valued math routine</td>
</tr>
<tr>
<td>OTS$SAMPLEq_E_Ry_Vz</td>
<td>Power routine or complex multiply and divide</td>
</tr>
</tbody>
</table>

Use one of the formats in Table 2-4 to call (from VAX MACRO) a vector math routine that does not enable underflow signaling.

<table>
<thead>
<tr>
<th>Format</th>
<th>Type of Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$VxSAMPLE_Ry_Vz</td>
<td>Real valued math routine</td>
</tr>
<tr>
<td>MTH$VCxSAMPLE_Ry_Vz</td>
<td>Complex valued math routine</td>
</tr>
<tr>
<td>OTS$SAMPLEq_Ry_Vz</td>
<td>Power routine or complex multiply/divide</td>
</tr>
</tbody>
</table>

In the preceding formats, the following conventions are used:

- $x$ The letter A (or blank) for F-floating, D for D-floating, G for G-floating.
- $y$ A number between 0 and 11 (inclusive). Ry means that the scalar registers R0 through Ry will be used by the routine SAMPLE. You must save these registers.
- $z$ A number between 0 and 15 (inclusive). Vz means that the vector registers V0 through Vz will be used by the routine SAMPLE. You must save these registers.
- $q$ Two letters denoting the base and power data type, as follows:

  - RR F-floating base raised to an F-floating power
  - RJ F-floating base raised to a longword power
  - DD D-floating base raised to a D-floating power
  - DJ D-floating base raised to a longword power
Vector Routines in MTH$

2.3 Vector Versions of Existing Scalar Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GG</td>
<td>G-floating base raised to a G-floating power</td>
</tr>
<tr>
<td>GJ</td>
<td>G-floating base raised to a longword power</td>
</tr>
<tr>
<td>JJ</td>
<td>Longword base raised to a longword power</td>
</tr>
</tbody>
</table>

2.3.4 Calling a Vector Math Routine

You can call the vector MTH$ routines directly if your program is written in VAX MACRO.

Note

If you are a VAX FORTRAN programmer, do not specify the MTH$ vector routines explicitly. Specify the FORTRAN intrinsic function name only. The VAX FORTRAN-HPO compiler will then determine whether the vector or scalar version of a routine should be used.

In the following examples, keep in mind that vector real arguments are passed in VO, VI, and so on, and vector real results are returned in VO. On the other hand, vector complex arguments are passed in V0 and V1, V2, and V3, and so on. Vector complex results are returned in V0 and V1.

<table>
<thead>
<tr>
<th>Argument Passed Register</th>
<th>Results Returned Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector real arguments</td>
<td>V0, V1, ...</td>
</tr>
<tr>
<td>Vector complex arguments</td>
<td>V0 and V1, V2 and V3, ...</td>
</tr>
</tbody>
</table>

Example 1

The following example shows how to call the vector version of MTH$EXP. Assume that you do not want underflows to be signaled, and you need to use the current contents of all vector and scalar registers after the invocation. Before you can call the vector routine from VAX MACRO, perform the following steps.

1. Find EXP in the column of scalar names in Appendix B to determine:
   - The full vector routine name: MTH$VEXP_R3_V6
   - How the routine is invoked (CALL or JSB): JSB
   - The scalar registers that must be saved: R0 through R3 (as specified by R3 in MTH$VEXP_R3_V6)
   - The vector registers that must be saved: V0 through V6 (as specified by V6 in MTH$VEXP_R3_V6)
   - The vector register(s) used to hold the input argument(s): V0
   - The vector register(s) used to hold the output argument(s): V0
   - If there is a vector version that signals underflow (not needed in this example)

2. Save the scalar registers R0, R1, R2, and R3.
3. Save the vector registers V0, V1, V2, V3, V4, V5, and V6.
4. Save the vector mask register VMR.
5. Save the vector count register VCR.
Vector Routines in MTH$
2.3 Vector Versions of Existing Scalar Routines

6. Load the vector length register VLR.
7. Load the vector register V0 with the argument for MTH$EXP.
8. JSB to MTH$VEXP_R3_V6.
9. Store result in memory.
10. Restore all scalar and vector registers except for V0. (The results of the "call" to MTH$VEXP_R3_V6 are stored in V0.)

The following MACRO program fragment shows this example. Assume that:
• VO through V6 and R0 through R3 have been saved.
• R4 points to a vector of 60 input values.
• R6 points to the location where the results of MTH$VEXP_R3_V6 will be stored.
• R5 contains the stride in bytes.

Note that MTH$VEXP_R3_V6 denotes an F-floating data type because there is no letter between V and E in the routine name. (For further explanation, refer to Section 2.3.3.) The stride (the number of array elements that are skipped) must be a multiple of 4 because each F-floating value requires 4 bytes.

```
MTVLR #60 ; Load VLR
MOVL #4, R5 ; Stride
VLDL (R4), R5, VO ; Load V0 with the actual arguments
JSB G A MTH$VEXP_R3_V6 ; JSB to MTH$VEXP
VSTL VO, (R6), R5 ; Store the results
```

Example 2

The following example demonstrates how to call the vector version of OTS$POWDD with a vector base raised to a scalar power. Before you can call the vector routine from VAX MACRO, perform the following steps.

1. Find POWDD (V5) in the column of scalar names in Appendix B to determine:
   • The full vector routine name: OTS$VPOWDD_R1_V8
   • How the routine is invoked (CALL or JSB): CALL
   • The scalar registers that must be saved: R0 through R1 (as specified by R1 in OTS$VPOWDD_R1_V8)
   • The vector registers that must be saved: V0 through V8 (as specified by V8 in OTS$VPOWDD_R1_V8)
   • The vector register(s) used to hold the input argument(s): V0, R0
   • The vector register(s) used to hold the output argument(s): V0
   • If there is a vector version that signals underflow (not needed in this example)

2. Save the scalar registers R0 and R1.
3. Save the vector registers V0, V1, V2, V3, V4, V5, V6, V7, and V8.
4. Save the vector mask register VMR.
5. Save the vector count register VCR.
6. Load the vector length register VLR.
2.3 Vector Versions of Existing Scalar Routines

7. Load the vector register V0 and the scalar register RO with the arguments for OTSSPOWDD.
8. Call OTSSVPOWDD_R1_V8.
9. Store result in memory.
10. Restore all scalar and vector registers except for V0. (The results of the call to OTSSVPOWDD_R1_V8 are stored in V0.)

The following MACRO program fragment shows how to call OTSSVPOWDD_R1_V8 to compute the result of raising 60 values to the power P. Assume that:

• V0 through V8 and RO and R1 have been saved.
• R4 points to the vector of 60 input base values.
• RO and R1 contain the D-floating value P.
• R6 points to the location where the results will be stored.
• R5 contains the stride.

Note that OTSSVPOWDD_R1_V8 raises a D-floating base to a D-floating power, which you determine from the DD in the routine name. (For further explanation, refer to Section 2.3.3.) The stride (the number of array elements that are skipped) must be a multiple of 8 because each D-floating value requires 8 bytes.

2.4 Fast-Vector Math Routines

This section describes the fast-vector math routines that offer significantly higher performance at the cost of slightly reduced accuracy when compared with corresponding standard vector math routines. Also note that some fast-vector math routines have restricted argument domains.

When you specify the compile command qualifiers /VECTOR and /MATH_LIBRARY=FAST, VAX FORTRAN-HPO Version 1.2 selects the appropriate fast-vector math routine, if one exists. The default is /MATH_LIBRARY=ACCURATE. You must specify the /G_FLOATING compile qualifier in conjunction with the /MATH_LIBRARY=FAST and /VECTOR qualifiers to access the G_floating versions from VAX FORTRAN-HPO. See the VAX FORTRAN-HPO V1.2 Release Notes for more information.

You can call these routines from VAX MACRO using the standard calling method. The math function names, together with corresponding entry points of the fast-vector math routines, are listed in Table 2-5.
Table 2-5 Fast-Vector Math Routines

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Data Type</th>
<th>Call or JSB</th>
<th>Vector Input Registers</th>
<th>Vector Output Registers</th>
<th>Vector Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATAN</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYATAN_R0_V3</td>
</tr>
<tr>
<td>DATAN</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDATAN_R0_V5</td>
</tr>
<tr>
<td>GATAN</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGATAN_R0_V5</td>
</tr>
<tr>
<td>ATAN2</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VYATAN2_R0_V5</td>
</tr>
<tr>
<td>DATAN2</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VYDATAN2_R0_V5</td>
</tr>
<tr>
<td>GATAN2</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VYGATAN2_R0_V5</td>
</tr>
<tr>
<td>COS</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYCOS_R0_V3</td>
</tr>
<tr>
<td>DCOS</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDCOS_R0_V3</td>
</tr>
<tr>
<td>GCOS</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGOS_R0_V3</td>
</tr>
<tr>
<td>EXP</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYEXP_R0_V4</td>
</tr>
<tr>
<td>DEXP</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDEXP_R0_V6</td>
</tr>
<tr>
<td>GEXP</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGEXP_R0_V6</td>
</tr>
<tr>
<td>LOG</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYLOG_R0_V5</td>
</tr>
<tr>
<td>DLOG</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDLOG_R0_V5</td>
</tr>
<tr>
<td>GLOG</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGLOG_R0_V5</td>
</tr>
<tr>
<td>LOG10</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYLOG10_R0_V5</td>
</tr>
<tr>
<td>DLOG10</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDLOG10_R0_V5</td>
</tr>
<tr>
<td>GLOG10</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGLOG10_R0_V5</td>
</tr>
<tr>
<td>SIN</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYSIN_R0_V3</td>
</tr>
<tr>
<td>DSIN</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDSIN_R0_V3</td>
</tr>
<tr>
<td>GSIN</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGSIN_R0_V3</td>
</tr>
<tr>
<td>SQRT</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYSQRT_R0_V4</td>
</tr>
<tr>
<td>DSQRT</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDSQRT_R0_V4</td>
</tr>
<tr>
<td>GSQRT</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGQRT_R0_V4</td>
</tr>
<tr>
<td>TAN</td>
<td>F_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYTAN_R0_V3</td>
</tr>
<tr>
<td>DTAN</td>
<td>D_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYDTAN_R0_V3</td>
</tr>
<tr>
<td>GTAN</td>
<td>G_floating</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VYGATAN_R0_V3</td>
</tr>
<tr>
<td>POWRR(X**Y)</td>
<td>F_floating</td>
<td>CALL</td>
<td>V0, R0</td>
<td>V0</td>
<td>OTS$VYPOWRR_R1_V4</td>
</tr>
<tr>
<td>POWDD(X**Y)</td>
<td>D_floating</td>
<td>CALL</td>
<td>V0, R0</td>
<td>V0</td>
<td>OTS$VYPOWDD_R1_V8</td>
</tr>
<tr>
<td>POWGG(X**Y)</td>
<td>G_floating</td>
<td>CALL</td>
<td>V0, R0</td>
<td>V0</td>
<td>OTS$VYPOGG_R1_V9</td>
</tr>
</tbody>
</table>

2.4.1 Exception Handling

The fast-vector math routines signal all errors except floating underflow. No intermediate calculations result in exceptions. To optimize performance, the following message signals all errors:

%SYSTEM-F-VARITH, vector arithmetic fault
2.4.2 Special Restrictions On Input Arguments

The special restrictions listed in Table 2–6 apply only to fast-vector routines SIN, COS, and TAN. The standard vector routines handle the full range of VAX floating point numbers.

Table 2–6 Input Argument Restrictions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Input Argument Domain (in Radians)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>~(-6746518783.0, 6746518783.0)</td>
</tr>
<tr>
<td>COS</td>
<td>~(-6746518783.0, 6746518783.0)</td>
</tr>
<tr>
<td>TAN</td>
<td>~(-3373259391.5, 3373259391.5)</td>
</tr>
</tbody>
</table>

If the application program uses arguments outside of the listed domain, the routine returns the following error message:

%SYSTEM-F-VARITH, vector arithmetic fault

If the application requires argument values beyond the listed limits, use the corresponding standard vector math routine.

2.4.3 Accuracy

The fast-vector math routines do not guarantee the same results as those obtained with the corresponding standard vector math routines. Calls to the fast-vector routines generally yield results that are different from the scalar and original vector MTH$ library routines. The typical maximum error is a 2-LSB (Least Significant Bit) error for the F_floating routines and a 4-LSB error for the D_floating and G_floating routines. This generally corresponds to a difference in the 6th significant decimal digit for the F_floating routines, the 15th digit for D_floating, and the 14th digit for G_floating.

2.4.4 Performance

The fast-vector math routines generally provide performance improvements over the standard vector routines ranging from 15 to 300 percent, depending on the routines called and input arguments to the routines. The overall performance improvement using fast-vector math routines in a typical user application will increase, but not at the same level as the routines themselves. You should do performance and correctness testing of your application using both the fast-vector and the standard vector math routines before deciding which to use for your application.
The Scalar MTH$ Reference Section provides detailed descriptions of the scalar routines provided by the OpenVMS RTL Mathematics (MTH$) Facility.
MTH$\times$ACOS—Arc Cosine of Angle Expressed in Radians

Given the cosine of an angle, the Arc Cosine of Angle Expressed in Radians routine returns that angle (in radians).

Format

MTH$ACOS$ cosine  
MTH$DACOS$ cosine  
MTH$GACOS$ cosine  

Each of the above three formats accepts one of the floating-point types as input.

JSB Entries

MTH$ACOS_R4$  
MTH$DACOS_R7$  
MTH$GACOS_R7$  

Each of the above three JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point  
type F_floating, D_floating, G_floating  
access write only  
mechanism by value  

Angle in radians. The angle returned will have a value in the range $0 \leq angle \leq \pi$

MTH$ACOS$ returns an F-floating number. MTH$DACOS$ returns a D-floating number. MTH$GACOS$ returns a G-floating number.

Arguments

cosine  
OpenVMS usage floating_point  
type F_floating, D_floating, G_floating  
access read only  
mechanism by reference  

The cosine of the angle whose value (in radians) is to be returned. The cosine argument is the address of a floating-point number that is this cosine. The absolute value of cosine must be less than or equal to 1. For MTH$ACOS$, cosine specifies an F-floating number. For MTH$DACOS$, cosine specifies a D-floating number. For MTH$GACOS$, cosine specifies a G-floating number.
MTH$ACOS

Description
The angle in radians whose cosine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Cosine</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>π/2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>π</td>
</tr>
<tr>
<td>0 &lt; X &lt; 1</td>
<td>zATAN(zSQRT(1 - X^2)/X), where zATAN and zSQRT are the Math Library arc tangent and square root routines, respectively, of the appropriate data type</td>
</tr>
<tr>
<td>-1 &lt; X &lt; 0</td>
<td>zATAN(zSQRT(1 - X^2)/X) + π</td>
</tr>
<tr>
<td>1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HACOS.

Condition Values Signaled

SS$_ROPRAND
Reserved operand. The MTH$ACOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$INVARGMAT
Invalid argument. The absolute value of cosine is greater than 1. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

Examples

```
1. 100  
!  This BASIC program demonstrates the use of ! MTH$ACOS.  
!  
EXTERNAL REAL FUNCTION MTH$ACOS  
DECLARE REAL COS_VALUE, ANGLE  
300 INPUT "Cosine value between -1 and +1 "; COS_VALUE  
400 IF (COS_VALUE < -1) OR (COS_VALUE > 1) THEN PRINT "Invalid cosine value"  
      GOTO 300  
500 ANGLE = MTH$ACOS( COS_VALUE )  
      PRINT "The angle with that cosine is "; ANGLE; " radians"  
32767 END
```
This BASIC program prompts for a cosine value and determines the angle that has that cosine. The output generated by this program is as follows:

```
$ RUN ACOS
Cosine value between -1 and +1 ? .5
The angle with that cosine is 1.0472 radians
```

This PASCAL program prompts for a cosine value and determines the angle that has that cosine. The output generated by this program is as follows:

```
$ RUN ACOS
Cosine value between -1 and +1: .5
The angle with that cosine is 1.04720E+00 radians
```
MTH$ACOSD

MTH$ACOSD—Arc Cosine of Angle Expressed in Degrees

Given the cosine of an angle, the Arc Cosine of Angle Expressed in Degrees routine returns that angle (in degrees).

Format

MTH$ACOSD cosine
MTH$DACOSD cosine
MTH$GACOSD cosine

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$ACOSD_R4
MTH$DACOSD_R7
MTH$GACOSD_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Angle in degrees. The angle returned will have a value in the range

\[ 0 \leq \text{angle} \leq 180 \]

MTH$ACOSD returns an F-floating number. MTH$DACOSD returns a D-floating number. MTH$GACOSD returns a G-floating number.

Arguments

cosine

OpenVMS usage floating_point
type F_floating, G_floating, D_floating
access read only
mechanism by reference

Cosine of the angle whose value (in degrees) is to be returned. The cosine argument is the address of a floating-point number that is this cosine. The absolute value of cosine must be less than or equal to 1. For MTH$ACOSD, cosine specifies an F-floating number. For MTH$DACOSD, cosine specifies a D-floating number. For MTH$GACOSD, cosine specifies a G-floating number.
Description

The angle in degrees whose cosine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Cosine</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>180</td>
</tr>
<tr>
<td>0 &lt; X &lt; 1</td>
<td>( z\text{ATAND}(z\text{SQRT}(1 - X^2)/X) ), where ( z\text{ATAND} ) and ( z\text{SQRT} ) are the Math Library arc tangent and square root routines, respectively, of the appropriate data type</td>
</tr>
<tr>
<td>-1 &lt; X &lt; 0</td>
<td>( z\text{ATAND}(z\text{SQRT}(1 - X^2)/X) + 180 )</td>
</tr>
<tr>
<td>1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$_\text{HACOSD}$.

Condition Values Signaled

**SS$_\text{$_\text{ROPRAND}}$**

Reserved operand. The MTH$_\text{xACOSD}$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_\text{$_\text{INVARGMAT}}$**

Invalid argument. The absolute value of cosine is greater than 1. LIB$_\text{SIGNAL}$ copies the floating-point reserved operand to the mechanism argument vector CHF$_L$ _MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$_L$ _MCH_SAVR0/R1.

Example

```pascal
PROGRAM ACOSD(INPUT,OUTPUT);
{ + }
{ This PASCAL program demonstrates the use of }
{ MTH$ACOSD. }
{ - }
FUNCTION MTH$ACOSD(COS : REAL) : REAL; EXTERN;
VAR
  COSINE : REAL;
  RET_STATUS : REAL;
BEGIN
  COSINE := 0.5;
  RET_STATUS := MTH$ACOSD(COSINE);
  WRITELN (‘The angle, in degrees, is: ’, RET_STATUS);
END.
```

MTH-7
The output generated by this PASCAL example program is as follows:

The angle, expressed in degrees, is: 6.00000E+01
MTH$\times$ASIN—Arc Sine in Radians

Given the sine of an angle, the Arc Sine in Radians routine returns that angle (in radians).

**Format**

- MTH$\times$ASIN sine
- MTH$\times$DASIN sine
- MTH$\times$GASIN sine

Each of the above formats accepts one of the floating-point types as input.

**JSB Entries**

- MTH$\times$ASIN_R4
- MTH$\times$DASIN_R7
- MTH$\times$GASIN_R7

Each of the above JSB entries accepts one of the floating-point types as input.

**Returns**

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Angle in radians. The angle returned will have a value in the range

\[-\pi/2 \leq \text{angle} \leq \pi/2\]

MTH$\times$ASIN returns an F-floating number. MTH$\times$DASIN returns a D-floating number. MTH$\times$GASIN returns a G-floating number.

**Arguments**

- **sine**

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

The sine of the angle whose value (in radians) is to be returned. The sine argument is the address of a floating-point number that is this sine. The absolute value of sine must be less than or equal to 1. For MTH$\times$ASIN, sine specifies an F-floating number. For MTH$\times$DASIN, sine specifies a D-floating number. For MTH$\times$GASIN, sine specifies a G-floating number.
Description

The angle in radians whose sine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Sine</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>( \pi/2 )</td>
</tr>
<tr>
<td>-1</td>
<td>(-\pi/2)</td>
</tr>
<tr>
<td>0 &lt;</td>
<td>X</td>
</tr>
<tr>
<td>1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under \text{MTH}\$\_\text{HASIN}.

Condition Values Signaled

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{SS}$_\text{ROPRAND}</td>
<td>Reserved operand. The \text{MTH}$_\text{ASIN} routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
<tr>
<td>\text{MTH}$_\text{INVARGMAT}</td>
<td>Invalid argument. The absolute value of sine is greater than 1. \text{LIB}$\text{SIGNAL} copies the floating-point reserved operand to the mechanism argument vector \text{CHF}$_\text{MCH}_\text{SAVR0/R1}. The result is the floating-point reserved operand unless you have written a condition handler to change \text{CHF}$_\text{MCH}_\text{SAVR0/R1}.</td>
</tr>
</tbody>
</table>
MTH$xASIND—Arc Sine in Degrees

Given the sine of an angle, the Arc Sine in Degrees routine returns that angle (in degrees).

Format

MTH$ASIND sine
MTH$DASIND sine
MTH$GASIND sine

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$ASIND_R4
MTH$DASIND_R7
MTH$GASIND_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Angle in degrees. The angle returned will have a value in the range

\[-90 \leq \text{angle} \leq 90\]

MTH$ASIND returns an F-floating number. MTH$DASIND returns a D-floating number. MTH$GASIND returns a G-floating number.

Arguments

sine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Sine of the angle whose value (in degrees) is to be returned. The sine argument is the address of a floating-point number that is this sine. The absolute value of sine must be less than or equal to 1. For MTH$ASIND, sine specifies an F-floating number. For MTH$DASIND, sine specifies a D-floating number. For MTH$GASIND, sine specifies a G-floating number.
The angle in degrees whose sine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Sine</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>-1</td>
<td>-90</td>
</tr>
</tbody>
</table>

\[ \text{zATAND}(X / \text{zSQRT}(1 - X^2)) \], where zATAND and zSQRT are the Math Library arc tangent and square root routines, respectively, of the appropriate data type.

1 < |X| The error MTH$_{\text{INVARGMAT}}$ is signaled

The routine description for the H-float version of this routine is listed alphabetically under MTH$_{\text{HASIND}}$.

**Condition Values Signaled**

- **SS$_{\text{ROPRAND}}$**
  
  Reserved operand. The MTH$_{\text{xASIND}}$ routine encountered a floating point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$_{\text{INVARGMAT}}$**
  
  Invalid argument. The absolute value of sine is greater than 1. LIB$_{\text{SIGNAL}}$ copies the floating-point reserved operand to the mechanism argument vector CHF$L\_MCH\_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\_MCH\_SAVR0/R1.
MTH$ATAN—Arc Tangent in Radians

Given the tangent of an angle, the Arc Tangent in Radians routine returns that angle (in radians).

Format

```
MTH$ATAN  tangent
MTH$DATAN  tangent
MTH$GATAN  tangent
```

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

```
MTH$ATAN_R4
MTH$DATAN_R7
MTH$GATAN_R7
```

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

```
OpenVMS usage   floating_point
type             F_floating, D_floating, G_floating
access           write only
mechanism        by value

Angle in radians. The angle returned will have a value in the range

\[-\pi/2 \leq \text{angle} \leq \pi/2\]
```

MTH$ATAN returns an F-floating number. MTH$DATAN returns a D-floating number. MTH$GATAN returns a G-floating number.

Arguments

```
tangent
OpenVMS usage   floating_point
type             F_floating, D_floating, G_floating
access           read only
mechanism        by reference
```

The tangent of the angle whose value (in radians) is to be returned. The tangent argument is the address of a floating-point number that is this tangent. For MTH$ATAN, tangent specifies an F-floating number. For MTH$DATAN, tangent specifies a D-floating number. For MTH$GATAN, tangent specifies a G-floating number.
Description

In radians, the computation of the arc tangent function is based on the following identities:

\[
\arctan(X) = X - X^3/3 + X^5/5 - X^7/7 + \ldots
\]
\[
\arctan(X) = X + X \cdot Q(X^2),
\text{ where } Q(Y) = -Y^3/3 + Y^5/5 - Y^7/7 + \ldots
\]
\[
\arctan(X) = X \cdot P(X^2),
\text{ where } P(Y) = 1 - Y^3/3 + Y^5/5 - Y^7/7 + \ldots
\]
\[
\arctan(X) = \pi/2 - \arctan(1/X)
\]
\[
\arctan(X) = \arctan(A) + \arctan((X - A)/(1 + A \cdot X))
\text{ for any real } A
\]

The angle in radians whose tangent is \(X\) is computed as:

<table>
<thead>
<tr>
<th>Value of (X)</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0 \leq X \leq 3/32)</td>
<td>(X + X \cdot Q(X^2))</td>
</tr>
<tr>
<td>(3/32 &lt; X \leq 11)</td>
<td>(\text{TAN}(A) + V \cdot (P(V^2))), where (A) and (\text{TAN}(A)) are chosen by table lookup and (V = (X - A)/(1 + A \cdot X))</td>
</tr>
<tr>
<td>(11 &lt; X)</td>
<td>(\pi/2 - W \cdot (P(W^2))) where (W = 1/X)</td>
</tr>
<tr>
<td>(X &lt; 0)</td>
<td>(-z\text{TAN}(</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HATAN.

Condition Value Signaled

\text{SSS$_{\text{ROPRAND}}$}

Reserved operand. The MTH$\text{xATAN}$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
**MTH$ATAND—Arc Tangent in Degrees**

Given the tangent of an angle, the Arc Tangent in Degrees routine returns that angle (in degrees).

**Format**

MTH$ATAND tangent  
MTH$DATAND tangent  
MTH$GATAND tangent

Each of the above formats accepts one of the floating-point types as input.

**JSB Entries**

MTH$ATAND_R4  
MTH$DATAND_R7  
MTH$GATAND_R7

Each of the above JSB entries accepts one of the floating-point types as input.

**Returns**

OpenVMS usage: floating_point  
Type: Floating, D_floating, G_floating  
Access: write only  
Mechanism: by value

Angle in degrees. The angle returned will have a value in the range

\[-90 \leq \text{angle} \leq 90\]

MTH$ATAND returns an F-floating number. MTH$DATAND returns a D-floating number. MTH$GATAND returns a G-floating number.

**Arguments**

**tangent**

OpenVMS usage: floating_point  
Type: F_floating, D_floating, G_floating  
Access: read only  
Mechanism: by reference

The tangent of the angle whose value (in degrees) is to be returned. The **tangent** argument is the address of a floating-point number that is this tangent. For MTH$ATAND, **tangent** specifies an F-floating number. For MTH$DATAND, **tangent** specifies a D-float number. For MTH$GATAND, **tangent** specifies a G-floating number.
Description

The computation of the arc tangent function is based on the following identities:

\[
\arctan(X) = \frac{180}{\pi} \times (X - X^3/3 + X^5/5 - X^7/7 + \ldots)
\]

\[
\arctan(X) = 64 \times X + X \times Q(X^2),
\]

where \( Q(Y) = \frac{180}{\pi} \times [(1 - 64 \times \pi/180)] - Y/3 + Y^2/5 - Y^3/7 + Y^4/9 \)

\[
\arctan(X) = X \times P(X^2),
\]

where \( P(Y) = \frac{180}{\pi} \times [1 - Y/3 + Y^2/5 - Y^3/7 + Y^4/9] \ldots \)

\[
\arctan(X) = 90 - \arctan(1/X)
\]

\[
\arctan(X) = \arctan(A) + \arctan((X - A)/(1 + A \times X))
\]

The angle in degrees whose tangent is \( X \) is computed as:

<table>
<thead>
<tr>
<th>Tangent</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X \leq 3/32 )</td>
<td>( 64 \times X + X \times Q(X^2) )</td>
</tr>
<tr>
<td>( 3/32 &lt; X \leq 11 )</td>
<td>( \text{ATAND}(A) + V \times P(V^2) ), where ( A ) and ( \text{ATAND}(A) ) are chosen by table lookup and ( V = (X - A)/(1 + A \times X) )</td>
</tr>
<tr>
<td>( 11 &lt; X )</td>
<td>( 90 - W \times (P(W^2)), where W = 1/X )</td>
</tr>
<tr>
<td>( X &lt; 0 )</td>
<td>( -\text{zATAND}(1/X) )</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH\$HATAND.

Condition Value Signaled

**SS\$ _ROPRAND**

Reserved operand. The MTH\$xATAND routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$\times ATAN2—Arc Tangent in Radians with Two Arguments

Given sine and cosine, the Arc Tangent in Radians with Two Arguments routine returns the angle (in radians) whose tangent is given by the quotient of sine and cosine (sine/cosine).

Format

MTH$ATAN2 sine , cosine
MTH$DATAN2 sine , cosine
MTH$GATAN2 sine , cosine

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Angle in radians. MTH$ATAN2 returns an F-floating number. MTH$DATAN2 returns a D-floating number. MTH$GATAN2 returns a G-floating number.

Arguments

sine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Dividend. The sine argument is the address of a floating-point number that is this dividend. For MTH$ATAN2, sine specifies an F-floating number. For MTH$DATAN2, sine specifies a D-floating number. For MTH$GATAN2, sine specifies a G-floating number.

cosine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Divisor. The cosine argument is the address of a floating-point number that is this divisor. For MTH$ATAN2, cosine specifies an F-floating number. For MTH$DATAN2, cosine specifies a D-floating number. For MTH$GATAN2, cosine specifies a G-floating number.
MTH$\times$ATAN2

Description

The angle in radians whose tangent is $Y/X$ is computed as follows, where $f$ is defined in the description of MTH$\times$COSH.

<table>
<thead>
<tr>
<th>Value of Input Arguments</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X = 0$ or $Y/X &gt; 2^{(f+1)}$</td>
<td>$\pi/2 \times (\text{sign}Y)$</td>
</tr>
<tr>
<td>$X &gt; 0$ and $Y/X \leq 2^{(f+1)}$</td>
<td>$\arctan(Y/X)$</td>
</tr>
<tr>
<td>$X &lt; 0$ and $Y/X \leq 2^{(f+1)}$</td>
<td>$\pi \times (\text{sign}Y) + \arctan(Y/X)$</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$\times$HATAN2.

Condition Values Signaled

SS$_\times$ROPRAND

Reserved operand. The MTH$\times$ATAN2 routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_\times$INVARGMAT

Invalid argument. Both cosine and sine are zero. LIB$\times$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L\_MCH\_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\_MCH\_SAVR0/R1.
MTH$ATAND2—Arc Tangent in Degrees with Two Arguments

Given sine and cosine, the Arc Tangent in Degrees with Two Arguments routine returns the angle (in degrees) whose tangent is given by the quotient of sine and cosine (sine/cosine).

Format

MTH$ATAND2 sine ,cosine
MTH$DATAND2 sine ,cosine
MTH$GATAND2 sine ,cosine

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Angle (in degrees). MTH$ATAND2 returns an F-floating number. MTH$DATAND2 returns a D-floating number. MTH$GATAND2 returns a G-floating number.

Arguments

sine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Dividend. The sine argument is the address of a floating-point number that is this dividend. For MTH$ATAND2, sine specifies an F-floating number. For MTH$DATAND2, sine specifies a D-floating number. For MTH$GATAND2, sine specifies a G-floating number.

cosine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Divisor. The cosine argument is the address of a floating-point number that is this divisor. For MTH$ATAND2, cosine specifies an F-floating number. For MTH$DATAND2, cosine specifies a D-floating number. For MTH$GATAND2, cosine specifies a G-floating number.
MTH$ATAND2

Description

The angle in degrees whose tangent is \( Y/X \) is computed below and where \( f \) is defined in the description of MTH$zCOSH.

<table>
<thead>
<tr>
<th>Value of Input Arguments</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X = 0 ) or ( Y/X &gt; 2^{(f+1)} )</td>
<td>( 90 \times (\text{sign}Y) )</td>
</tr>
<tr>
<td>( X &gt; 0 ) and ( Y/X \leq 2^{(f+1)} )</td>
<td>( z\text{ATAND}(Y/X) )</td>
</tr>
<tr>
<td>( X &lt; 0 ) and ( Y/X &lt; 2^{(f+1)} )</td>
<td>( 180 \times (\text{sign}Y) + z\text{ATAND}(Y/X) )</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HATAND2.

Condition Values Signaled

MTH$_{INVARGMAT}$

Invalid argument. Both cosine and sine are zero. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVRO/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVRO/R1.

SS$_{ROPRAND}$

Reserved operand. The MTH$xATAND2 routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
**MTH$ATANH—Hyperbolic Arc Tangent**

Given the hyperbolic tangent of an angle, the Hyperbolic Arc Tangent routine returns the hyperbolic arc tangent of that angle.

**Format**

- MTH$ATANH hyperbolic-tangent
- MTH$DATANH hyperbolic-tangent
- MTH$GATANH hyperbolic-tangent

Each of the above formats accepts one of the floating-point types as input.

**Returns**

- OpenVMS usage floating_point
- type F_floating, D_floating, G_floating
- access write only
- mechanism by value

The hyperbolic arc tangent of `hyperbolic-tangent`. MTH$ATANH returns an F-floating number. MTH$DATANH returns a D-floating number. MTH$GATANH returns a G-floating number.

**Arguments**

- **hyperbolic-tangent**
  - OpenVMS usage floating_point
  - type F_floating, D_floating, G_floating
  - access read only
  - mechanism by reference

Hyperbolic tangent of an angle. The `hyperbolic-tangent` argument is the address of a floating-point number that is this hyperbolic tangent. For MTH$ATANH, `hyperbolic-tangent` specifies an F-floating number. For MTH$DATANH, `hyperbolic-tangent` specifies a D-floating number. For MTH$GATANH, `hyperbolic-tangent` specifies a G-floating number.

**Description**

The hyperbolic arc tangent function is computed as follows:

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>X</td>
</tr>
<tr>
<td>(</td>
<td>X</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HATANH.
Condition Values Signaled

**SS$_$ROPRAND**

Reserved operand. The MTH$xATANH routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_$INVARGMAT**

Invalid argument: $|X| \geq 1$. LIB$SIGNAL$ copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_ SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_ SAVR0/R1.
MTH$CABS—Complex Absolute Value

The Complex Absolute Value routine returns the absolute value of a complex number \((r, i)\).

**Format**

- \(\text{MTH$CABS} \ \text{complex-number}\)
- \(\text{MTH$CDABS} \ \text{complex-number}\)
- \(\text{MTH$CGABS} \ \text{complex-number}\)

Each of the above three formats accepts one of the three floating-point complex types as input.

**Returns**

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

The absolute value of a complex number. \(\text{MTH$CABS}\) returns an F-floating number. \(\text{MTH$CDABS}\) returns a D-floating number. \(\text{MTH$CGABS}\) returns a G-floating number.

**Arguments**

\text{complex-number}

OpenVMS usage complex_number
type F_floating complex, D_floating complex, G_floating complex
access read only
mechanism by reference

A complex number \((r, i)\), where \(r\) and \(i\) are both floating-point complex values. The \text{complex-number} argument is the address of this complex number. For \(\text{MTH$CABS}\), \text{complex-number} specifies an F-floating complex number. For \(\text{MTH$CDABS}\), \text{complex-number} specifies a D-floating complex number. For \(\text{MTH$CGABS}\), \text{complex-number} specifies a G-floating complex number.

**Description**

The complex absolute value is computed as follows, where \(\text{MAX}\) is the larger of \(|r|\) and \(|i|\), and \(\text{MIN}\) is the smaller of \(|r|\) and \(|i|\).

\[
\text{result} = \text{MAX} \cdot \sqrt{(\text{MIN}/\text{MAX})^2 + 1}
\]
Condition Values Signaled

**SS$_ROPRAND**
Reserve operand. The MTH$CxABS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_FLOOVEMAT**
Floating-point overflow in Math Library when both r and i are large.

Examples

1. C
   This FORTRAN example forms the absolute value of an F-floating complex number using MTH$CABS and the FORTRAN random number generator RAN.
   Declare Z as a complex value and MTH$CABS as a REAL*4 value.
   MTH$CABS will return the absolute value of Z: Z_NEW = MTH$CABS(Z).
   
   ```fortran
   COMPLEX Z
   COMPLEX CMPLX
   REAL*4 Z_NEW, MTH$CABS
   INTEGER M
   M = 1234567
   
   Z = CMPLX (RAN (M) , RAN (M))
   
   Z is a complex number (r,i) with real part "r" and imaginary part "i".
   TYPE *, ' The complex number z is',Z
   TYPE *, ' It has real part',REAL(Z),'and imaginary part',AIMAG(Z)
   END
   
   Z_NEW = MTH$CABS(Z)
   TYPE *, ' The complex absolute value of',Z,' is',Z_NEW
   END
   
   This example uses an F-floating complex number for complex-number. The output of this FORTRAN example is as follows:
   The complex number z is (0.8535407,0.2043402)
   It has real part 0.8535407 and imaginary part 0.2043402
   The complex absolute value of (0.8535407,0.2043402) is 0.8776597
   ```
This FORTRAN example forms the absolute value of a G-floating complex number using MTH$CGABS and the FORTRAN random number generator RAN.

Declare Z as a complex value and MTH$CGABS as a REAL*8 value. MTH$CGABS will return the absolute value of Z: Z_NEW = MTH$CGABS(Z).

```
COMPLEX*16 Z
REAL*8 Z_NEW,MTH$CGABS
```

Generate a random complex number with the FORTRAN generic CMPLX.

```
Z = (12.34567890123,45.536376385345)
```

Compute the complex absolute value of Z.

```
Z_NEW = MTH$CGABS(Z)
```

This FORTRAN example uses a G-floating complex number for complex-number. Because this example uses a G-floating number, it must be compiled as follows:

```
$ FORTRAN/G MTHEX.FOR
```

Notice the difference in the precision of the output generated:

The complex number z is (12.3456789012300,45.5363763853450)
The complex absolute value of (12.3456789012300,45.5363763853450) is 47.1802645376230
MTH$CCOS—Cosine of a Complex Number (F-Floating Value)

The Cosine of a Complex Number (F-Floating Value) routine returns the cosine of a complex number as an F-floating value.

Format

MTH$CCOS complex-number

Returns

OpenVMS usage complex_number
type F_floating complex
access write only
mechanism by value

The complex cosine of the complex input number. MTH$CCOS returns an F-floating complex number.

Arguments

complex-number
OpenVMS usage complex_number
type F_floating complex
access read only
mechanism by reference

A complex number (r,i) where r and i are floating-point numbers. The complex-number argument is the address of this complex number. For MTH$CCOS, complex-number specifies an F-floating complex number.

Description

The complex cosine is calculated as follows:

\[ \text{result} = (\cos(r) \cdot \cosh(i), -\sin(r) \cdot \sinh(i)) \]

The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$CxCOS.

Condition Values Signaled

SS$_$ROPRAND

Reserved operand. The MTH$CCOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_$FLOOVEMAT

Floating-point overflow in Math Library: the absolute value of i is greater than about 88.029 for F-floating values.
Example

This FORTRAN example forms the complex cosine of an F-floating complex number using MTH$CCOS and the FORTRAN random number generator RAN.

Declare Z and MTH$CCOS as complex values. MTH$CCOS will return the cosine value of Z:

\[
Z \leftarrow Z_{\text{NEW}} = \text{MTH$CCOS}(Z)
\]

```fortran
COMPLEX Z,Z_NEW,MTH$CCOS
COMPLEX CMPLX
INTEGER M
M = 1234567

Z = CMPLX (RAN (M), RAN (M))

Z is a complex number \((r,i)\) with real part \("r"\) and imaginary part \("i"\).

Z_NEW = MTH$CCOS(Z)

END
```

This FORTRAN example demonstrates the use of MTH$CCOS, using the MTH$CCOS entry point. The output of this program is as follows:

The complex number \(z\) is \((0.8535407,0.2043402)\)
It has real part 0.8535407 and imaginary part 0.2043402
The complex cosine value of \((0.8535407,0.2043402)\) is \((0.6710899,-0.1550672)\)

MTH-27
MTH$CxCOS—Cosine of a Complex Number

The Cosine of a Complex Number routine returns the cosine of a complex number.

Format

MTH$CDCOS  complex-cosine ,complex-number
MTH$CGCOS  complex-cosine ,complex-number

Each of the above formats accepts one of the floating-point complex types as input.

Returns

None.

Arguments

complex-cosine
OpenVMS usage  complex_number
  type     D_floating complex, G_floating complex
  access   write only
  mechanism by reference

Complex cosine of the complex-number. The complex cosine routines that have D-floating and G-floating complex input values write the address of the complex cosine into the complex-cosine argument. For MTH$CDCOS, the complex-cosine argument specifies a D-floating complex number. For MTH$CGCOS, the complex-number argument specifies a G-floating complex number.

complex-number
OpenVMS usage  complex_number
  type     D_floating complex, G_floating complex
  access   read only
  mechanism by reference

A complex number (r,i) where r and i are floating-point numbers. The complex-number argument is the address of this complex number. For MTH$CDCOS, complex-number specifies a D-floating complex number. For MTH$CGCOS, complex-number specifies a G-floating complex number.

Description

The complex cosine is calculated as follows:

\[ \text{result} = (\cos(r) \cdot \cosh(i), -\sin(r) \cdot \sinh(i)) \]
Condition Values Signaled

**SS$_$ROPRAND**

Reserved operand. The MTH$CxCOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_$FLOOVEMAT**

Floating-point overflow in Math Library: the absolute value of i is greater than about 88.029 for F-floating and D-floating values, or greater than 709.089 for G-floating values.

Example

```fortran
C+  This FORTRAN example forms the complex cosine of a D-floating complex number using MTH$CDCOS and the FORTRAN random number generator RAN.
C+  
C+  Declare Z and MTH$CDCOS as complex values.
C+  MTH$CDCOS will return the cosine value of Z: Z_NEW = MTH$CDCOS(Z)
C-

COMPLEX*16 Z,Z_NEW,MTH$CDCOS
COMPLEX*16 DCMPLX
INTEGER M
M = 1234567

C+  Generate a random complex number with the FORTRAN generic DCMPLX.
C-

Z = DCMPLX(RAN(M),RAN(M))

C+  Z is a complex number (r,i) with real part "r" and imaginary part "i".
C-

TYPE *, ' The complex number z is',z
TYPE *, ' '

C+  Compute the complex cosine value of Z.
C-

Z_NEW = MTH$CDCOS(Z)
TYPE *, ' The complex cosine value of',z,' is',Z_NEW
END
```

This FORTRAN example program demonstrates the use of MTH$CxCOS, using the MTH$CDCOS entry point. Notice the high precision of the output generated:

The complex number z is (0.8535407185554504,0.2043401598930359)
The complex cosine value of (0.8535407185554504,0.2043401598930359) is (0.671089928500762,-0.1550672019621661)
The Complex Exponential (F-Floating Value) routine returns the complex exponential of a complex number as an F-floating value.

**Format**

```
MTH$CEXP  complex-number
```

**Returns**

OpenVMS usage: complex_number  
Type: F_floating complex  
Access: read only  
Mechanism: by reference  

Complex exponential of the complex input number. MTH$CEXP returns an F-floating complex number.

**Arguments**

**complex-number**

OpenVMS usage: complex_number  
Type: F_floating complex  
Access: read only  
Mechanism: by reference  

Complex number whose complex exponential is to be returned. This complex number has the form \( (r,i) \), where \( r \) is the real part and \( i \) is the imaginary part. The `complex-number` argument is the address of this complex number. For MTH$CEXP, `complex-number` specifies an F-floating number.

**Description**

The complex exponential is computed as follows:

\[
\text{complex} - \text{exponent} = (\text{EXP}(r) \cdot \text{COS}(i), \text{EXP}(r) \cdot \text{SIN}(i))
\]

The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$CxEXP.

**Condition Values Signaled**

- **SS$_$ROPRAND**: Reserved operand. The MTH$CEXP routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$_$FLOOVEMAT**: Floating-point overflow in Math Library: the absolute value of \( r \) is greater than about 88.029 for F-floating values.
Example

This FORTRAN example forms the complex exponential of an F-floating complex number using MTH$CEXP and the FORTRAN random number generator RAN.

Declare Z and MTH$CEXP as complex values. MTH$CEXP will return the exponential value of Z: Z_NEW = MTH$CEXP(Z)

```
COMPLEX Z,Z_NEW,MTH$CEXP
COMPLEX CMPLX
INTEGER M
M = 1234567

Z = CMPLX(RAN(M),RAN(M))

Z is a complex number (r,i) with real part "r" and imaginary part "i".

Z_NEW = MTH$CEXP(Z)
```

This FORTRAN program demonstrates the use of MTH$CEXP as a function call. The output generated by this example is as follows:

The complex number z is (0.8535407,0.2043402)
It has real part 0.8535407 and imaginary part 0.2043402
The complex exponential value of (0.8535407,0.2043402) is (2.299097,0.4764476)
MTH$CxEXP—Complex Exponential

The Complex Exponential routine returns the complex exponential of a complex number.

Format

MTH$CDEXP  complex-exponent,complex-number
MTH$CGEXP  complex-exponent,complex-number

Each of the above formats accepts one of the floating-point complex types as input.

Returns

None.

Arguments

complex-exponent
OpenVMS usage  complex_number
type  D_floating complex, G_floating complex
access  write only
mechanism  by reference

Complex exponential of complex-number. The complex exponential routines that have D-floating complex and G-floating complex input values write the complex-exponent into this argument. For MTH$CDEXP, complex-exponent argument specifies a D-floating complex number. For MTH$CGEXP, complex-exponent specifies a G-floating complex number.

complex-number
OpenVMS usage  complex_number
type  D_floating complex, G_floating complex
access  read only
mechanism  by reference

Complex number whose complex exponential is to be returned. This complex number has the form (r,i), where r is the real part and i is the imaginary part. The complex-number argument is the address of this complex number. For MTH$CDEXP, complex-number specifies a D-floating number. For MTH$CGEXP, complex-number specifies a G-floating number.

Description

The complex exponential is computed as follows:

\[ \text{complex - exponent} = (\exp(r) \cdot \cos(i), \exp(r) \cdot \sin(i)) \]
Condition Values Signaled

SS$_$ROPRAND

Reserved operand. The MTH$_$CxEXP routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_$FLOOHEMA

Floating-point overflow in Math Library: the absolute value of $r$ is greater than about 88.029 for D-floating values, or greater than about 709.089 for G-floating values.

Example

C+ This FORTRAN example forms the complex exponential
C of a G-floating complex number using MTH$_$CGEXP
C and the FORTRAN random number generator RAN.
C
C Declare Z and MTH$_$CGEXP as complex values.
C MTH$_$CGEXP will return the exponential value
C of Z: CALL MTH$_$CGEXP(Z_NEW,Z)
C-
C0MPLEX*16 Z,Z_NEW
COMPLEX*16 MTH$_$GCMPLX
REAL*8 R,I
INTEGER M
M = 1234567
C+
C Generate a random complex number with the FORTRAN
C- generic CMPLX.
C-
R = RAN(M)
I = RAN(M)
Z = MTH$_$GCMPLX(R,I)
TYPE *, ' The complex number $z$ is', Z
C+
C Compute the complex exponential value of Z.
C-
CALL MTH$_$CGEXP(Z_NEW,Z)
TYPE *, ' The complex exponential value of', Z, ' is', Z_NEW
END

This FORTRAN example demonstrates how to access MTH$_$CGEXP as a procedure call. Because G-floating numbers are used, this program must be compiled using the command "FORTRAN/G filename".

Notice the high precision of the output generated:

The complex number $z$ is (0.853540718555450,0.204340159893036)
The complex exponential value of (0.853540718555450,0.204340159893036) is (2.2990677719458,0.476447678044977)
MTH$CLOG—Complex Natural Logarithm (F-Floating Value)

The Complex Natural Logarithm (F-Floating Value) routine returns the complex natural logarithm of a complex number as an F-floating value.

Format

MTH$CLOG  complex-number

Returns

OpenVMS usage complex_number

type F_floating complex

access write only

mechanism by value

The complex natural logarithm of a complex number. MTH$CLOG returns an F-floating complex number.

Arguments

complex-number

OpenVMS usage complex_number

type F_floating complex

access read only

mechanism by reference

Complex number whose complex natural logarithm is to be returned. This complex number has the form (r,i), where r is the real part and i is the imaginary part. The complex-number argument is the address of this complex number. For MTH$CLOG, complex-number specifies an F-floating number.

Description

The complex natural logarithm is computed as follows:

\[ CLOG(z) = (LOG(CABS(z)), ATAN2(i, r)) \]

The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$CxLOG.

Condition Value Signaled

SS$_ROPRAND

Reserved operand. The MTH$CLOG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Examples of using MTH$CLOG from VAX MACRO (using both the CALLS and the CALLG instructions) appear in the introductory section of this manual.
The Complex Natural Logarithm routine returns the complex natural logarithm of a complex number.

**Format**

MTH$CDLOG complex-natural-log, complex-number

MTH$CGLOG complex-natural-log, complex-number

Each of the above formats accepts one of the floating-point complex types as input.

**Returns**

None.

**Arguments**

- **complex-natural-log**
  - OpenVMS usage: complex number
  - type: D_floating complex, G_floating complex
  - access: write only
  - mechanism: by reference

  Natural logarithm of the complex number specified by `complex-number`. The complex natural logarithm routines that have D-floating complex and G-floating complex input values write the address of the complex natural logarithm into `complex-natural-log`. For MTH$CDLOG, the `complex-natural-log` argument specifies a D-floating complex number. For MTH$CGLOG, the `complex-natural-log` argument specifies a G-floating complex number.

- **complex-number**
  - OpenVMS usage: complex number
  - type: D_floating complex, G_floating complex
  - access: read only
  - mechanism: by reference

  Complex number whose complex natural logarithm is to be returned. This complex number has the form \((r, i)\), where \(r\) is the real part and \(i\) is the imaginary part. The `complex-number` argument is the address of this complex number. For MTH$CDLOG, `complex-number` specifies a D-floating number. For MTH$CGLOG, `complex-number` specifies a G-floating number.

**Description**

The complex natural logarithm is computed as follows:

\[
CLOG(z) = (\text{LOG}(\text{CABS}(z)), \text{ATAN2}(i, r))
\]
Condition Value Signaled

MTH$_INVARGMAT

Invalid argument: \( r = i = 0 \). LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.

SS$_FLTOVF_F

Floating point overflow can occur. This condition value is signaled from MTH$CxABS when MTH$CxABS overflows.

SS$_ROPRAND

Reserved operand. The MTH$CxLOG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

Example

```fortran
C+ This FORTRAN example forms the complex logarithm of a D-floating complex number by using MTH$CDLOG and the FORTRAN random number generator RAN.
C C Declare Z and MTH$CDLOG as complex values. Then MTH$CDLOG will return the logarithm of Z: CALL MTH$CDLOG(Z_NEW,Z).
C C Declare Z,Z_LOG, and MTH$DCMPLX as complex values, and R and I as real values. MTH$DCMPLX takes two real arguments and returns one complex number.
C C Given a complex number Z, MTH$CDLOG(Z) returns the complex natural logarithm of Z.
C-

COMPLEX*16 Z,Z_NEW,MTH$DCMPLX
REAL*8 R,I
R = 3.1425637846746565
I = 7.43678469887
Z = MTH$DCMPLX(R,I)

C+ Z is a complex number (r,i) with real part "r" and imaginary part "i".
C C TYPE *, ' The complex number z is','z
C TYPE *, ',
C CALL MTH$CDLOG(Z_NEW,Z)
C TYPE *,', The complex logarithm of','z,' is','Z_NEW
END
```

MTH-37
This FORTRAN example program uses MTH$CDLOG by calling it as a procedure. The output generated by this program is as follows:

The complex number \( z \) is \((3.142563784674657, 7.436784698870000)\)

The complex logarithm of \((3.142563784674657, 7.436784698870000)\) is
\((2.088587642177504, 1.170985519274141)\)
MTH$CMPLX—Complex Number Made from F-Floating-Point

The Complex Number Made from F-Floating-Point routine returns a complex number from two floating-point input values.

Format

```
MTH$CMPLX real-part , imaginary-part
```

Returns

<table>
<thead>
<tr>
<th>OpenVMS usage</th>
<th>complex_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>F_floating complex</td>
</tr>
<tr>
<td>access</td>
<td>write only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by value</td>
</tr>
</tbody>
</table>

A complex number. MTH$CMPLX returns an F-floating complex number.

Arguments

**real-part**

- OpenVMS usage: floating_point
- type: F_floating
- access: read only
- mechanism: by reference

Real part of a complex number. The **real-part** argument is the address of a floating-point number that contains this real part, r, of (r,i). For MTH$CMPLX, **real-part** specifies an F-floating number.

**imaginary-part**

- OpenVMS usage: floating_point
- type: F_floating
- access: read only
- mechanism: by reference

Imaginary part of a complex number. The **imaginary-part** argument is the address of a floating-point number that contains this imaginary part, i, of (r,i). For MTH$CMPLX, **imaginary-part** specifies an F-floating number.

Description

The MTH$CMPLX routines return a complex number from two F-floating input values. The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$xCMPLX.
Reserved operand. The MTH$CMPLX routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

Example

C+ This FORTRAN example forms two F-floating point complex numbers using MTH$CMPLX and the FORTRAN random number generator RAN.
C
C Declare Z and MTH$CMPLX as complex values, and R and I as real values. MTH$CMPLX takes two real F-floating point values and returns one COMPLEX*8 number.
C
Note, since CMPLX is a generic name in FORTRAN, it would be sufficient to use CMPLX.
CMPLX must be declare to be of type COMPLEX*8.
C
Z = CMPLX(R,I)
C
C+ This FORTRAN example forms two F-floating point complex numbers using MTH$CMPLX and the FORTRAN random number generator RAN.
C
C Declare Z and MTH$CMPLX as complex values, and R and I as real values. MTH$CMPLX takes two real F-floating point values and returns one COMPLEX*8 number.
C
Note, since CMPLX is a generic name in FORTRAN, it would be sufficient to use CMPLX.
CMPLX must be declare to be of type COMPLEX*8.
C
Z = CMPLX(R,I)
C
This FORTRAN example program demonstrates the use of MTH$CMPLX. The output generated by this program is as follows:

The two input values are: 0.8535407 0.2043402
The complex number z is (0.8535407,0.2043402)
Using the FORTRAN generic CMPLX with random R and I:
The complex number z is (0.5722565,0.1857677)
The Complex Number Made from D- or G-Floating-Point routine returns a complex number from two D- or G-floating input values.

**Format**

- MTH$DCMPLX complx ,real-part ,imaginary-part
- MTH$GCMPLX complx ,real-part ,imaginary-part

Each of the above formats accepts one of floating-point complex types as input.

**Returns**

None.

**Arguments**

- **complx**
  - OpenVMS usage: complex_number
  - type: D_floating complex, G_floating complex
  - access: write only
  - mechanism: by reference

  The floating-point complex value of a complex number. The complex exponential functions that have D-floating complex and G-floating complex input values write the address of this floating-point complex value into **complx**. For MTH$DCMPLX, **complx** specifies a D-floating complex number. For MTH$GCMPLX, **complx** specifies a G-floating complex number. For MTH$CMPLX, **complx** is not used.

- **real-part**
  - OpenVMS usage: floating_point
  - type: D_floating, G_floating
  - access: read only
  - mechanism: by reference

  Real part of a complex number. The **real-part** argument is the address of a floating-point number that contains this real part, r, of (r,i). For MTH$DCMPLX, **real-part** specifies a D-floating number. For MTH$GCMPLX, **real-part** specifies a G-floating number.

- **imag-part**
  - OpenVMS usage: floating_point
  - type: D_floating, G_floating
  - access: read only
  - mechanism: by reference

  Imaginary part of a complex number. The **imag-part** argument is the address of a floating-point number that contains this imaginary part, i, of (r,i). For MTH$DCMPLX, **imag-part** specifies a D-floating number. For MTH$GCMPLX, **imag-part** specifies a G-floating number.
Condition Value Signaled

**SS$_ROPRAND**

Reserved operand. The MTH$xCmplx routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

Example

```fortran
C
This FORTRAN example forms two D-floating point complex numbers using MTH$DCMPLX and the FORTRAN random number generator RAN.

Declare Z and MTH$DCMPLX as complex values, and R and I as real values. MTH$DCMPLX takes two real D-floating point values and returns one COMPLEX*16 number.

COMPLEX*16 Z
REAL*8 R,I
INTEGER M
M = 1234567
R = RAN(M)
I = RAN(M)
CALL MTH$DCMPLX(Z,R,I)

Z is a complex number (r,i) with real part "r" and imaginary part "i".

TYPE *, ' The two input values are:',R
TYPE *, ' The complex number z is:',Z
END
```

This FORTRAN example demonstrates how to make a procedure call to MTH$DCMPLX. Notice the difference in the precision of the output generated.

The two input values are: 0.8535407185554504 0.2043401598930359
The complex number z is (0.8535407185554504,0.2043401598930359)
MTH$CONJG—Conjugate of a Complex Number (F-Floating Value)

The Conjugate of a Complex Number (F-Floating Value) routine returns the complex conjugate \((r,-i)\) of a complex number \((r,i)\) as an F-floating value.

Format

\[
\text{MTH$CONJG \ complex-number}
\]

Returns

OpenVMS usage \(\text{complex_number}\)
type \(\text{F-floating complex}\)
access \(\text{read only}\)
mechanism \(\text{by reference}\)

Complex conjugate of a complex number. MTH$CONJG returns an F-floating complex number.

Arguments

\[
\text{complex-number}
\]

OpenVMS usage \(\text{complex_number}\)
type \(\text{F-floating complex}\)
access \(\text{read only}\)
mechanism \(\text{by reference}\)

A complex number \((r,i)\), where \(r\) and \(i\) are floating-point numbers. The \text{complex-number} argument is the address of this floating-point complex number. For MTH$CONJG, \text{complex-number} specifies an F-floating number.

Description

The MTH$CONJG routine returns the complex conjugate \((r,-i)\) of a complex number \((r,i)\) as an F-floating value. The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$xCONJG.

Condition Value Signaled

\[
\text{SS$_{\text{ROPRAND}}$}
\]

Reserved operand. The MTH$CONJG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$CONJG—Conjugate of a Complex Number

The Conjugate of a Complex Number routine returns the complex conjugate \((r, -i)\) of a complex number \((r, i)\).

**Format**

\[
\text{MTH$DCONJG} \quad \text{complex-conjugate} , \text{complex-number} \\
\text{MTH$GCONJG} \quad \text{complex-conjugate} , \text{complex-number}
\]

Each of the above formats accepts one of the floating-point complex types as input.

**Returns**

None.

**Arguments**

**complex-conjugate**

OpenVMS usage complex_number

- type: D_floating complex, G_floating complex
- access: write only
- mechanism: by reference

The complex conjugate \((r, -i)\) of the complex number specified by `complex-number`. MTH$DCONJG and MTH$GCONJG write the address of this complex conjugate into `complex-conjugate`. For MTH$DCONJG, the `complex-conjugate` argument specifies the address of a D-floating complex number. For MTH$GCONJG, the `complex-conjugate` argument specifies the address of a G-floating complex number.

**complex-number**

OpenVMS usage complex_number

- type: D_floating complex, G_floating complex
- access: read only
- mechanism: by reference

A complex number \((r, i)\), where \(r\) and \(i\) are floating-point numbers. The `complex-number` argument is the address of this floating-point complex number.

For MTH$DCONJG, `complex-number` specifies a D-floating number. For MTH$GCONJG, `complex-number` specifies a G-floating number.

**Condition Value Signaled**

**SS$ _ROPRAND**

Reserved operand. The MTH$CONJG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Example

This FORTRAN example forms the complex conjugate of a G-floating complex number using MTH$GCONJG and the FORTRAN random number generator RAN.

Declare Z, Z_NEW, and MTH$GCONJG as a complex values. MTH$GCONJG will return the complex conjugate value of Z: Z_NEW = MTH$GCONJG(Z).

C
COMPLEX*16 Z,Z_NEW,MTH$GCONJG
COMPLEX*16 MTH$GCMPLX
REAL*8 R,I,MTH$GREAL,MTH$GIMAG
INTEGER M
M = 1234567
C
Generate a random complex number with the FORTRAN generic CMPLX.

R = RAN(M)
I = RAN(M)
Z = MTH$GCMPLX(R, I)

C
Compute the complex absolute value of Z.

Z_NEW = MTH$GCONJG(Z)

This FORTRAN example demonstrates how to make a function call to MTH$GCONJG. Because G-floating numbers are used, the examples must be compiled with the statement "FORTRAN/G filename".

The output generated by this program is as follows:

The complex number z is (0.853540718555450,0.204340159893036)
with real part 0.8535407185554504
and imaginary part 0.2043401598930359

The complex conjugate value of (0.853540718555450,0.204340159893036) is
(0.853540718555450,-0.20434015989303596)
with real part 0.8535407185554504
and imaginary part -0.20434015989303596
MTH$xCOS

MTH$xCOS—Cosine of Angle Expressed in Radians

The Cosine of Angle Expressed in Radians routine returns the cosine of a given angle (in radians).

Format

MTH$COS  angle-in-radians
MTH$DCOS  angle-in-radians
MTH$GCOS  angle-in-radians

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$COS_R4
MTH$DCOS_R7
MTH$GCOS_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage  floating_point
type            F_floating, D_floating, G_floating
access          write only
mechanism       by value

Cosine of the angle. MTH$COS returns an F-floating number. MTH$DCOS returns a D-floating number. MTH$GCOS returns a G-floating number.

Arguments

angle-in-radians
OpenVMS usage  floating_point
type            F_floating, D_floating, G_floating
access          read only
mechanism       by reference

The angle in radians. The angle-in-radians argument is the address of a floating-point number. For MTH$COS, angle-in-radians is an F-floating number. For MTH$DCOS, angle-in-radians specifies a D-floating number. For MTH$GCOS, angle-in-radians specifies a G-floating number.

Description

See the MTH$xSINCOS routine for the algorithm used to compute the cosine.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HCOS.
Reserved operand. The MTH$xCOS procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$xCOSD

MTH$xCOSD—Cosine of Angle Expressed in Degrees

The Cosine of Angle Expressed in Degrees routine returns the cosine of a given angle (in degrees).

Format

MTH$COSD angle-in-degrees
MTH$DCOSD angle-in-degrees
MTH$GCOSD angle-in-degrees

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$COSD_R4
MTH$DCOSD_R7
MTH$GCOSD_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Cosine of the angle. MTH$COSD returns an F-floating number. MTH$DCOSD returns a D-floating number. MTH$GCOSD returns a G-floating number.

Arguments

angle-in-degrees
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Angle (in degrees). The angle-in-degrees argument is the address of a floating-point number. For MTH$COSD, angle-in-degrees specifies an F-floating number. For MTH$DCOSD, angle-in-degrees specifies a D-floating number. For MTH$GCOSD, angle-in-degrees specifies a G-floating number.

Description

See the MTH$SIN COSD routine for the algorithm used to compute the cosine.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$H COSD.
Condition Value Signaled

Reserved operand. The MTH$x$COSD procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$xCOSH—Hyperbolic Cosine

The Hyperbolic Cosine routine returns the hyperbolic cosine of the input value.

Format

MTH$COSH  floating-point-input-value  
MTH$DCOSH  floating-point-input-value  
MTH$GCOSH  floating-point-input-value  

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage  floating_point  
type  F_floating, D_floating, G_floating  
access  write only  
mechanism  by value  

The hyperbolic cosine of the input value  floating-point-input-value.  
MTH$COSH returns an F-floating number. MTH$DCOSH returns a D-floating number. MTH$GCOSH returns a G-floating number.

Arguments

floating-point-input-value  
OpenVMS usage  floating_point  
type  F_floating, D_floating, G_floating  
access  read only  
mechanism  by reference  

The input value. The  floating-point-input-value argument is the address of this input value. For MTH$COSH,  floating-point-input-value specifies an F-floating number. For MTH$DCOSH,  floating-point-input-value specifies a D-floating number. For MTH$GCOSH,  floating-point-input-value specifies a G-floating number.

Description

Computation of the hyperbolic cosine depends on the magnitude of the input argument. The range of the function is partitioned using four data-type-dependent constants: a(z), b(z), and c(z). The subscript z indicates the data type. The constants depend on the number of exponent bits (e) and the number of fraction bits (f) associated with the data type (z).

The values of e and f are:

<table>
<thead>
<tr>
<th>z</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>G</td>
<td>11</td>
<td>53</td>
</tr>
</tbody>
</table>
The values of the constants in terms of $e$ and $f$ are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a(z)$</td>
<td>$2^{-f/2}$</td>
</tr>
<tr>
<td>$b(z)$</td>
<td>CEILING$[(f + 1)/2 \cdot \ln(2)]$</td>
</tr>
<tr>
<td>$c(z)$</td>
<td>$(2^{e-1}) \cdot \ln(2)$</td>
</tr>
</tbody>
</table>

Based on the above definitions, $z\cosh(X)$ is computed as follows:

<table>
<thead>
<tr>
<th>Value of $X$</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>$a(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$.25 \leq</td>
<td>X</td>
</tr>
<tr>
<td>$b(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$c(z) \leq</td>
<td>X</td>
</tr>
</tbody>
</table>

This routine description for the H-floating point value is listed alphabetically under MTH$\times$HCOSH.

**Condition Values Signaled**

- **SS$\_RORAND**
  
  Reserved operand. The MTH$\times$COSH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\_FLOOVEMAT**
  
  Floating-point overflow in Math Library: the absolute value of floating-point-input-value is greater than about $yyy$; LIB$\_SIGNAL$ copies the reserved operand to the signal mechanism vector. The result is the reserved operand -0.0 unless a condition handler changes the signal mechanism vector.

  The values of $yyy$ are:
  
  MTH$\_COSH$—88.722  
  MTH$\_DCOSH$—88.722  
  MTH$\_GCOSH$—709.782
MTH$CSIN

MTH$CSIN—Sine of a Complex Number (F-Floating Value)

The Sine of a Complex Number (F-Floating Value) routine returns the sine of a complex number \((r,i)\) as an F-floating value.

Format

\[
\text{MTH$CSIN} \text{ complex-number}
\]

Returns

OpenVMS usage complex_number

\[\text{type \hspace{1em} F\text{-floating complex}}\]

\[\text{access \hspace{1em} write only}\]

\[\text{mechanism \hspace{1em} by value}\]

Complex sine of the complex number. MTH$CSIN returns an F-floating complex number.

Arguments

complex-number

OpenVMS usage complex_number

\[\text{type \hspace{1em} F\text{-floating complex}}\]

\[\text{access \hspace{1em} read only}\]

\[\text{mechanism \hspace{1em} by reference}\]

A complex number \((r,i)\), where \(r\) and \(i\) are floating-point numbers. The \text{complex-number} argument is the address of this complex number. For MTH$CSIN, \text{complex-number} specifies an F-floating complex number.

Description

The complex sine is computed as follows:

\[
\text{complex-sine} = (\sin(r) \times \cosh(i), \cos(r) \times \sinh(i))
\]

The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$CxSIN.

Condition Values Signaled

SS$_{\text{ROPRAND}}$ Reserved operand. The MTH$CSIN$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_{\text{FLOOVEMAT}}$ Floating-point overflow in Math Library: the absolute value of \(i\) is greater than about 88.029 for F-floating values.
MTH$CxSIN—Sine of a Complex Number

The Sine of a Complex Number routine returns the sine of a complex number (r,i).

Format

MTH$CDSIN  complex-sine, complex-number
MTH$CGSIN  complex-sine, complex-number

Each of the above formats accepts one of the floating-point complex types as input.

Returns

None.

Arguments

complex-sine
OpenVMS usage  complex_number
  type          D_floating complex, G_floating complex
  access        write only
  mechanism     by reference

Complex sine of the complex number. The complex sine routines with D-floating complex and G-floating complex input values write the complex sine into this complex-sine argument. For MTH$CDSIN, complex-sine specifies a D-floating complex number. For MTH$CGSIN, complex-sine specifies a G-floating complex number.

complex-number
OpenVMS usage  complex_number
  type          D_floating complex, G_floating complex
  access        read only
  mechanism     by reference

A complex number (r,i), where r and i are floating-point numbers. The complex-number argument is the address of this complex number. For MTH$CDSIN, complex-number specifies a D-floating complex number. For MTH$CGSIN, complex-number specifies a G-floating complex number.

Description

The complex sine is computed as follows:

\[
\text{complex sine} = (\sin(r) \cdot \cosh(i), \cos(r) \cdot \sinh(i))
\]
Reserved operand. The MTH$CxSIN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

Floating-point overflow in Math Library: the absolute value of $i$ is greater than about 88.029 for D-floating values, or greater than about 709.089 for G-floating values.

This FORTRAN example demonstrates a procedure call to MTH$CGSIN. Because this program uses G-floating numbers, it must be compiled with the statement "FORTRAN/G filename".
The output generated by this program is as follows:

The complex number $z$ is $(0.853540718555450, 0.204340159893036)$
The complex sine value of $(0.853540718555450, 0.204340159893036)$ is
$(0.769400835484975, 0.135253340912255)$
MTH$CSQRT

MTH$CSQRT—Complex Square Root (F-Floating Value)

The Complex Square Root (F-Floating Value) routine returns the complex square root of a complex number (r,i).

Format

MTH$CSQRT complex-number

Returns

OpenVMS usage complex_number
type F_floating complex
access write only
mechanism by value

The complex square root of complex-number. MTH$CSQRT returns an F-floating number.

Arguments

complex-number
OpenVMS usage complex_number
type F_floating complex
access read only
mechanism by reference

Complex number (r,i). The complex-number argument contains the address of this complex number. For MTH$CSQRT, complex-number specifies an F-floating number.

Description

The complex square root is computed as follows.

First, calculate \( \text{ROOT} \) and \( \text{Q} \) using the following equations:

\[
\text{ROOT} = \text{SQRT}\left(\frac{\text{ABS}(r) + \text{CABS}(r,i)}{2}\right)
\]

\[
\text{Q} = \frac{i}{2 \cdot \text{ROOT}}
\]

Then, the complex result is given as follows:

<table>
<thead>
<tr>
<th>r</th>
<th>i</th>
<th>CSQRT((r,i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>Any</td>
<td>(ROOT,Q)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>≥0</td>
<td>(Q,ROOT)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&lt;0</td>
<td>(-Q,-ROOT)</td>
</tr>
</tbody>
</table>

The routine descriptions for the D- and G-floating point versions of this routine are listed alphabetically under MTH$CxSQRT.
Floating point overflow can occur. Reserved operand. The MTH$CSQRT procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$CxSQRT

MTH$CxSQRT—Complex Square Root

The Complex Square Root routine returns the complex square root of a complex number \((r,i)\).

Format

\[
\begin{align*}
\text{MTH$CDSQRT} & \quad \text{complex-square-root ,complex-number} \\
\text{MTH$CGSQRT} & \quad \text{complex-square-root ,complex-number}
\end{align*}
\]

Each of the above formats accepts one of the floating-point complex types as input.

Returns

None.

Arguments

\text{complex-square-root} \hspace{1cm} \text{OpenVMS usage complex_number} \\
\text{type} \quad \text{D_floating complex, G_floating complex} \\
\text{access} \quad \text{write only} \\
\text{mechanism} \quad \text{by reference}

Complex square root of the complex number specified by \text{complex-number}. The complex square root routines that have D-floating complex and G-floating complex input values write the complex square root into \text{complex-square-root}. For MTH$CDSQRT, \text{complex-square-root} specifies a D-floating complex number. For MTH$CGSQRT, \text{complex-square-root} specifies a G-floating complex number.

\text{complex-number} \hspace{1cm} \text{OpenVMS usage complex_number} \\
\text{type} \quad \text{D_floating complex, G_floating complex} \\
\text{access} \quad \text{read only} \\
\text{mechanism} \quad \text{by reference}

Complex number \((r,i)\). The \text{complex-number} argument contains the address of this complex number. For MTH$CDSQRT, \text{complex-number} specifies a D-floating number. For MTH$CGSQRT, \text{complex-number} specifies a G-floating number.

Description

The complex square root is computed as follows.

First, calculate \(ROOT\) and \(Q\) using the following equations:

\[
\begin{align*}
ROOT & = \sqrt{(ABS(r) + CABS(r,i))/2} \\
Q & = i/(2 \times ROOT)
\end{align*}
\]
Then, the complex result is given as follows:

<table>
<thead>
<tr>
<th>r</th>
<th>l</th>
<th>CSQRT((r,l))</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>any</td>
<td>(ROOT,Q)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>≥0</td>
<td>(Q,ROOT)</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&lt;0</td>
<td>(-Q,-ROOT)</td>
</tr>
</tbody>
</table>

**Condition Value Signaled**

- **SS$\_FLTOVF\_F**
  - Floating point overflow can occur.
- **SS$\_ROPRAND**
  - Reserved operand. The MTH$CxSQRT procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**Example**

```
C* This FORTRAN example forms the complex square root of a D-floating complex number using MTH$CDSQRT and the FORTRAN random number generator RAN.
C*
C Declare Z and Z_NEW as complex values. MTH$CDSQRT will return the complex square root of Z: CALL MTH$CDSQRT(Z_NEW,Z).
C-
COMPLEX*16 Z,Z_NEW
COMPLEX*16 DCMPLX
INTEGER M
M = 1234567
C*
C Generate a random complex number with the FORTRAN generic CMPLX.
C-
Z = DCMPLX(RAN(M),RAN(M))
C*
Z is a complex number (r,i) with real part "r" and imaginary part "i".
C-
TYPE *, ' The complex number z is',z
TYPE *, ''
C*
Compute the complex complex square root of Z.
C-
CALL MTH$CDSQRT(Z_NEW,Z)
TYPE *, ' The complex square root of z, is',Z_NEW
END
```
This FORTRAN example program demonstrates a procedure call to MTH$CDSQRT. The output generated by this program is as follows:

The complex number $z$ is $(0.8535407185554504, 0.2043401598930359)$
The complex square root of $(0.8535407185554504, 0.2043401598930359)$ is $(0.9303763973040062, 0.1098158554350485)$
MTH$CVT_x_x—Convert One Double-Precision Value

The Convert One Double-Precision Value routines convert one double-precision value to the destination data type and return the result as a function value. MTH$CVT_D_G converts a D-floating value to G-floating and MTH$CVT_G_D converts a G-floating value to a D-floating value.

Format

MTH$CVT_D_G  floating-point-input-val
MTH$CVT_G_D  floating-point-input-val

Returns

OpenVMS usage  floating_point
  type  G_floating, D_floating
  access  write only
  mechanism  by value

The converted value. MTH$CVT_D_G returns a G-floating value. MTH$CVT_G_D returns a D-floating value.

ARGUMENT

floating-point-input-val
OpenVMS usage  floating_point
  type  D_floating, G_floating
  access  read only
  mechanism  by reference

The input value to be converted. The floating-point-input-val argument is the address of this input value. For MTH$CVT_D_G, the floating-point-input-val argument specifies a D-floating number. For MTH$CVT_G_D, the floating-point-input-val argument specifies a G-floating number.

Description

These procedures are designed to function as hardware conversion instructions. They fault on reserved operands. If floating-point overflow is detected, an error is signaled. If floating-point underflow is detected and floating-point underflow is enabled, an error is signaled.
Condition Values Signaled

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$ _ROPRAND</td>
<td>Reserved operand. The MTH$CVT_x_x procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
<tr>
<td>MTH$_FLOOVEMAT</td>
<td>Floating-point overflow in Math Library.</td>
</tr>
<tr>
<td>MTH$_FLOUNDMAT</td>
<td>Floating-point underflow in Math Library.</td>
</tr>
</tbody>
</table>
MTH$CVT_xA_xA—Convert an Array of Double-Precision Values

The Convert an Array of Double-Precision Values routines convert a contiguous array of double-precision values to the destination data type and return the results as an array. MTH$CVT_DA_GA converts D-floating values to G-floating and MTH$CVT_GA_DA converts G-floating values to D-floating.

Format

MTH$CVT_DA_GA floating-point-input-array ,floating-point-dest-array [,array-size]
MTH$CVT_GA_DA floating-point-input-array ,floating-point-dest-array [,array-size]

Returns

OpenVMS usage HEADONLY

MTH$CVT_DA_GA and MTH$CVT_GA_DA return the address of the output array to the floating-point-dest-array argument.

Arguments

floating-point-input-array
OpenVMS usage floating_point
type D_floating, G_floating
access read only
mechanism by reference, array reference

Input array of values to be converted. The floating-point-input-array argument is the address of an array of floating-point numbers. For MTH$CVT_DA_GA, floating-point-input-array specifies an array of D-floating numbers. For MTH$CVT_GA_DA, floating-point-input-array specifies an array of G-floating numbers.

floating-point-dest-array
OpenVMS usage floating_point
type G_floating, D_floating
access write only
mechanism by reference, array reference

Output array of converted values. The floating-point-dest-array argument is the address of an array of floating-point numbers. For MTH$CVT_DA_GA, floating-point-dest-array specifies an array of G-floating numbers. For MTH$CVT_GA_DA, floating-point-dest-array specifies an array of D-floating numbers.

array-size
OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Number of array elements to be converted. The default value is 1. The array-size argument is the address of a longword containing this number of elements.
These procedures are designed to function as hardware conversion instructions. They fault on reserved operands. If floating-point overflow is detected, an error is signaled. If floating-point underflow is detected and floating-point underflow is enabled, an error is signaled.

**Condition Values Signaled**

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$_ROPRAND</td>
<td>Reserved operand. The MTH$CVT_xA_xA procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
<tr>
<td>MTH$_FLOOVEMAT</td>
<td>Floating-point overflow in Math Library.</td>
</tr>
<tr>
<td>MTH$_FLOUNDMAT</td>
<td>Floating-point underflow in Math Library.</td>
</tr>
</tbody>
</table>
**MTH$\times$EXP—Exponential**

The Exponential routine returns the exponential of the input value.

**Format**

- `MTH$\times$EXP floating-point-input-value`
- `MTH$\times$DEXP floating-point-input-value`
- `MTH$\times$GEXP floating-point-input-value`

Each of the above formats accepts one of the floating-point types as input.

**JSB Entries**

- `MTH$\times$EXP_R4`
- `MTH$\times$DEXP_R6`
- `MTH$\times$GEXP_R6`

Each of the above JSB entries accepts one of the floating-point types as input.

**Returns**

- OpenVMS usage: `floating_point`
- type: `F_floating, D_floating, G_floating`
- access: `write only`
- mechanism: `by value`

The exponential of `floating-point-input-value`. `MTH$\times$EXP` returns an F-floating number. `MTH$\times$DEXP` returns a D-floating number. `MTH$\times$GEXP` returns a G-floating number.

**Arguments**

- `floating-point-input-value`

  - OpenVMS usage: `floating_point`
  - type: `F_floating, D_floating, G_floating`
  - access: `read only`
  - mechanism: `by reference`

The input value. The `floating-point-input-value` argument is the address of a floating-point number. For `MTH$\times$EXP`, `floating-point-input-value` specifies an F-floating number. For `MTH$\times$DEXP`, `floating-point-input-value` specifies a D-floating number. For `MTH$\times$GEXP`, `floating-point-input-value` specifies a G-floating number.

**Description**

The exponential of $x$ is computed as:

<table>
<thead>
<tr>
<th>Value of $x$</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &gt; c(z)$</td>
<td>Overflow occurs</td>
</tr>
<tr>
<td>$X \leq -c(z)$</td>
<td>0</td>
</tr>
</tbody>
</table>
### MTH$xEXP

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>Otherwise</td>
<td>$2^Y \cdot 2^U \cdot 2^W$</td>
</tr>
</tbody>
</table>

where: $Y = INTEGER(z \cdot \ln2(E))$, $V = FRAC(z \cdot \ln2(E)) \cdot 16$
$U = INTEGER(V)/16$, $W = FRAC(V)/16 \cdot 2^W$ = polynomial approximation of degree 4, 8, or 8 for $z = F$, D, or G.

See also the section on the hyperbolic cosine for definitions of $f$ and $c(z)$.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HEXP$.

### Condition Values Signaled

**SS$_$ROPRAND**

Reserved operand. The MTH$xEXP$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$$_$FLOOVEMAT**

Floating-point overflow in Math Library: floating-point-input-value is greater than $yyy$; LIB$SIGNAL$ copies the reserved operand to the signal mechanism vector. The result is the reserved operand -0.0 unless a condition handler changes the signal mechanism vector.

The values of $yyy$ are approximately:

- MTH$EXP$—88.029
- MTH$DEXP$—88.029
- MTH$GEXP$—709.089

**MTH$$_$FLOUNDMAT**

Floating-point underflow in Math Library: floating-point-input-value is less than or equal to $yyy$ and the caller (CALL or JSB) has set hardware floating-point underflow enable. The result is set to 0.0. If the caller has not enabled floating-point underflow (the default), a result of 0.0 is returned but no error is signaled.

The values of $yyy$ are approximately:

- MTH$EXP$—88.722
- MTH$DEXP$—88.722
- MTH$GEXP$—709.774
Example

IDENTIFICATION DIVISION.
PROGRAM-ID. FLOATING_POINT.
* Calls MTH$EXP using a Floating Point data type.
* Calls MTH$DEXP using a Double Floating Point data type.
*
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 FLOAT_PT COMP-1.
  01 ANSWER_F COMP-1.
  01 DOUBLE_PT COMP-2.
  01 ANSWER_D COMP-2.
PROCEDURE DIVISION.
  P0.
    MOVE 12.34 TO FLOAT_PT.
    MOVE 3.456 TO DOUBLE_PT.
    CALL "MTH$EXP" USING BY REFERENCE FLOAT_PT GIVING ANSWER_F.
    DISPLAY " MTH$EXP of ", FLOAT_PT CONVERSION, " is ", ANSWER_F CONVERSION.
    CALL "MTH$DEXP" USING BY REFERENCE DOUBLE_PT GIVING ANSWER_D.
    DISPLAY " MTH$DEXP of ", DOUBLE_PT CONVERSION, " is ", ANSWER_D CONVERSION.
    STOP RUN.

This sample program demonstrates calls to MTH$EXP and MTH$DEXP from COBOL.

The output generated by this program is as follows:

MTH$EXP of 1.234000E+01 is 2.286620E+05
MTH$DEXP of 3.456000000000000E+00 is 3.168996280537917E+01
MTH$HACOS

MTH$HACOS—Arc Cosine of Angle Expressed in Radians
(H-Floating Value)

Given the cosine of an angle, the Arc Cosine of Angle Expressed in Radians
(H-Floating Value) routine returns that angle (in radians) in H-floating-point
precision.

Format

MTH$HACOS  h-radians ,cosine

JSB Entries

MTH$HACOS_R8

Returns

None.

Arguments

h-radians
OpenVMS usage  floating_point
type  H_floating
access  write only
mechanism  by reference

Angle (in radians) whose cosine is specified by cosine. The h-radians argument
is the address of an H-floating number that is this angle. MTH$HACOS writes
the address of the angle into h-radians.

cosine
OpenVMS usage  floating_point
type  H_floating
access  read only
mechanism  by reference

The cosine of the angle whose value (in radians) is to be returned. The cosine
argument is the address of a floating-point number that is this cosine. The
absolute value of cosine must be less than or equal to 1. For MTH$HACOS,
cosine specifies an H-floating number.

Description

The angle in radians whose cosine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Cosine</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>π/2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>π</td>
</tr>
</tbody>
</table>
Value of Cosine | Value Returned
---|---
0 < X < 1 | z\(ATAN(zSQRT(1 - X^2)/X)\), where z\(ATAN\) and z\(SQRT\) are the Math Library arc tangent and square root routines, respectively, of the appropriate data type
-1 < X < 0 | z\(ATAN(zSQRT(1 - X^2)/X) + \pi\)
1 < |X| | The error MTH$INVARGMAT is signaled

### Condition Values Signaled

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$$_{ROPRAND}$$</td>
<td>Reserved operand. The MTH$\times$ACOS routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
<tr>
<td>MTH$$_{INVARGMAT}$$</td>
<td>Invalid argument. The absolute value of \texttt{cosine} is greater than 1. LIB$SIGNAL$ copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.</td>
</tr>
</tbody>
</table>
MTH$HACOSD—Arc Cosine of Angle Expressed in Degrees (H-Floating Value)

Given the cosine of an angle, the Arc Cosine of Angle Expressed in Degrees (H-Floating Value) routine returns that angle (in degrees) as an H-floating value.

Format

MTH$HACOSD  h-degrees, cosine

JSB Entries

MTH$HACOSD_R8

Returns

None.

Arguments

h-degrees
OpenVMS usage floating_point
  type H_floating
  access write only
  mechanism by reference

Angle (in degrees) whose cosine is specified by cosine. The h-degrees argument is the address of an H-floating number that is this angle. MTH$HACOSD writes the address of the angle into h-degrees.

cosine
OpenVMS usage floating_point
  type H_floating
  access read only
  mechanism by reference

Cosine of the angle whose value (in degrees) is to be returned. The cosine argument is the address of a floating-point number that is this cosine. The absolute value of cosine must be less than or equal to 1. For MTH$HACOSD, cosine specifies an H-floating number.

Description

The angle in degrees whose cosine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Cosine</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>180</td>
</tr>
<tr>
<td>0 &lt; X &lt; 1</td>
<td>ATAND((\sqrt{1 - X^2}))/X, where ATAND and SQRT are the Math Library arc tangent and square root routines, respectively, of the appropriate data type</td>
</tr>
<tr>
<td>Value of Cosine</td>
<td>Angle Returned</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>(-1 &lt; X &lt; 0)</td>
<td>$z \text{ATAND}(z \text{SQRT}(1 - X^2)/X) + 180$</td>
</tr>
<tr>
<td>$1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

**Condition Values Signaled**

- **SS$\_\text{ROPRAND}$**: Reserved operand. The MTH$\_\text{xACOSD}$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\_\text{INVARGMAT}$**: Invalid argument. The absolute value of cosine is greater than 1. LIB$\text{SIGNAL}$ copies the floating-point reserved operand to the mechanism argument vector CHF$\_\text{L_MCH_SAVR0/R1}$. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$\_\text{L_MCH_SAVR0/R1}$.
MTH$HASIN—Arc Sine in Radians (H-Floating Value)

Given the sine of an angle, the Arc Sine in Radians (H-Floating Value) routine returns that angle (in radians) as an H-floating value.

Format

MTH$HASIN h-radians ,sine

JSB Entries

MTH$HASIN_R8

Returns

None.

Arguments

h-radians
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Angle (in radians) whose sine is specified by sine. The h-radians argument is the address of an H-floating number that is this angle. MTH$HASIN writes the address of the angle into h-radians.

sine
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The sine of the angle whose value (in radians) is to be returned. The sine argument is the address of a floating-point number that is this sine. The absolute value of sine must be less than or equal to 1. For MTH$HASIN, sine specifies an H-floating number.

Description

The angle in radians whose sine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Sine</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>( \pi/2 )</td>
</tr>
<tr>
<td>-1</td>
<td>(-\pi/2)</td>
</tr>
<tr>
<td>0 &lt; (</td>
<td>X</td>
</tr>
<tr>
<td>1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

MTH-72
Condition Values Signaled

SS$_ROPRAND

Reserved operand. The MTH$_xASIN routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_INVARGMAT

Invalid argument. The absolute value of \texttt{sine} is greater than 1. LIB$_SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L$_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L$_MCH_SAVR0/R1.
MTH$HASIND—Arc Sine in Degrees (H-Floating Value)

Given the sine of an angle, the Arc Sine in Degrees (H-Floating Value) routine returns that angle (in degrees) as an H-floating value.

Format

```
MTH$HASIND h-degrees, sine
```

JSB Entries

```
MTH$HASIND_R8
```

Returns

None.

Arguments

**h-degrees**

OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Angle (in degrees) whose sine is specified by sine. The h-degrees argument is the address of an H-floating number that is this angle. MTH$HASIND writes the address of the angle into h-degrees.

**sine**

OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Sine of the angle whose value (in degrees) is to be returned. The sine argument is the address of a floating-point number that is this sine. The absolute value of sine must be less than or equal to 1. For MTH$HASIND, sine specifies an H-floating number.

Description

The angle in degrees whose sine is X is computed as:

<table>
<thead>
<tr>
<th>Value of Sine</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>-1</td>
<td>-90</td>
</tr>
<tr>
<td>0 &lt;</td>
<td>X</td>
</tr>
<tr>
<td>1 &lt;</td>
<td>X</td>
</tr>
</tbody>
</table>

MTH-74
Condition Values Signaled

**SS$_ROPRAND**

Reserved operand. The MTH$_xASIND routine encountered a floating point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_INVARGMAT**

Invalid argument. The absolute value of sine is greater than 1. LIB$_SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$HATAN

MTH$HATAN—Arc Tangent in Radians (H-Floating Value)

Given the tangent of an angle, the Arc Tangent in Radians (H-Floating Value) routine returns that angle (in radians) as an H-floating value.

Format

MTH$HATAN  h-radians, tangent

JSB Entries

MTH$HATAN_R8

Returns

None.

Arguments

h-radians
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Angle (in radians) whose tangent is specified by tangent. The h-radians argument is the address of an H-floating number that is this angle. MTH$HATAN writes the address of the angle into h-radians.

tangent
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The tangent of the angle whose value (in radians) is to be returned. The tangent argument is the address of a floating-point number that is this tangent. For MTH$HATAN, tangent specifies an H-floating number.

Description

In radians, the computation of the arc tangent function is based on the following identities:

\[
\arctan(X) = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + ... \\
\arctan(X) = X + X \cdot Q(X^2), \text{ where } Q(Y) = -\frac{Y}{3} + \frac{Y^2}{5} - \frac{Y^3}{7} + ... \\
\arctan(X) = X \cdot P(X^2), \text{ where } P(Y) = 1 - \frac{Y}{3} + \frac{Y^2}{5} - \frac{Y^3}{7} + ... \\
\arctan(X) = \frac{\pi}{2} - \arctan(\frac{1}{X}) \\
\arctan(X) = \arctan(A) + \arctan\left(\frac{X - A}{1 + A \cdot X}\right) \text{ for any real } A
\]

MTH-76
The angle in radians whose tangent is $X$ is computed as:

<table>
<thead>
<tr>
<th>Value of $X$</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq X \leq 3/32$</td>
<td>$X + X \cdot Q(X^2)$</td>
</tr>
<tr>
<td>$3/32 &lt; X \leq 11$</td>
<td>$ATAN(A) + V \cdot (P(V^2))$, where $A$ and ATAN(A) are chosen by table lookup and $V = (X - A)/(1 + A \cdot X)$</td>
</tr>
<tr>
<td>$11 &lt; X$</td>
<td>$\pi/2 - W \cdot (P(W^2))$ where $W = 1/X$</td>
</tr>
<tr>
<td>$X &lt; 0$</td>
<td>$-ATAN(</td>
</tr>
</tbody>
</table>

**Condition Value Signaled**

**SS$_R$OPRAND**

Reserved operand. The MTH$\times$ATAN routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$HATAND

MTH$HATAND—Arc Tangent in Degrees (H-Floating Value)

Given the tangent of an angle, the Arc Tangent in Degrees (H-Floating Value) routine returns that angle (in degrees) as an H-floating point value.

Format

MTH$HATAND h-degrees ,tangent

JSB Entries

MTH$HATAND_R8

Returns

None.

Arguments

h-degrees
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Angle (in degrees) whose tangent is specified by tangent. The h-degrees argument is the address of an H-floating number that is this angle. MTH$HATAND writes the address of the angle into h-degrees.

tangent
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The tangent of the angle whose value (in degrees) is to be returned. The tangent argument is the address of a floating-point number that is this tangent. For MTH$HATAND, tangent specifies an H-floating number.

Description

The computation of the arc tangent function is based on the following identities:

\[
\begin{align*}
\arctan(X) &= 180/\pi \cdot (X - X^3/3 + X^5/5 - X^7/7 + \ldots) \\
\arctan(X) &= 64 \cdot X + X \cdot Q(X^2), \\
&\text{where } Q(Y) = 180/\pi \cdot \left[ (1 - 64 \cdot \pi/180) - Y/3 + Y^2/5 - Y^3/7 + \ldots \right] \\
\arctan(X) &= X \cdot P(X^2), \\
&\text{where } P(Y) = 180/\pi \cdot \left[ (1 - Y/3 + Y^2/5) - Y^3/7 + Y^4/9 + \ldots \right] \\
\arctan(X) &= 90 - \arctan(1/X) \\
\arctan(X) &= \arctan(A) + \arctan((X - A)/(1 + A \cdot X))
\end{align*}
\]

MTH-78
The angle in degrees whose tangent is $X$ is computed as:

<table>
<thead>
<tr>
<th>Tangent</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \leq 3/32$</td>
<td>$64 \cdot X + X \cdot Q(X^2)$</td>
</tr>
<tr>
<td>$3/32 &lt; X \leq 11$</td>
<td>$\text{ATAND}(A) + V \cdot P(V^2)$, where $A$ and $\text{ATAND}(A)$ are chosen by table lookup and $V = (X - A)/(1 + A \cdot X)$</td>
</tr>
<tr>
<td>$11 &lt; X$</td>
<td>$90 - W \cdot (P(W^2))$, where $W = 1/X$</td>
</tr>
<tr>
<td>$X &lt; 0$</td>
<td>$-\text{ATAND}(</td>
</tr>
</tbody>
</table>

**Condition Value Signaled**

**SS$_{ROPRAND}$**

Reserved operand. The MTH$	ext{xATAND}$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$HATAN2—Arc Tangent in Radians (H-Floating Value) with Two Arguments

Given sine and cosine, the Arc Tangent in Radians (H-Floating Value) with Two Arguments routine returns the angle (in radians) as an H-floating value whose tangent is given by the quotient of sine and cosine, (sine/cosine).

Format

MTH$HATAN2  h-radians ,sine ,cosine

Returns

None.

Arguments

h-radians
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Angle (in radians) whose tangent is specified by (sine/cosine). The h-radians argument is the address of an H-floating number that is this angle. MTH$HATAN2 writes the address of the angle into h-radians.

sine
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Dividend. The sine argument is the address of a floating-point number that is this dividend. For MTH$HATAN2, sine specifies an H-floating number.

cosine
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Divisor. The cosine argument is the address of a floating-point number that is this divisor. For MTH$HATAN2, cosine specifies an H-floating number.

Description

The angle in radians whose tangent is Y/X is computed as follows, where f is defined in the description of MTH$zCOSH:

<table>
<thead>
<tr>
<th>Value of Input Arguments</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 0 or Y/X &gt; 2^(f+1)</td>
<td>\pi/2 * (signY)</td>
</tr>
<tr>
<td>X &gt; 0 and Y/X ≤ 2^(f+1)</td>
<td>zATAN(Y/X)</td>
</tr>
</tbody>
</table>

MTH–80
### Value of Input Arguments

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &lt; 0$ and $Y/X \leq 2^{(f+1)}$</td>
<td>$\pi \cdot (\text{sign} Y) + \pi \text{ATAN}(Y/X)$</td>
</tr>
</tbody>
</table>

### Condition Values Signaled

- **SS$\_\text{ROPRAND}**: Reserved operand. The MTH$\_\text{HATAN2}$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\_\text{INVARGMAT}**: Invalid argument. Both cosine and sine are zero. LIB$\_\text{SIGNAL}$ copies the floating-point reserved operand to the mechanism argument vector CHF$L\_\text{MCH\_SAVR0/R1}$. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\_\text{MCH\_SAVR0/R1}$.
MTH$HATAND2

MTH$HATAND2—Arc Tangent in Degrees (H-Floating Value) with Two Arguments

Given sine and cosine, the Arc Tangent in Degrees (H-Floating Value) with Two Arguments routine returns the angle (in degrees) whose tangent is given by the quotient of sine and cosine, \( \frac{\text{sine}}{\text{cosine}} \).

Format

\[
\text{MTH$HATAND2 \ h-degrees ,sine ,cosine}
\]

Returns

None.

Arguments

- **h-degrees**
  - OpenVMS usage: floating_point
  - Type: H_floating
  - Access: write only
  - Mechanism: by reference
  - Angle (in degrees) whose tangent is specified by \( \frac{\text{sine}}{\text{cosine}} \). The h-degrees argument is the address of an H-floating number that is this angle. MTH$HATAND2 writes the address of the angle into h-degrees.

- **sine**
  - OpenVMS usage: floating_point
  - Type: H_floating
  - Access: read only
  - Mechanism: by reference
  - Dividend. The sine argument is the address of a floating-point number that is this dividend. For MTH$HATAND2, sine specifies an H-floating number.

- **cosine**
  - OpenVMS usage: floating_point
  - Type: H_floating
  - Access: read only
  - Mechanism: by reference
  - Divisor. The cosine argument is the address of a floating-point number that is this divisor. For MTH$HATAND2, cosine specifies an H-floating number.

Description

The angle in degrees whose tangent is \( \frac{Y}{X} \) is computed below. The value of \( f \) is defined in the description of MTH$zCOSH.

<table>
<thead>
<tr>
<th>Value of Input Arguments</th>
<th>Angle Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X = 0 ) or ( Y/X &gt; 2^{(f+1)} )</td>
<td>( 90 \times (\text{sign} \ Y) )</td>
</tr>
<tr>
<td>( X &gt; 0 ) and ( Y/X \leq 2^{(f+1)} )</td>
<td>( \text{ATAND}(Y/X) )</td>
</tr>
</tbody>
</table>

MTH-82
Value of Input Arguments | Angle Returned
---|---
$X < 0$ and $Y/X \leq 2^{(j+1)}$ | $180 \cdot (\text{sign}Y) + z\text{ATAND}(Y/X)$

**Condition Values Signaled**

- **SS$\_ROPRAND**
  Reserved operand. The MTH$\_HATAND2$ routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\_INVARGMAT**
  Invalid argument. Both cosine and sine are zero. LIB$\_SIGNAL$ copies the floating-point reserved operand to the mechanism argument vector CHF$L\_MCH\_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\_MCH\_SAVR0/R1.
MTH$HATANH—Hyperbolic Arc Tangent (H-Floating Value)

Given the hyperbolic tangent of an angle, the Hyperbolic Arc Tangent (H-Floating Value) routine returns the hyperbolic arc tangent (as an H-floating Value) of that angle.

Format

MTH$HATANH  h-atanh ,hyperbolic-tangent

Returns

None.

Arguments

h-atanh
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Hyperbolic arc tangent of the hyperbolic tangent specified by hyperbolic-tangent. The h-atanh argument is the address of an H-floating number that is this hyperbolic arc tangent. MTH$HATANH writes the address of the hyperbolic arc tangent into h-atanh.

hyperbolic-tangent
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Hyperbolic tangent of an angle. The hyperbolic-tangent argument is the address of a floating-point number that is this hyperbolic tangent. For MTH$HATANH, hyperbolic-tangent specifies an H-floating number.

Description

The hyperbolic arc tangent function is computed as follows:

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>$</td>
<td>X</td>
</tr>
</tbody>
</table>

MTH-84
Reserved operand. The MTH$XATANH routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

Invalid argument: \( |X| \geq 1 \). LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$HCOS—Cosine of Angle Expressed in Radians (H-Floating Value)

The Cosine of Angle Expressed in Radians (H-Floating Value) routine returns the cosine of a given angle (in radians) as an H-floating value.

Format

MTH$HCOS h-cosine ,angle-in-radians

JSB Entries

MTH$HCOS_R5

Returns

None.

Arguments

h-cosine
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Cosine of the angle specified by angle-in-radians. The h-cosine argument is the address of an H-floating number that is this cosine. MTH$HCOS writes the address of the cosine into h-cosine.

angle-in-radians
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The angle in radians. The angle-in-radians argument is the address of a floating-point number. For MTH$HCOS, angle-in-radians specifies an H-floating number.

Description

See the MTH$xSINCOS routine for the algorithm used to compute the cosine.

Condition Value Signaled

SS$_ROPRAND Reserved operand. The MTH$HCOS procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$HCOSD—Cosine of Angle Expressed in Degrees (H-Floating Value)

The Cosine of Angle Expressed in Degrees (H-Floating Value) routine returns the cosine of a given angle (in degrees) as an H-floating value.

Format

MTH$HCOSD h-cosine ,angle-in-degrees

JSB Entries

MTH$HCOSD_R5

Returns

None.

Arguments

h-cosine
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Cosine of the angle specified by angle-in-degrees. The h-cosine argument is the address of an H-floating number that is this cosine. MTH$HCOSD writes this cosine into h-cosine.

angle-in-degrees
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Angle (in degrees). The angle-in-degrees argument is the address of a floating-point number. For MTH$HCOSD, angle-in-degrees specifies an H-floating number.

Description

See the MTH$SINCOSD routine for the algorithm used to compute the cosine.

Condition Value Signaled

SS$_ROPRAND

Reserved operand. The MTH$HCOSD procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$HCOSH—Hyperbolic Cosine (H-Floating Value)

The Hyperbolic Cosine (H-Floating Value) routine returns the hyperbolic cosine of the input value as an H-floating value.

Format

MTH$HCOSH  h-cosh , floating-point-input-value

Returns

None.

Arguments

h-cosh

OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Hyperbolic cosine of the input value specified by floating-point-input-value. The h-cosh argument is the address of an H-floating number that is this hyperbolic cosine. MTH$HCOSH writes the address of the hyperbolic cosine into h-cosh.

floating-point-input-value

OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of this input value. For MTH$HCOSH, floating-point-input-value specifies an H-floating number.

Description

Computation of the hyperbolic cosine depends on the magnitude of the input argument. The range of the function is partitioned using four data-type-dependent constants: a(z), b(z), and c(z). The subscript z indicates the data type. The constants depend on the number of exponent bits (e) and the number of fraction bits (f) associated with the data type (z).

The values of e and f are as follows:

\[ e = 15 \]
\[ f = 113 \]
The values of the constants in terms of $e$ and $f$ are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a(z)$</td>
<td>$2^{-f/2}$</td>
</tr>
<tr>
<td>$b(z)$</td>
<td>$(f + 1)/2 \cdot \ln(2)$</td>
</tr>
<tr>
<td>$c(z)$</td>
<td>$2^{e-1} \cdot \ln(2)$</td>
</tr>
</tbody>
</table>

Based on the above definitions, $z\text{COSH}(X)$ is computed as follows:

<table>
<thead>
<tr>
<th>Value of $X$</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>$a(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$.25 \leq</td>
<td>X</td>
</tr>
<tr>
<td>$b(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$c(z) \leq</td>
<td>X</td>
</tr>
</tbody>
</table>

**Condition Values Signaled**

- **SS$_{ROPRAND}$**
  
  Reserved operand. The MTH$\text{HCOSH}$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$_{FLOOVEMAT}$**
  
  Floating-point overflow in Math Library: the absolute value of floating-point-input-value is greater than about $yyy$; LIB$\text{SIGNAL}$ copies the reserved operand to the signal mechanism vector. The result is the reserved operand -$0.0$ unless a condition handler changes the signal mechanism vector. The value of $yyy$ is 11356.523.
MTH$HEXP—Exponential (H-Floating Value)

The Exponential (H-Floating Value) routine returns the exponential of the input value as an H-floating value.

Format

MTH$HEXP h-exp, floating-point-input-value

JSB Entries

MTH$HEXP_R6

Returns

None.

Arguments

**h-exp**

OpenVMS usage floating_point  
type H_floating  
access write only  
mechanism by reference

Exponential of the input value specified by floating-point-input-value. The h-exp argument is the address of an H-floating number that is this exponential. MTH$HEXP writes the address of the exponential into h-exp.

**floating-point-input-value**

OpenVMS usage floating_point  
type H_floating  
access read only  
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number. For MTH$HEXP, floating-point-input-value specifies an H-floating number.

Description

The exponential of \( x \) is computed as:

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &gt; c(z) )</td>
<td>Overflow occurs</td>
</tr>
<tr>
<td>( x \leq -c(z) )</td>
<td>0</td>
</tr>
<tr>
<td>(</td>
<td>x</td>
</tr>
<tr>
<td>Otherwise</td>
<td>( 2^Y \cdot 2^U \cdot 2^W )</td>
</tr>
</tbody>
</table>

where: \( Y = \text{INTEGER}(x \cdot \ln2(E)) \)  
\( V = \text{FRAC}(x \cdot \ln2(E)) \cdot 16 \)  
\( U = \text{INTEGER}(V)/16 \)  
\( W = \text{FRAC}(V)/16 \cdot 2^W \)  
polynomial approximation of degree 14 for \( z = H \).

See also the section on the hyperbolic cosine for definitions of \( f \) and \( c(z) \).
Condition Values Signaled

SS$_$ROPRAND
Reserved operand. The MTH$_$xEXP routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_$FLOOVMAT
Floating-point overflow in Math Library: floating-point-input-value is greater than yyyy; LIB$SIGNAL copies the reserved operand to the signal mechanism vector. The result is the reserved operand -0.0 unless a condition handler changes the signal mechanism vector.

The value of yyyy is approximately 11355.830 for MTH$_$HEXP.

MTH$_$FLOUNDMAT
Floating-point underflow in Math Library: floating-point-input-value is less than or equal to yyyy and the caller (CALL or JSB) has set hardware floating-point underflow enable. The result is set to 0.0. If the caller has not enabled floating-point underflow (the default), a result of 0.0 is returned but no error is signaled. The value of yyyy is approximately −11356.523 for MTH$_$HEXP.
MTH$HLOG—Natural Logarithm (H-Floating Value)

The Natural Logarithm (H-Floating Value) routine returns the natural (base e) logarithm of the input argument as an H-floating value.

Format

```
MTH$HLOG h-natlog floating-point-input-value
```

JSB Entries

```
MTH$HLOG_R8
```

Returns

None.

Arguments

- **h-natlog**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  - Natural logarithm of `floating-point-input-value`. The `h-natlog` argument is the address of an H-floating number that is this natural logarithm. MTH$HLOG writes the address of this natural logarithm into `h-natlog`.

- **floating-point-input-value**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  - The input value. The `floating-point-input-value` argument is the address of a floating-point number that is this value. For MTH$HLOG, `floating-point-input-value` specifies an H-floating number.

Description

Computation of the natural logarithm routine is based on the following:

1. \( \ln(A' \times Y) = \ln(A') + \ln(Y) \)
2. \( \ln(1 + X) = X - X^2/2 + X^3/3 - X^4/4 + \ldots \) for \( |X| < 1 \)
3. \( \ln(X) = \ln(A) + 2 \times (V^3/3 + V^5/5 + V^7/7 + \ldots) \)
   where \( V = (X - A)/(X + A), A > 0, \) and \( p(y) = 2 \times (1 + y^3/3 + y^5/5 + \ldots) \)

For \( x = 2^n \times f \), where \( n \) is an integer and \( f \) is in the interval of 0.5 to 1, define the following quantities:

- If \( n \geq 1 \), then \( N = n - 1 \) and \( F = 2f \)
- If \( n \leq 0 \), then \( N = n \) and \( F = f \)
From (1) it follows that:

4. \( \ln(X) = N \cdot \ln(2) + \ln(F) \)

Based on the previous relationships, \( z\text{LOG} \) is computed as follows:

1. If \(|F - 1| < 2^{-5}\),
   \[
   z\text{LOG}(X) = N \cdot z\text{LOG}(2) + W + W \cdot p(W),
   \]
   where \( W = F - 1 \).

2. Otherwise,
   \[
   z\text{LOG}(X) = N \cdot z\text{LOG}(2) + z\text{LOG}(A) + V \cdot p(V^2),
   \]
   where \( V = (F - A)/(F + A) \) and \( A \) and \( z\text{LOG}(A) \) are obtained by table look up.

**Condition Values Signaled**

- **SS$_{\text{ROPRAND}}$**: Reserved operand. The MTH$\text{HLOG}$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\text{LOGZERNEG}$**: Logarithm of zero or negative value. Argument **floating-point-input-value** is less than or equal to 0.0. LIB$\text{SIGNAL}$ copies the floating-point reserved operand to the mechanism argument vector CHF$L\text{_MCH_SAVR0/R1}$. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\text{_MCH_SAVR0/R1}$. 
MTH$HLOG2—Base 2 Logarithm (H-Floating Value)

The Base 2 Logarithm (H-Floating Value) routine returns the base 2 logarithm of the input value specified by `floating-point-input-value` as an H-floating value.

Format

MTH$HLOG2 h-log2 , floating-point-input-value

Returns

None.

Arguments

- **h-log2**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  - Base 2 logarithm of `floating-point-input-value`. The `h-log2` argument is the address of an H-floating number that is this base 2 logarithm. MTH$HLOG2 writes the address of this logarithm into `h-log2`.

- **floating-point-input-value**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  - The input value. The `floating-point-input-value` argument is the address of a floating-point number that is this input value. For MTH$HLOG2, `floating-point-input-value` specifies an H-floating number.

Description

The base 2 logarithm function is computed as follows:

\[ z\log_2(X) = z\log_2(E) \times z\log(X) \]

Condition Values Signaled

- SS$_{\text{ROPRAND}}$
  - Reserved operand. The MTH$HLOG2$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$LOGZERNEG

Logarithm of zero or negative value. Argument floating-point-input-value is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$HLOG10—Common Logarithm (H-Floating Value)

The Common Logarithm (H-Floating Value) routine returns the common (base 10) logarithm of the input argument as an H-floating value.

Format

MTH$HLOG10  h-log10, floating-point-input-value

JSB Entries

MTH$HLOG10_R8

Returns

None.

Arguments

h-log10
OpenVMS usage  floating_point
type            H_floating
access          write only
mechanism       by reference

Common logarithm of the input value specified by floating-point-input-value. The h-log10 argument is the address of an H-floating number that is this common logarithm. MTH$HLOG10 writes the address of the common logarithm into h-log10.

floating-point-input-value
OpenVMS usage  floating_point
type            H_floating
access          read only
mechanism       by reference

The input value. The floating-point-input-value argument is the address of a floating-point number. For MTH$HLOG10, floating-point-input-value specifies an H-floating number.

Description

The common logarithm function is computed as follows:

\[ zLOG10(X) = zLOG10(E) \times zLOG(X) \]
Condition Values Signaled

SS$_$ROPRAND  
Reserved operand. The MTH$HLOG10 procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_$LOGZERNeneg  
Logarithm of zero or negative value. Argument `floating-point-input-value` is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$HSIN—Sine of Angle Expressed in Radians (H-Floating Value)

The Sine of Angle Expressed in Radians (H-Floating Value) routine returns the sine of a given angle (in radians) as an H-floating value.

Format

MTH$HSIN h-sine ,angle-in-radians

JSB Entries

MTH$HSIN_R5

Returns

None.

Arguments

h-sine
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

The sine of the angle specified by angle-in-radians. The h-sine argument is the address of an H-floating number that is this sine. MTH$HSIN writes the address of the sine into h-sine.

angle-in-radians
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

Angle (in radians). The angle-in-radians argument is the address of a floating-point number that is this angle. For MTH$HSIN, angle-in-radians specifies an H-floating number.

Description

See the MTH$SINCOS routine for the algorithm used to compute this sine.

Condition Value Signaled

SS$_ROPRAND Reserved operand. The MTH$HSIN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
**MTH$HSIND—Sine of Angle Expressed in Degrees (H-Floating Value)**

The Sine of Angle Expressed in Degrees (H-Floating Value) routine returns the sine of a given angle (in degrees) as an H-floating value.

**Format**

```
MTH$HSIND h-sine ,angle-in-degrees
```

**JSB Entries**

MTH$HSIND_R5

**Returns**

None.

**Arguments**

- **h-sine**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  - Sine of the angle specified by angle-in-degrees. The h-sine argument is the address of an H-floating number that is this sine. MTH$HSIND writes the address of the angle into h-sine.

- **angle-in-degrees**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  - Angle (in degrees). The angle-in-degrees argument is the address of a floating-point number that is this angle. For MTH$HSIND, angle-in-degrees specifies an H-floating number.

**Description**

See MTH$SINCOSD for the algorithm used to compute the sine.

**Condition Values Signaled**

- **SS$ _ROPRAND**
  - Reserved operand. The MTH$HSIND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Floating-point underflow in Math Library. The absolute value of the input angle is less than $\frac{180}{\pi} \times 2^{-m}$ (where $m = 16,384$ for H-floating).
MTH$HSINH—Hyperbolic Sine (H-Floating Value)

The Hyperbolic Sine (H-Floating Value) routine returns the hyperbolic sine of the input value specified by `floating-point-input-value` as an H-floating value.

Format

```
MTH$HSINH h-sinh ,floating-point-input-value
```

Returns

None.

Arguments

- **h-sinh**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  
  Hyperbolic sine of the input value specified by `floating-point-input-value`. The `h-sinh` argument is the address of an H-floating number that is this hyperbolic sine. MTH$HSINH writes the address of the hyperbolic sine into `h-sinh`.

- **floating-point-input-value**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  
  The input value. The `floating-point-input-value` argument is the address of a floating-point number that is this value. For MTH$HSINH, `floating-point-input-value` specifies an H-floating number.

Description

Computation of the hyperbolic sine function depends on the magnitude of the input argument. The range of the function is partitioned using three data type dependent constants: a(z), b(z), and c(z). The subscript z indicates the data type. The constants depend on the number of exponent bits (e) and the number of fraction bits (f) associated with the data type (z).

The values of e and f are as follows:

\[
e = 15
\]

\[
f = 113
\]
The values of the constants in terms of $e$ and $f$ are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a(z)$</td>
<td>$2^{-(f/2)}$</td>
</tr>
<tr>
<td>$b(z)$</td>
<td>$(f + 1)/2 \times \ln(2)$</td>
</tr>
<tr>
<td>$c(z)$</td>
<td>$2^{-1} \times \ln(2)$</td>
</tr>
</tbody>
</table>

Based on the above definitions, $z\text{SINH}(X)$ is computed as follows:

<table>
<thead>
<tr>
<th>Value of $X$</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>$a(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$1.0 \leq</td>
<td>X</td>
</tr>
<tr>
<td>$b(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>$c(z) \leq</td>
<td>X</td>
</tr>
</tbody>
</table>

**Condition Values Signaled**

- **SS$\_ROPRAND**
  - Reserved operand. The MTH$\_HSINH$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$\_FLOOVEMAT**
  - Floating-point overflow in Math Library: the absolute value of floating-point-input-value is greater than $yyy$. LIB$\_SIGNAL$ copies the floating-point reserved operand to the mechanism argument vector CHF$L\_MCH\_SAVR0/R1$. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L\_MCH\_SAVR0/R1$. The value of $yyy$ is approximately 11356.523.
MTH$HSQRT—Square Root (H-Floating Value)

The Square Root (H-Floating Value) routine returns the square root of the input value `floating-point-input-value` as an H-floating value.

**Format**

MTH$HSQRT h-sqrt, floating-point-input-value

**JSB Entries**

MTH$HSQRT_R8

**Returns**

None.

**Arguments**

**h-sqrt**

- **OpenVMS usage**: floating_point
- **type**: H_floating
- **access**: write only
- **mechanism**: by reference

Square root of the input value specified by `floating-point-input-value`. The `h-sqrt` argument is the address of an H-floating number that is this square root. MTH$HSQRT writes the address of the square root into `h-sqrt`.

**floating-point-input-value**

- **OpenVMS usage**: floating_point
- **type**: H_floating
- **access**: read only
- **mechanism**: by reference

Input value. The `floating-point-input-value` argument is the address of a floating-point number that contains this input value. For MTH$HSQRT, `floating-point-input-value` specifies an H-floating number.

**Description**

The square root of \( X \) is computed as follows:

If \( X < 0 \), an error is signaled.

Let \( X = 2^K \times F \)

where:

- \( K \) is the exponential part of the floating-point data
- \( F \) is the fractional part of the floating-point data

If \( K \) is even:

\[
X = 2^{(2 \times P)} \times F,
\]

\[
zSQRT(X) = 2^P \times zSQRT(F),
\]

\[
1/2 \leq F < 1, \text{ where } P = K/2
\]
If $K$ is odd:
\[ X = 2^{(2P+1)} \cdot F = 2^{(2P+2)} \cdot (F/2), \]
\[ z\sqrt{X} = 2^{(P+1)} \cdot z\sqrt{F/2}, \]
\[ 1/4 < F/2 < 1/2, \text{ where } p = (K-1)/2 \]

Let $F' = A \cdot F + B$, when $K$ is even:

- $A = 0.95F6198$ (hex)
- $B = 0.6BA5918$ (hex)

Let $F' = A \cdot (F/2) + B$, when $K$ is odd:

- $A = 0.D413CCC$ (hex)
- $B = 0.4C1E248$ (hex)

Let $K' = P$, when $K$ is even
Let $K' = P+1$, when $K$ is odd

Let $Y[0] = 2^{K'} \cdot F'$ be a straight line approximation within the given interval using coefficients $A$ and $B$ which minimize the absolute error at the midpoint and endpoint.

Starting with $Y[0]$, $n$ Newton-Raphson iterations are performed:

\[ Y[n+1] = 1/2 \cdot (Y[n] + X/Y[n]) \]

where $n = 5$ for H-floating.

**Condition Values Signaled**

**SS$\_ROPRAND**

Reserved operand. The MTH$\_HSQRT$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$\_SQUROONEG**

Square root of negative number. Argument `floating-point-input-value` is less than 0.0. LIB$\_SIGNAL$ copies the floating-point reserved operand to the mechanism argument vector `CHF$L_\_MCH_SAVR0/R1`. The result is the floating-point reserved operand unless you have written a condition handler to change `CHF$L_\_MCH_SAVR0/R1`.
MTH$HTAN—Tangent of Angle Expressed in Radians (H-Floating Value)

The Tangent of Angle Expressed in Radians (H-Floating Value) routine returns the tangent of a given angle (in radians) as an H-floating value.

Format

MTH$HTAN  h-tan ,angle-in-radians

JSB Entries

MTH$HTAN_R5

Returns

None.

Arguments

- **h-tan**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  - Tangent of the angle specified by *angle-in-radians*. The *h-tan* argument is the address of an H-floating number that is this tangent. MTH$HTAN writes the address of the tangent into *h-tan*.

- **angle-in-radians**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  - The input angle (in radians). The *angle-in-radians* argument is the address of a floating-point number that is this angle. For MTH$HTAN, *angle-in-radians* specifies an H-floating number.

Description

When the input argument is expressed in radians, the tangent function is computed as follows:

1. If $|X| < 2^{(-1/2)}$, then $zTAN(X) = X$ (see the section on MTH$zCOSH for the definition of $f$)
2. Otherwise, call MTH$zSINCOS to obtain $zSIN(X)$ and $zCOS(X)$; then
   a. If $zCOS(X) = 0$, signal overflow
   b. Otherwise, $zTAN(X) = zSIN(X)/zCOS(X)$
MTH$HTAN

Condition Values Signaled

SS$_ROPRAND
Reserved operand. The MTH$HTAN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_FLOOVEMAT
Floating-point overflow in math library.
### MTH$HTAND—Tangent of Angle Expressed in Degrees (H-Floating Value)

The Tangent of Angle Expressed in Degrees (H-Floating Value) routine returns the tangent of a given angle (in degrees) as an H-floating value.

**Format**

```
MTH$HTAND h-tan ,angle-in-degrees
```

**JSB Entries**

```
MTH$HTAND_R5
```

**Returns**

None.

**Arguments**

- **h-tan**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: write only
  - mechanism: by reference
  - Tangent of the angle specified by **angle-in-degrees**. The **h-tan** argument is the address of an H-floating number that is this tangent. MTH$HTAND writes the address of the tangent into **h-tan**.

- **angle-in-degrees**
  - OpenVMS usage: floating_point
  - type: H_floating
  - access: read only
  - mechanism: by reference
  - The input angle (in degrees). The **angle-in-degrees** argument is the address of a floating-point number which is this angle. For MTH$HTAND, **angle-in-degrees** specifies an H-floating number.

**Description**

When the input argument is expressed in degrees, the tangent function is computed as follows:

1. If $|X| < (180/\pi) \cdot 2^{\text{-}2/(e-1)}$ and underflow signaling is enabled, underflow is signaled (see the section on MTH$zCOSH for the definition of $e$).
2. Otherwise, if $|X| < (180/\pi) \cdot 2^{\text{-}f/2}$, then $zTAND(X) = (\pi/180) \cdot X$. See the description of MTH$zCOSH for the definition of $f$.
3. Otherwise, call MTH$zSINCOSD to obtain $zSIND(X)$ and $zCOSD(X)$.
   a. Then, if $zCOSD(X) = 0$, signal overflow
   b. Else, $zTAND(X) = zSIND(X)/zCOSD(X)$
Condition Values Signaled

SS$_$ROPRAND

Reserved operand. The MTH$HTAND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$FLOOVEMAT

Floating-point overflow in math library.
MTH$HTANH—Compute the Hyperbolic Tangent (H-Floating Value)

The Compute the Hyperbolic Tangent (H-Floating Value) routine returns the hyperbolic tangent of the input value as an H-floating value.

Format

MTH$HTANH h-tanh ,floating-point-input-value

Returns

None.

Arguments

h-tanh
OpenVMS usage floating_point
type H_floating
access write only
mechanism by reference

Hyperbolic tangent of the value specified by floating-point-input-value. The h-tanh argument is the address of a H-floating number that is this hyperbolic tangent. MTH$HTANH writes the address of the hyperbolic tangent into h-tanh.

floating-point-input-value
OpenVMS usage floating_point
type H_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number that contains this input value. For MTH$HTANH, floating-point-input-value specifies an H-floating number.

Description

For MTH$HTANH, the hyperbolic tangent of X is computed using a value of 56 for g and a value of 40 for h. The hyperbolic tangent of X is computed as follows:

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Hyperbolic Tangent Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2^-9 &lt;</td>
<td>X</td>
</tr>
<tr>
<td>0.25 &lt;</td>
<td>X</td>
</tr>
<tr>
<td>h ≤</td>
<td>X</td>
</tr>
</tbody>
</table>
Reserved operand. The MTH$HTANH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$xlMAG—Imaginary Part of a Complex Number

The Imaginary Part of a Complex Number routine returns the imaginary part of a complex number.

Format

MTH$AIMAG complex-number
MTH$DIMAG complex-number
MTH$GIMAG complex-number

Each of the above three formats corresponds to one of the three floating-point complex types.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Imaginary part of the input complex-number. MTH$AIMAG returns an F-floating number. MTH$DIMAG returns a D-floating number. MTH$GIMAG returns a G-floating number.

ARGUMENT

complex-number
OpenVMS usage complex_number
type F_floating complex, D_floating complex, G_floating complex
access read only
mechanism by reference

The input complex number. The complex-number argument is the address of this floating-point complex number. For MTH$AIMAG, complex-number specifies an F-floating number. For MTH$DIMAG, complex-number specifies a D-floating number. For MTH$GIMAG, complex-number specifies a G-floating number.

Condition Value Signaled

SS$_ROPRAND

Reserved operand. The MTH$xlMAG routine encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
This FORTRAN example demonstrates a procedure call to MTH$GIMAG. Because this example uses G-floating numbers, it must be compiled with the statement "FORTRAN/G filename".

The output generated by this program is as follows:

The complex number z is (0.8535407185554504, 0.2043401598930359)
It has imaginary part 0.2043401598930359
MTH$XLOG—Natural Logarithm

The Natural Logarithm routine returns the natural (base e) logarithm of the input argument.

Format

MTH$ALOG  floating-point-input-value
MTH$DLOG  floating-point-input-value
MTH$GLOG  floating-point-input-value

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$ALOG_R5
MTH$DLOG_R8
MTH$GLOG_R8

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point

type F_floating, D_floating, G_floating
access write only
mechanism by value

The natural logarithm of floating-point-input-value. MTH$ALOG returns an F-floating number. MTH$DLOG returns a D-floating number. MTH$GLOG returns a G-floating number.

Arguments

floating-point-input-value

OpenVMS usage floating_point

type F_floating, D_floating, G_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number that is this value. For MTH$ALOG, floating-point-input-value specifies an F-floating number. For MTH$DLOG, floating-point-input-value specifies a D-floating number. For MTH$GLOG, floating-point-input-value specifies a G-floating number.

Description

Computation of the natural logarithm routine is based on the following:

1. \( \ln(X \times Y) = \ln(X) + \ln(Y) \)
2. \( \ln(1 + X) = X - X^2/2 + X^3/3 - X^4/4 \ldots \) for \( |X| < 1 \)
3. \( \ln(X) = \ln(A) + 2 \times (V^3/3 + V^5/5 + V^7/7 + \ldots) \)
   \[ = \ln(A) + V \times p(V^2), \text{ where } V = (X - A)/(X + A), \]
   \( A > 0, \text{ and } p(y) = 2 \times (1 + y/3 + y^2/5 + \ldots) \)

For \( x = 2^n \times f, \) where \( n \) is an integer and \( f \) is in the interval of 0.5 to 1, define the following quantities:

\[ \text{If } n \geq 1, \text{ then } N = n - 1 \text{ and } F = 2f \]
\[ \text{If } n \leq 0, \text{ then } N = n \text{ and } F = f \]

From (1) above it follows that:

4. \( \ln(X) = N \times \ln(2) + \ln(F) \)

Based on the above relationships, \( z\text{LOG} \) is computed as follows:

1. If \( |F - 1| < 2^{-5}, z\text{LOG}(X) = N \times z\text{LOG}(2) + W + W \times p(W), \)
   where \( W = F - 1. \)
2. Otherwise, \( z\text{LOG}(X) = N \times z\text{LOG}(2) + z\text{LOG}(A) + V \times p(V^2), \)
   where \( V = (F - A)/(F + A) \) and \( A \) and \( z\text{LOG}(A) \)
   are obtained by table look up.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HLOG.

**Condition Values Signaled**

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$_LOGZERNEG</td>
<td>Logarithm of zero or negative value. Argument floating-point-input-value is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.</td>
</tr>
<tr>
<td>SS$_ROPRAND</td>
<td>Reserved operand. The MTH$xLOG procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
</tbody>
</table>
MTH$xLOG2—Base 2 Logarithm

The Base 2 Logarithm routine returns the base 2 logarithm of the input value specified by floating-point-input-value.

Format

MTH$ALOG2 floating-point-input-value
MTH$DLOG2 floating-point-input-value
MTH$GLOG2 floating-point-input-value

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

The base 2 logarithm of floating-point-input-value. MTH$ALOG2 returns an F-floating number. MTH$DLOG2 returns a D-floating number. MTH$GLOG2 returns a G-floating number.

Arguments

floating-point-input-value
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number that is this input value. For MTH$ALOG2, floating-point-input-value specifies an F-floating number. For MTH$DLOG2, floating-point-input-value specifies a D-floating number. For MTH$GLOG2, floating-point-input-value specifies a G-floating number.

Description

The base 2 logarithm function is computed as follows:

\[ z \log_2(X) = z \log_2(E) \cdot z \log(X) \]

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HLOG2.
Condition Values Signaled

SS$_ROPRAND

Reserved operand. The MTH$xLOG2 procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$_LOGZERNEG

Logarithm of zero or negative value. Argument floating-point-input-value is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$LOG10—Common Logarithm

The Common Logarithm routine returns the common (base 10) logarithm of the input argument.

Format

MTH$ALOG10 floating-point-input-value
MTH$DLOG10 floating-point-input-value
MTH$GLOG10 floating-point-input-value

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$ALOG10_R5
MTH$DLOG10_R8
MTH$GLOG10_R8

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

The common logarithm of floating-point-input-value. MTH$ALOG10 returns an F-floating number. MTH$DLOG10 returns a D-floating number. MTH$GLOG10 returns a G-floating number.

Arguments

floating-point-input-value
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number. For MTH$ALOG10, floating-point-input-value specifies an F-floating number. For MTH$DLOG10, floating-point-input-value specifies a D-floating number. For MTH$GLOG10, floating-point-input-value specifies a G-floating number.

Description

The common logarithm function is computed as follows:

$$z \log_{10}(X) = z \log_{10}(E) \cdot z \log(X)$$

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HLOG10.
Condition Values Signaled

**SS$_ROPRAND**

Reserved operand. The MTH$xLOG10 procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_LOGZERNEG**

Logarithm of zero or negative value. Argument `floating-point-input-value` is less than or equal to 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
MTH$RANDOM—Random Number Generator, Uniformly Distributed

The Random Number Generator, Uniformly Distributed routine is a general random number generator.

Format

MTH$RANDOM seed

Returns

OpenVMS usage floating_point
type F_floating
access write only
mechanism by value

MTH$RANDOM returns an F-floating random number.

ARGUMENT

seed
OpenVMS usage longword_unsigned
type longword (unsigned)
access modify
mechanism by reference

The integer seed, a 32-bit number whose high-order 24 bits are converted by MTH$RANDOM to an F-floating random number. The seed argument is the address of an unsigned longword that contains this integer seed. The seed is modified by each call to MTH$RANDOM.

Description

This routine must be called again to obtain the next pseudorandom number. The seed is updated automatically.

The result is a floating-point number that is uniformly distributed between 0.0 inclusively and 1.0 exclusively.

There are no restrictions on the seed, although it should be initialized to different values on separate runs in order to obtain different random sequences. MTH$RANDOM uses the following method to update the seed passed as the argument:

$$SEED = (69069 \times SEED + 1) \pmod{2^{32}}$$

Condition Value Signaled

SS$_{\text{ROPRAND}}$ Reserved operand. The MTH$RANDOM procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Example

MTH$RANDOM

```pli
RAND: PROCEDURE OPTIONS (MAIN);
DECLARE FOR$SECNDS ENTRY (FLOAT BINARY (24))
    RETURNS (FLOAT BINARY (24));
DECLARE MTH$RANDOM ENTRY (FIXED BINARY (31))
    RETURNS (FLOAT BINARY (24));
DECLARE TIME FLOAT BINARY (24);
DECLARE SEED FIXED BINARY (31);
DECLARE I FIXED BINARY (7);
DECLARE RESULT FIXED DECIMAL (2);
/* Get floating random time value */
TIME = FOR$SECNDS (0E0);
/* Convert to fixed */
SEED = TIME;
/* Generate 100 random numbers between 1 and 10 */
DO I = 1 TO 100;
    RESULT = 1 + FIXED ( (10E0 * MTH$RANDOM (SEED) ) , 31 );
    PUT LIST (RESULT);
END;
END RAND;
```

This PL/I program demonstrates the use of MTH$RANDOM. The value returned by FOR$SECNDS is used as the seed for the random-number generator to ensure a different sequence each time the program is run. The random value returned is scaled so as to represent values between 1 and 10.

Because this program generates random numbers, the output generated will be different each time the program is executed. One example of the output generated by this program is as follows:

```
7 4 6 5 9 10 5 5 3 8 8 1 3 1 3 2
4 4 2 4 4 8 3 8 9 1 7 1 8 6 9 10
1 10 10 6 7 3 2 2 1 2 6 6 3 9 5 8
6 2 3 6 10 8 5 5 4 2 8 5 9 6 4 2
8 5 4 9 8 7 6 6 8 10 9 5 9 4 5 7
1 2 2 3 6 5 2 3 4 4 8 9 2 8 5 5
3 8 1 5
```

MTH-120
MTH$xREAL—Real Part of a Complex Number

The Real Part of a Complex Number routine returns the real part of a complex number.

Format

MTH$REAL complex-number
MTH$DREAL complex-number
MTH$GREAL complex-number

Each of the above three formats accepts one of the three floating-point complex types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Real part of the complex number. MTH$REAL returns an F-floating number. MTH$DREAL returns a D-floating number. MTH$GREAL returns a G-floating number.

ARGUMENT

complex-number

OpenVMS usage complex_number
type F_floating complex, D_floating complex, G_floating complex
access read only
mechanism by reference

The complex number whose real part is returned by MTH$REAL. The complex-number argument is the address of this floating-point complex number. For MTH$REAL, complex-number is an F-floating complex number. For MTH$DREAL, complex-number is a D-floating complex number. For MTH$GREAL, complex-number is a G-floating complex number.

Condition Value Signaled

SS$_ROPRAND

Reserved operand. The MTH$xREAL procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Example

This FORTRAN example forms the real part of an F-floating complex number using MTH$REAL and the FORTRAN random number generator RAN.

Declare Z as a complex value and MTH$REAL as a REAL*4 value. MTH$REAL will return the real part of Z: \( Z_{\text{NEW}} = \text{MTH$REAL}(Z) \).

```fortran
COMPLEX Z
COMPLEX CMPLX
REAL*4 MTH$REAL
INTEGER M
M = 1234567

GENERATE A RANDOM COMPLEX NUMBER WITH THE FORTRAN GENERIC CMPLX.

\[ Z = \text{CMPLX}(\text{RAN}(M), \text{RAN}(M)) \]

\( Z \) is a complex number \((r,i)\) with real part \(r\) and imaginary part \(i\).

```

This FORTRAN example demonstrates the use of MTH$REAL. The output of this program is as follows:

The complex number \( z \) is \((0.8535407,0.2043402)\)
It has real part 0.8535407
MTH$xSIN—Sine of Angle Expressed in Radians

The Sine of Angle Expressed in Radians routine returns the sine of a given angle (in radians).

Format

MTH$SIN angle-in-radians
MTH$DSIN angle-in-radians
MTH$GSIN angle-in-radians

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$SIN_R4
MTH$DSIN_R7
MTH$GSIN_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

Sine of the angle specified by angle-in-radians. MTH$SIN returns an F-floating number. MTH$DSIN returns a D-floating number. MTH$GSIN returns a G-floating number.

Arguments

angle-in-radians
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Angle (in radians). The angle-in-radians argument is the address of a floating-point number that is this angle. For MTH$SIN, angle-in-radians specifies an F-floating number. For MTH$DSIN, angle-in-radians specifies a D-floating number. For MTH$GSIN, angle-in-radians specifies a G-floating number.

Description

See the MTH$SINCOS routine for the algorithm used to compute this sine.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HSIN.
Reserved operand. The MTH$xSIN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$xSINCOS—Sine and Cosine of Angle Expressed in Radians

The Sine and Cosine of Angle Expressed in Radians routine returns the sine and cosine of a given angle (in radians).

Format

MTH$SINCOS angle-in-radians ,sine ,cosine
MTH$DSINCOS angle-in-radians ,sine ,cosine
MTH$GSINCOS angle-in-radians ,sine ,cosine
MTH$HSINCOS angle-in-radians ,sine ,cosine

Each of the above four formats accepts one of the four floating-point types as input.

JSB Entries

MTH$SINCOS_R5
MTH$DSINCOS_R7
MTH$GSINCOS_R7
MTH$HSINCOS_R7

Each of the above four JSB entries accepts one of the four floating-point types as input.

Returns

MTH$SINCOS, MTH$DSINCOS, MTH$GSINCOS, and MTH$HSINCOS return the sine and cosine of the input angle by reference in the sine and cosine arguments.

Arguments

angle-in-radians
OpenVMS usage floating_point
type F_floating, D_floating, G_floating, H_floating
access read only
mechanism by reference

Angle (in radians) whose sine and cosine are to be returned. The angle-in-radians argument is the address of a floating-point number that is this angle. For MTH$SINCOS, angle-in-radians is an F-floating number. For MTH$DSINCOS, angle-in-radians is a D-floating number. For MTH$GSINCOS, angle-in-radians is a G-floating number. For MTH$HSINCOS, angle-in-radians is an H-floating number.

sine
OpenVMS usage floating_point
type F_floating, D_floating, G_floating, H_floating
access write only
mechanism by reference
Sine of the angle specified by `angle-in-radians`. The `sine` argument is the address of a floating-point number. `MTH$SINCOS` writes an F-floating number into `sine`. `MTH$DSINCOS` writes a D-floating number into `sine`. `MTH$GSINCOS` writes a G-floating number into `sine`. `MTH$HSINCOS` writes an H-floating number into `sine`.

**cosine**

OpenVMS usage floating_point  
type  F_floating, D_floating, G_floating, H_floating  
access write only  
mechanism by reference

Cosine of the angle specified by `angle-in-radians`. The `cosine` argument is the address of a floating-point number. `MTH$SINCOS` writes an F-floating number into `cosine`. `MTH$DSINCOS` writes a D-floating number into `cosine`. `MTH$GSINCOS` writes a G-floating number into `cosine`. `MTH$HSINCOS` writes an H-floating number into `cosine`.

**Description**

All routines with JSB entry points accept a single argument in R0:Rm, where `m`, which is defined below, is dependent on the data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_floating</td>
<td>0</td>
</tr>
<tr>
<td>D_floating</td>
<td>1</td>
</tr>
<tr>
<td>G_floating</td>
<td>1</td>
</tr>
<tr>
<td>H_floating</td>
<td>3</td>
</tr>
</tbody>
</table>

In general, Run-Time Library routines with JSB entry points return one value in R0:Rm. The `MTH$SINCOS` routine returns two values, however. The sine of `angle-in-radians` is returned in R0:Rm and the cosine of `angle-in-radians` is returned in (R<m+l>:R<2*m+l>).

In radians, the computation of zSIN(X) and zCOS(X) is based on the following polynomial expansions:

\[
\sin(X) = X - X^3/(3!) + X^5/(5!) - X^7/(7!) + \ldots = X + X \cdot P(X^2), \text{ where} \\
P(y) = y/(3!) + y^2/(5!) + y^3/(7!) + \ldots
\]

\[
\cos(X) = 1 - X^2/(2!) + X^4/(4!) - X^6/(6!) + \ldots = Q(X^2), \text{ where} \\
Q(y) = (1 - y/(2!)) + y^2/(4!) + y^3/(6!) + \ldots
\]

1. If \(|X| < 2^{-f/2}\),
   then zSIN(X) = X and zCOS(X) = 1  
   (see the section on MTH$zCOSH for the definition of f)

2. If \(2^{-f/2} \leq |X| < \pi/4\),
   then zSIN(X) = X + P(X^2)  
   and zCOS(X) = Q(X^2)
3. If $\pi/4 \leq |X|$ and $X > 0$,
   a. Let $J = \text{INT}(X/(\pi/4))$
      and $I = J \bmod 8$
   b. If $J$ is even, let $Y = X - J \cdot (\pi/4)$
      otherwise, let $Y = (J + 1) \cdot (\pi/4) - X$

With the above definitions, the following table relates $\sin(X)$ and $\cos(X)$ to $\sin(Y)$ and $\cos(Y)$:

<table>
<thead>
<tr>
<th>Value of $I$</th>
<th>$\sin(X)$</th>
<th>$\cos(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\sin(Y)$</td>
<td>$\cos(Y)$</td>
</tr>
<tr>
<td>1</td>
<td>$\cos(Y)$</td>
<td>$\sin(Y)$</td>
</tr>
<tr>
<td>2</td>
<td>$\cos(Y)$</td>
<td>$-\sin(Y)$</td>
</tr>
<tr>
<td>3</td>
<td>$\sin(Y)$</td>
<td>$-\cos(Y)$</td>
</tr>
<tr>
<td>4</td>
<td>$-\sin(Y)$</td>
<td>$-\cos(Y)$</td>
</tr>
<tr>
<td>5</td>
<td>$-\cos(Y)$</td>
<td>$-\sin(Y)$</td>
</tr>
<tr>
<td>6</td>
<td>$-\cos(Y)$</td>
<td>$\sin(Y)$</td>
</tr>
<tr>
<td>7</td>
<td>$-\sin(Y)$</td>
<td>$\cos(Y)$</td>
</tr>
</tbody>
</table>

c. $\sin(Y)$ and $\cos(Y)$ are computed as follows:
   
   $\sin(Y) = Y + P(Y^2)$,
   and $\cos(Y) = Q(Y^2)$

4. If $\pi/4 \leq |X|$ and $X < 0$,
   then $\sin(X) = -\sin(|X|)$
   and $\cos(X) = \cos(|X|)$

**Condition Value Returned**

**SS$_{\text{ROPRAND}}$**

Reserved operand. The MTHS$x\text{SINCOS}$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
3. If $(180/\pi) \cdot 2^{(-I/2)} \leq |X| < 45$
   then $z\text{SIND}(X) = X/2^6 + P(X^2)$
   and $z\text{COSD}(X) = Q(X^2)$

4. If $45 \leq |X|$ and $X > 0$,
   
   a. Let $J = \text{INT}(X/(45))$ and
      \[ I = J \text{ modulo } 8 \]
   
   b. If $J$ is even, let $Y = X - J \cdot 45$;
      otherwise, let $Y = (J + 1) \cdot 45 - X$.

   With the above definitions, the following table relates
   $z\text{SIND}(X)$ and $z\text{COSD}(X)$ to $z\text{SIND}(Y)$ and $z\text{COSD}(Y)$:

<table>
<thead>
<tr>
<th>Value of $I$</th>
<th>$z\text{SIND}(Y)$</th>
<th>$z\text{COSD}(Y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$z\text{SIND}(Y)$</td>
<td>$z\text{COSD}(Y)$</td>
</tr>
<tr>
<td>1</td>
<td>$z\text{COSD}(Y)$</td>
<td>$z\text{SIND}(Y)$</td>
</tr>
<tr>
<td>2</td>
<td>$z\text{COSD}(Y)$</td>
<td>$-z\text{SIND}(Y)$</td>
</tr>
<tr>
<td>3</td>
<td>$z\text{SIND}(Y)$</td>
<td>$-z\text{COSD}(Y)$</td>
</tr>
<tr>
<td>4</td>
<td>$-z\text{SIND}(Y)$</td>
<td>$-z\text{COSD}(Y)$</td>
</tr>
<tr>
<td>5</td>
<td>$-z\text{COSD}(Y)$</td>
<td>$-z\text{SIND}(Y)$</td>
</tr>
<tr>
<td>6</td>
<td>$-z\text{COSD}(Y)$</td>
<td>$z\text{SIND}(Y)$</td>
</tr>
<tr>
<td>7</td>
<td>$-z\text{SIND}(Y)$</td>
<td>$z\text{COSD}(Y)$</td>
</tr>
</tbody>
</table>

   c. $z\text{SIND}(Y)$ and $z\text{COSD}(Y)$ are computed as follows:
      \[ z\text{SIND}(Y) = Y/2^6 + P(Y^2) \]
      \[ z\text{COSD}(Y) = Q(Y^2) \]

   d. If $45 \leq |X|$ and $X < 0$,
      then $z\text{SIND}(X) = -z\text{SIND}(|X|)$
      and $z\text{COSD}(X) = z\text{COSD}(|X|)$

Condition Values Signaled

SS$_{ROPRAND}$

Reserved operand. The MTH$xSINCOSD$
procedure encountered a floating-point reserved
operand due to incorrect user input. A floating-
point reserved operand is a floating-point datum
with a sign bit of 1 and a biased exponent of 0.
Floating-point reserved operands are reserved for
future use by Digital.

MTH$_{FLOUNDMAT}$

Floating-point underflow in math library. The
absolute value of the input angle is less than
$180/\pi \cdot 2^{-m}$ (where $m = 128$ for F-floating and
D-floating, 1,024 for G-floating, and 16,384 for
H-floating).
MTH$xSIND—Sine of Angle Expressed in Degrees

The Sine of Angle Expressed in Degrees routine returns the sine of a given angle (in degrees).

Format

MTH$SIND angle-in-degrees
MTH$DSIND angle-in-degrees
MTH$GSIND angle-in-degrees

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$SIND_R4
MTH$DSIND_R7
MTH$GSIND_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

The sine of the angle. MTH$SIND returns an F-floating number. MTH$DSIND returns a D-floating number. MTH$GSIND returns a G-floating number.

Arguments

angle-in-degrees
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

Angle (in degrees). The angle-in-degrees argument is the address of a floating-point number that is this angle. For MTH$SIND, angle-in-degrees specifies an F-floating number. For MTH$DSIND, angle-in-degrees specifies a D-floating number. For MTH$GSIND, angle-in-degrees specifies a G-floating number.

Description

See MTH$SINCOSD for the algorithm that is used to compute the sine.

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HSIND.
Condition Values Signaled

**SS$_$ROPRAND**
Reserved operand. The MTH$SIND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

**MTH$_$FLOUNDMAT**
Floating-point underflow in math library. The absolute value of the input angle is less than $180/\pi \cdot 2^{-m}$ (where $m = 128$ for F-floating and D-floating, and 1,024 for G-floating).
MTH$xSINH—Hyperbolic Sine

The Hyperbolic Sine routine returns the hyperbolic sine of the input value specified by floating-point-input-value.

Format

MTH$SINH  floating-point-input-value
MTH$DSINH  floating-point-input-value
MTH$GSINH  floating-point-input-value

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access write only
mechanism by value

The hyperbolic sine of floating-point-input-value. MTH$SINH returns an F-floating number. MTH$DSINH returns a D-floating number. MTH$GSINH returns a G-floating number.

Arguments

floating-point-input-value
OpenVMS usage floating_point
type F_floating, D_floating, G_floating
access read only
mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number that is this value. For MTH$SINH, floating-point-input-value specifies an F-floating number. For MTH$DSINH, floating-point-input-value specifies a D-floating number. For MTH$GSINH, floating-point-input-value specifies a G-floating number.

Description

Computation of the hyperbolic sine function depends on the magnitude of the input argument. The range of the function is partitioned using four data type dependent constants: a(z), b(z), and c(z). The subscript z indicates the data type. The constants depend on the number of exponent bits (e) and the number of fraction bits (f) associated with the data type (z).
The values of \( e \) and \( f \) are:

<table>
<thead>
<tr>
<th>( z )</th>
<th>( e )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F )</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>( D )</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>( G )</td>
<td>11</td>
<td>53</td>
</tr>
</tbody>
</table>

The values of the constants in terms of \( e \) and \( f \) are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(z) )</td>
<td>( 2^{(-f/2)} )</td>
</tr>
<tr>
<td>( b(z) )</td>
<td>( \text{CEILING}[(f + 1)/2 \cdot \ln(2)] )</td>
</tr>
<tr>
<td>( c(z) )</td>
<td>( (2^{(e-1)} \cdot \ln(2)) )</td>
</tr>
</tbody>
</table>

Based on the above definitions, \( z\text{SINH}(X) \) is computed as follows:

<table>
<thead>
<tr>
<th>Value of ( X )</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>X</td>
</tr>
<tr>
<td>( a(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>( 1.0 \leq</td>
<td>X</td>
</tr>
<tr>
<td>( b(z) \leq</td>
<td>X</td>
</tr>
<tr>
<td>( c(z) \leq</td>
<td>X</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$\text{HSINH}$.

**Condition Values Signaled**

**SS$_\text{ROPRAND}$**

Reserved operand. The MTH$\text{xSINH}$ procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
Floating-point overflow in Math Library: the absolute value of \texttt{floating-point-input-value} is greater than \texttt{yyy}. \texttt{LIB$\text{SIGNAL}} copies the floating-point reserved operand to the mechanism argument vector \texttt{CHF$L\_MCH\_SAVR0/R1}. The result is the floating-point reserved operand unless you have written a condition handler to change \texttt{CHF$L\_MCH\_SAVR0/R1}.

The values of \texttt{yyy} are approximately:

- \texttt{MTH$\text{SINH}—88.722}
- \texttt{MTH$\text{DSINH—88.722}}
- \texttt{MTH$\text{GSINH—709.782}}
**MTH$xSQRT**

---

**MTH$xSQRT—Square Root**

The Square Root routine returns the square root of the input value `floating-point-input-value`.

**Format**

- `MTH$SQRT floating-point-input-value`
- `MTH$DSQRT floating-point-input-value`
- `MTH$GSQRT floating-point-input-value`

Each of the above formats accepts one of the floating-point types as input.

**JSB Entries**

- `MTH$SQRT_R3`
- `MTH$DSQRT_R5`
- `MTH$GSQRT_R5`

Each of the above JSB entries accepts one of the floating-point types as input.

**Returns**

OpenVMS usage `floating_point`

- Type: `F_floating, D_floating, G_floating`
- Access: `write only`
- Mechanism: `by value`

The square root of `floating-point-input-value`. `MTH$SQRT` returns an F-floating number. `MTH$DSQRT` returns a D-floating number. `MTH$GSQRT` returns a G-floating number.

**Arguments**

`floating-point-input-value`

OpenVMS usage `floating_point`

- Type: `F_floating, D_floating, G_floating`
- Access: `read only`
- Mechanism: `by reference`

Input value. The `floating-point-input-value` argument is the address of a floating-point number that contains this input value. For `MTH$SQRT`, `floating-point-input-value` specifies an F-floating number. For `MTH$DSQRT`, `floating-point-input-value` specifies a D-floating number. For `MTH$GSQRT`, `floating-point-input-value` specifies a G-floating number.

**Description**

The square root of $X$ is computed as follows:

If $X < 0$, an error is signaled.

Let $X = 2^K \ast F$

where:
K is the exponential part of the floating-point data.
F is the fractional part of the floating-point data.
If K is even:
\[ X = 2^{2P} \times F, \]
\[ z\text{SQRT}(X) = 2^P \times z\text{SQRT}(F), \]
\[ 1/2 \leq F < 1, \text{ where } P = K/2 \]
If K is odd:
\[ X = 2^{(2P+1)} \times F = 2^{(2P+2)} \times (F/2), \]
\[ z\text{SQRT}(X) = 2^{(P+1)} \times z\text{SQRT}(F/2), \]
\[ 1/4 \leq F/2 < 1/2, \text{ where } p = (K-1)/2 \]
Let \( F' = A \times F + B \), when K is even:
\[ A = 0.95F6198 \text{ (hex)} \]
\[ B = 0.6BA5918 \text{ (hex)} \]
Let \( F' = A \times (F/2) + B \), when K is odd:
\[ A = 0.D413CCC \text{ (hex)} \]
\[ B = 0.4C1E248 \text{ (hex)} \]
Let \( K' = P \), when K is even
Let \( K' = P+1 \), when K is odd
Let \( Y[0] = 2^{K'} \times F' \) be a straight line approximation within the given interval using coefficients A and B which minimize the absolute error at the midpoint and endpoint.
Starting with \( Y[0] \), n Newton-Raphson iterations are performed:
\[ Y[n + 1] = 1/2 \times (Y[n] + X/Y[n]) \]
where \( n = 2, 3, \text{ or } 3 \) for \( z = \text{F-floating, D-floating, or G-floating} \), respectively.
The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HSQRT.

Condition Values Signaled

SS$\_ROPRAND

Reserved operand. The MTH$xSQRT procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

MTH$\_SQUROONEG

Square root of negative number. Argument \( \text{floating-point-input-value} \) is less than 0.0. LIB$SIGNAL copies the floating-point reserved operand to the mechanism argument vector CHF$L_MCH_SAVR0/R1. The result is the floating-point reserved operand unless you have written a condition handler to change CHF$L_MCH_SAVR0/R1.
The Tangent of Angle Expressed in Radians routine returns the tangent of a given angle (in radians).

Format

MTH$TAN  angle-in-radians
MTH$DTAN  angle-in-radians
MTH$GTAN  angle-in-radians

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$TAN_R4
MTH$DTAN_R7
MTH$GTAN_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage: floating_point
type: F_floating, D_floating, G_floating
access: write only
mechanism: by value

The tangent of the angle specified by angle-in-radians. MTH$TAN returns an F-floating number. MTH$DTAN returns a D-floating number. MTH$GTAN returns a G-floating number.

Arguments

angle-in-radians
OpenVMS usage: floating_point
type: F_floating, D_floating, G_floating
access: read only
mechanism: by reference

The input angle (in radians). The angle-in-radians argument is the address of a floating-point number that is this angle. For MTH$TAN, angle-in-radians specifies an F-floating number. For MTH$DTAN, angle-in-radians specifies a D-floating number. For MTH$GTAN, angle-in-radians specifies a G-floating number.

Description

When the input argument is expressed in radians, the tangent function is computed as follows:

1. If $|X| < 2^{-f/2}$, then $\tan(X) = X$ (see the section on MTH$zCOSH for the definition of $f$)
2. Otherwise, call MTH$xSINCOS to obtain zSIN(X) and zCOS(X); then
   a. If zCOS(X) = 0, signal overflow
   b. Otherwise, zTAN(X) = zSIN(X)/zCOS(X)

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HTAN.

**Condition Values Signaled**

- **SS$$_{ROPRAND}**
  - Reserved operand. The MTH$xTAN procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.

- **MTH$$_{FLOOVEMAT}**
  - Floating-point overflow in Math Library.
MTH$TAND—Tangent of Angle Expressed in Degrees

The Tangent of Angle Expressed in Degrees routine returns the tangent of a given angle (in degrees).

Format

MTH$TAND  angle-in-degrees
MTH$DTAND  angle-in-degrees
MTH$GTAND  angle-in-degrees

Each of the above formats accepts one of the floating-point types as input.

JSB Entries

MTH$TAND_R4
MTH$DTAND_R7
MTH$GTAND_R7

Each of the above JSB entries accepts one of the floating-point types as input.

Returns

OpenVMS usage  floating_point
type            F_floating, D_floating, G_floating
access          write only
mechanism       by value

Tangent of the angle specified by angle-in-degrees. MTH$TAND returns an F-floating number. MTH$DTAND returns a D-floating number. MTH$GTAND returns a G-floating number.

Arguments

angle-in-degrees
OpenVMS usage  floating_point
type            F_floating, D_floating, G_floating
access          read only
mechanism       by reference

The input angle (in degrees). The angle-in-degrees argument is the address of a floating-point number which is this angle. For MTH$TAND, angle-in-degrees specifies an F-floating number. For MTH$DTAND, angle-in-degrees specifies a D-floating number. For MTH$GTAND, angle-in-degrees specifies a G-floating number.

Description

When the input argument is expressed in degrees, the tangent function is computed as follows:

1. If |X| < (180/\pi) * 2^{(-2/\epsilon)} and underflow signaling is enabled, underflow is signaled (see the section on MTH$zCOSH for the definition of \epsilon).
2. Otherwise, if |X| < (180/\pi) * 2^{(-f/2)}, then zTAND(X) = (\pi/180) * X. See the description of MTH$zCOSH for the definition of f.
3. Otherwise, call MTH$zSINCO$D to obtain zSIND(X) and zCOSD(X).
   a. Then, if zCOSD(X) = 0, signal overflow
   b. Else, zTAND(X) = zSIND(X)/zCOSD(X)

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HTAND.

**Condition Values Signaled**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$$_ROPRAND</td>
<td>Reserved operand. The MTH$xTAND procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.</td>
</tr>
<tr>
<td>MTH$$_FLOOVEMAT</td>
<td>Floating-point overflow in Math Library.</td>
</tr>
<tr>
<td>MTH$$_FLOUNDMAT</td>
<td>Floating-point underflow in Math Library.</td>
</tr>
</tbody>
</table>
MTH$TANH—Compute the Hyperbolic Tangent

The Compute the Hyperbolic Tangent routine returns the hyperbolic tangent of the input value.

Format

MTH$TANH  floating-point-input-value
MTH$DTANH  floating-point-input-value
MTH$GTANH  floating-point-input-value

Each of the above formats accepts one of the floating-point types as input.

Returns

OpenVMS usage floating_point
  type F_floating, D_floating, G_floating
  access write only
  mechanism by value

The hyperbolic tangent of floating-point-input-value. MTH$TANH returns an F-floating number. MTH$DTANH returns a D-floating number. MTH$GTANH returns a G-floating number. Unlike the other three routines, MTH$HTANH returns the hyperbolic tangent by reference in the h-tanh argument.

Arguments

floating-point-input-value
  OpenVMS usage floating_point
  type F_floating, D_floating, G_floating
  access read only
  mechanism by reference

The input value. The floating-point-input-value argument is the address of a floating-point number that contains this input value. For MTH$TANH, floating-point-input-value specifies an F-floating number. For MTH$DTANH, floating-point-input-value specifies a D-floating number. For MTH$GTANH, floating-point-input-value specifies a G-floating number.

Description

In calculating the hyperbolic tangent of \( x \), the values of \( g \) and \( h \) are:

<table>
<thead>
<tr>
<th>( z )</th>
<th>( g )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>G</td>
<td>26</td>
<td>20</td>
</tr>
</tbody>
</table>
For MTH$TANH, MTH$DTANH, and MTH$GTANH the hyperbolic tangent of \( x \) is then computed as follows:

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Hyperbolic Tangent Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>x</td>
</tr>
<tr>
<td>(2^{-9} &lt;</td>
<td>x</td>
</tr>
<tr>
<td>(0.5 \leq</td>
<td>x</td>
</tr>
<tr>
<td>(1.0 &lt;</td>
<td>x</td>
</tr>
<tr>
<td>(h \leq</td>
<td>x</td>
</tr>
</tbody>
</table>

The routine description for the H-floating point version of this routine is listed alphabetically under MTH$HTANH.

**Condition Value Signaled**

**SS$_{RPRAND}$**

Reserved operand. The MTH$X$TANH procedure encountered a floating-point reserved operand due to incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of 0. Floating-point reserved operands are reserved for future use by Digital.
MTH$UMAX—Compute Unsigned Maximum

The Compute Unsigned Maximum routine computes the unsigned longword maximum of n unsigned longword arguments, where n is greater than or equal to 1.

Format

MTH$UMAX  argument [argument,...]

Returns

OpenVMS usage  longword_unsigned
type  longword (unsigned)
access  write only
mechanism  by value

Maximum value returned by MTH$UMAX.

Arguments

argument
OpenVMS usage  longword_unsigned
type  longword (unsigned)
access  read only
mechanism  by reference

Argument whose maximum MTH$UMAX computes. Each argument argument is an unsigned longword that contains one of these values.

argument
OpenVMS usage  longword_unsigned
type  longword (unsigned)
access  read only
mechanism  by reference

Additional arguments whose maximum MTH$UMAX computes. Each argument argument is an unsigned longword that contains one of these values.

Description

MTH$UMAX is the unsigned version of MTH$JMAX0.

Condition Values Returned

None.
MTH$UMIN—Compute Unsigned Minimum

The Compute Unsigned Minimum routine computes the unsigned longword minimum of n unsigned longword arguments, where n is greater than or equal to 1.

Format

MTH$UMIN argument [argument,...]

Returns

OpenVMS usage longword_unsigned
type longword (unsigned)
access write only
mechanism by value

Minimum value returned by MTH$UMIN.

Arguments

argument
OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by reference

Argument whose minimum MTH$UMIN computes. Each argument argument is an unsigned longword that contains one of these values.

argument
OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by reference

Additional arguments whose minimum MTH$UMIN computes. Each argument argument is an unsigned longword that contains one of these values.

Description

MTH$UMIN is the unsigned version of MTH$JMIN0.

Condition Values Returned

None.
Vector MTH$ Reference Section

The Vector MTH$ Reference Section provides detailed descriptions of two sets of vector routines provided by the OpenVMS RTL Mathematics (MTH$) Facility, BLAS Level 1 and FOLR. The BLAS Level 1 are the Basic Linear Algebraic Subroutines designed by Lawson, Hanson, Kincaid, and Krogh (1978). The FOLR (First Order Linear Recurrence) routines provide a vectorized algorithm for the linear recurrence relation.
Vector MT2 Reference Section
BLAS1$VlxAMAX—Obtain the Index of the First Element of a Vector Having the Largest Absolute Value

The Obtain the Index of the First Element of a Vector Having the Largest Absolute Value routines find the index of the first occurrence of a vector element having the maximum absolute value.

Format

BLAS1$VISAMAX n, x, incx
BLAS1$VIDAMAX n, x, incx
BLAS1$VIGAMAX n, x, incx
BLAS1$VICAMAX n, x, incx
BLAS1$VIZAMAX n, x, incx
BLAS1$VIWAMAX n, x, incx

Use BLAS1$VISAMAX for single-precision real operations. Use BLAS1$VIDAMAX for double-precision real (D-floating) operations and BLAS1$VIGAMAX for double-precision real (G-floating) operations.

Use BLAS1$VICAMAX for single-precision complex operations. Use BLAS1$VIZAMAX for double-precision complex (D-floating) operations and BLAS1$VIWAMAX for double-precision complex (G-floating) operations.

Returns

OpenVMS usage longword_signed
type longword integer (signed)
access write only
mechanism by value

For the real versions of this routine, the function value is the index of the first occurrence of a vector element having the maximum absolute value, as follows:

$$|x_i| = \max \{|x_j| \text{ for } j = 1, 2, \ldots, n\}$$

For the complex versions of this routine, the function value is the index of the first occurrence of a vector element having the largest sum of the absolute values of the real and imaginary parts of the vector elements, as follows:

$$|\Re(x_i)| + |\Im(x_i)| = \max \{|\Re(x_j)| + |\Im(x_j)| \text{ for } j = 1, 2, \ldots, n\}$$

Arguments

n
OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference

Number of elements in vector x. The n argument is the address of a signed longword integer containing the number of elements. If you specify a negative value or 0 for n, 0 is returned.
BLAS1$VixAMAX

\( \mathbf{x} \)

**OpenVMS usage:** floating_point or complex_number

**type:**
- F\_floating, D\_floating, G\_floating real or F\_floating, D\_floating, G\_floating complex

**access:** read only

**mechanism:** by reference, array reference

Array containing the elements to be accessed. All elements of array \( \mathbf{x} \) are accessed only if the increment argument of \( \mathbf{x} \), called \( \text{incx} \), is 1. The \( \mathbf{x} \) argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least

\[ 1 + (n - 1) \times |\text{incx}| \]

where:

- \( n \) = number of vector elements specified in \( \mathbf{n} \)
- \( \text{incx} \) = increment argument for the array \( \mathbf{x} \) specified in \( \text{incx} \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for ( \mathbf{x} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VISAMAX</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VIDAMAX</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VIGAMAX</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VICAMAX</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VIZAMAX</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VIWAMAX</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( \text{imax} \) is 0.

\( \text{incx} \)

**OpenVMS usage:** longword\_signed

**type:** longword\_integer\_signed\_interleaved

**access:** read only

**mechanism:** by reference

Increment argument for the array \( \mathbf{x} \). The \( \text{incx} \) argument is the address of a signed longword integer containing the increment argument. If \( \text{incx} \) is greater than or equal to 0, then \( \mathbf{z} \) is referenced forward in array \( \mathbf{x} \); that is, \( \mathbf{z}_i \) is referenced as:

\[ \mathbf{z}(1 + (i - 1) \times |\text{incx}|) \]

where:

- \( \mathbf{z} \) = array specified in \( \mathbf{x} \)
- \( i \) = element of the vector \( \mathbf{z} \)
- \( \text{incx} \) = increment argument for the array \( \mathbf{x} \) specified in \( \text{incx} \)

If you specify a negative value for \( \text{incx} \), it is interpreted as the absolute value of \( \text{incx} \).
Description

BLAS1$VISAMAX, BLAS1$VIDAMAX, and BLAS1$VIGAMAX find the index, $i$, of the first occurrence of a vector element having the maximum absolute value.
BLAS1$VICAMAX, BLAS1$VIZAMAX, and BLAS1$VIWAMAX find the index, $i$, of the first occurrence of a vector element having the largest sum of the absolute values of the real and imaginary parts of the vector elements.

Vector $x$ contains $n$ elements that are accessed from array $x$ by stepping $\text{incx}$ elements at a time. The vector $x$ is a real or complex single-precision or double-precision (D and G) $n$-element vector. The vector can be a row or a column of a matrix. Both forward and backward indexing are permitted.

BLAS1$VISAMAX, BLAS1$VIDAMAX, and BLAS1$VIGAMAX determine the smallest integer $i$ of the $n$-element vector $x$ such that:

$$|x_i| = \max \{|x_j| \text{ for } j = 1, 2, \ldots, n\}$$

BLAS1$VICAMAX, BLAS1$VIZAMAX, and BLAS1$VIWAMAX determine the smallest integer $i$ of the $n$-element vector $x$ such that:

$$|\text{Re}(x_i)| + |\text{Im}(x_i)| = \max \{|\text{Re}(x_j)| + |\text{Im}(x_j)| \text{ for } j = 1, 2, \ldots, n\}$$

You can use the BLAS1$VixAMAX routines to obtain the pivots in Gaussian elimination.

The public-domain BLAS Level 1IxAMAX routines require a positive value for $\text{incx}$. The Run-Time Library BLAS Level 1 routines interpret a negative value for $\text{incx}$ as the absolute value of $\text{incx}$.

The algorithm does not provide a special case for $\text{incx} = 0$. Therefore, specifying 0 for $\text{incx}$ has the effect of setting $\text{imax}$ equal to 1 using vector operations.

Example

C
C To obtain the index of the element with the maximum
C absolute value.
C
INTEGER IMAX, N, INCX
REAL X(40)
INCX = 2
N = 20
IMAX = BLAS1$VISAMAX(N, X, INCX)
BLAS1$VxASUM—Obtain the Sum of the Absolute Values of the Elements of a Vector

The Obtain the Sum of the Absolute Values of the Elements of a Vector routines determine the sum of the absolute values of the elements of the $n$-element vector $x$.

**Format**

\[
\begin{align*}
\text{BLAS1$VSASUM$} & \quad n, x, incx \\
\text{BLAS1$VDASUM$} & \quad n, x, incx \\
\text{BLAS1$VGASUM$} & \quad n, x, incx \\
\text{BLAS1$VSCASUM$} & \quad n, x, incx \\
\text{BLAS1$VDZASUM$} & \quad n, x, incx \\
\text{BLAS1$VGWASUM$} & \quad n, x, incx
\end{align*}
\]

Use BLAS1$VSASUM$ for single-precision real operations. Use BLAS1$VDASUM$ for double-precision real (D-floating) operations and BLAS1$VGASUM$ for double-precision real (G-floating) operations.

Use BLAS1$VSCASUM$ for single-precision complex operations. Use BLAS1$VDZASUM$ for double-precision complex (D-floating) operations and BLAS1$VGWASUM$ for double-precision complex (G-floating) operations.

**Returns**

OpenVMS usage floating_point

<table>
<thead>
<tr>
<th>type</th>
<th>F_floating, D_floating, or G_floating real</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>write only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by value</td>
</tr>
</tbody>
</table>

The function value, called `sum`, is the sum of the absolute values of the elements of the vector $x$. The data type of the function value is a real number; for the BLAS1$VSCASUM$, BLAS1$VDZASUM$, and BLAS1$VGWASUM$ routines, the data type of the function value is the real data type corresponding to the complex argument data type.

**Arguments**

**n**

OpenVMS usage longword_signed

<table>
<thead>
<tr>
<th>type</th>
<th>longword integer (signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>read only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by reference</td>
</tr>
</tbody>
</table>

Number of elements in vector $x$ to be added. The $n$ argument is the address of a signed longword integer containing the number of elements.

**x**

OpenVMS usage floating_point or complex_number

<table>
<thead>
<tr>
<th>type</th>
<th>F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>read only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by reference, array reference</td>
</tr>
</tbody>
</table>
Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( \text{incx} \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least
\[
1 + (n - 1) \cdot |\text{incx}|
\]
where:
- \( n \) = number of vector elements specified in \( n \)
- \( \text{incx} \) = increment argument for the array \( x \) specified in \( \text{incx} \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for ( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSASUM</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDASUM</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGASUM</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VSCASUM</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VDZASUM</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VGWASUM</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( \text{sum} \) is 0.0.

**incx**

OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference

Increment argument for the array \( x \). The \( \text{incx} \) argument is the address of a signed longword integer containing the increment argument. If \( \text{incx} \) is greater than or equal to 0, then \( x \) is referenced forward in array \( x \); that is, \( z_i \) is referenced in:
\[
x(1 + (i - 1) \cdot \text{incx})
\]
where:
- \( x \) = array specified in \( x \)
- \( i \) = element of the vector \( x \)
- \( \text{incx} \) = increment argument for the array \( x \) specified in \( \text{incx} \)

If you specify a negative value for \( \text{incx} \), it is interpreted as the absolute value of \( \text{incx} \).

**Description**

BLAS1$VSASUM, BLAS1$VDASUM, and BLAS1$VGASUM obtain the sum of the absolute values of the elements of the \( n \)-element vector \( x \). BLAS1$VSCASUM, BLAS1$VDZASUM, and BLAS1$VGWASUM obtain the sum of the absolute values of the real and imaginary parts of the elements of the \( n \)-element vector \( x \).

Vector \( x \) contains \( n \) elements that are accessed from array \( x \) by stepping \( \text{incx} \) elements at a time. The vector \( x \) is a real or complex single-precision or double-precision (D and G) \( n \)-element vector. The vector can be a row or a column of a matrix. Both forward and backward indexing are permitted.
BLAS1$VxASUM

BLAS1$VSASUM, BLAS1$VDASUM, and BLAS1$VGASUM compute the sum of the absolute values of the elements of \( x \), which is expressed as follows:

\[
\sum_{i=1}^{n} |x_i| = |x_1| + |x_2| + \ldots + |x_n|
\]

BLAS1$VSCASUM, BLAS1$VDZASUM, and BLAS1$VGWASUM compute the sum of the absolute values of the real and imaginary parts of the elements of \( x \), which is expressed as follows:

\[
\sum_{i=1}^{n} (|a_i| + |b_i|) = (|a_1| + |b_1|) + (|a_2| + |b_2|) + \ldots + (|a_n| + |b_n|)
\]

where \( |x_i| = (a_i, b_i) \)

and \( |a_i| + |b_i| = |\text{real}| + |\text{imaginary}| \)

The public-domain BLAS Level 1 xASUM routines require a positive value for \( \text{incx} \). The Run-Time Library BLAS Level 1 routines interpret a negative value for \( \text{incx} \) as the absolute value of \( \text{incx} \).

The algorithm does not provide a special case for \( \text{incx} = 0 \). Therefore, specifying \( 0 \) for \( \text{incx} \) has the effect of computing \( n \times |x_1| \) using vector operations.

Rounding in the summation occurs in a different order than in a sequential evaluation of the sum, so the final result may differ from the result of a sequential evaluation.

**Example**

```c
C To obtain the sum of the absolute values of the
C elements of vector x:
C
INTEGER N,INCX
REAL X(20),SUM
INCX = 1
N = 20
SUM = BLAS1$VSASUM(N,X,INCX)
```

MTH-154
BLAS1$VxAXPY—Multiply a Vector by a Scalar and Add a Vector

The Multiply a Vector by a Scalar and Add a Vector routines compute \( ax + y \), where \( a \) is a scalar number and \( x \) and \( y \) are \( n \)-element vectors.

Format

- \text{BLAS1$VSAXPY} \ n, a, x, \text{incx}, y, \text{incy}
- \text{BLAS1$VDAXPY} \ n, a, x, \text{incx}, y, \text{incy}
- \text{BLAS1$VGAXPY} \ n, a, x, \text{incx}, y, \text{incy}
- \text{BLAS1$VCAXPY} \ n, a, x, \text{incx}, y, \text{incy}
- \text{BLAS1$VZAXPY} \ n, a, x, \text{incx}, y, \text{incy}
- \text{BLAS1$VWAXPY} \ n, a, x, \text{incx}, y, \text{incy}

Use \text{BLAS1$VSAXPY} for single-precision real operations. Use \text{BLAS1$VDAXPY} for double-precision real (D-floating) operations and \text{BLAS1$VGAXPY} for double-precision real (G-floating) operations.

Use \text{BLAS1$VCAXPY} for single-precision complex operations. Use \text{BLAS1$VZAXPY} for double-precision complex (D-floating) operations and \text{BLAS1$VWAXPY} for double-precision complex (G-floating) operations.

Returns

None.

Arguments

- \text{n}
  - OpenVMS usage: longword_signed
  - type: longword integer (signed)
  - access: read only
  - mechanism: by reference
  - Number of elements in vectors \( x \) and \( y \). The \( n \) argument is the address of a signed longword integer containing the number of elements. If \( n \) is less than or equal to 0, then \( y \) is unchanged.

- \text{a}
  - OpenVMS usage: floating_point or complex_number
  - type: \text{F\_floating}, \text{D\_floating}, \text{G\_floating} real or \text{F\_floating}, \text{D\_floating}, \text{G\_floating} complex
  - access: read only
  - mechanism: by reference, array reference
  - Scalar multiplier for the array \( x \). The \( a \) argument is the address of a floating-point or floating-point complex number that is this multiplier. If \( a \) equals 0, then \( y \) is unchanged. If \( a \) shares a memory location with any element of the vector \( y \), results are unpredictable. Specify the same data type for arguments \( a, x, \) and \( y \).
**BLAS1$VxAXPY**

**x**

OpenVMS usage floating_point or complex_number

Type F-floating, D-floating, G-floating real or F-floating, D-floating, G-floating complex

Access read only

Mechanism by reference, array reference

Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( \text{incx} \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. The length of this array is at least \( 1 + (n - 1) \times |\text{incx}| \)

where:

\( n \) = number of vector elements specified in \( n \)

\( \text{incx} \) = increment argument for the array \( x \) specified in \( \text{incx} \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for ( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSAXPY</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDAXPY</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGAXPY</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCAXPY</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZAXPY</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWAXPY</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If any element of \( x \) shares a memory location with an element of \( y \), the results are unpredictable.

**incx**

OpenVMS usage longword_signed

Type longword integer (signed)

Access read only

Mechanism by reference

Increment argument for the array \( x \). The \( \text{incx} \) argument is the address of a signed longword integer containing the increment argument. If \( \text{incx} \) is greater than or equal to 0, then \( x \) is referenced forward in array \( x \); that is, \( x_i \) is referenced in:

\[ x(1 + (i - 1) \times \text{incx}) \]

where:

\( x \) = array specified in \( x \)

\( i \) = element of the vector \( x \)

\( \text{incx} \) = increment argument for the array \( x \) specified in \( \text{incx} \)

If \( \text{incx} \) is less than 0, then \( x \) is referenced backward in array \( x \); that is, \( x_i \) is referenced in:

\[ x(1 + (n - i) \times |\text{incx}|) \]
where:

\[ x = \text{array specified in } x \]
\[ n = \text{number of vector elements specified in } n \]
\[ i = \text{element of the vector } x \]
\[ incx = \text{increment argument for the array } x \text{ specified in } incx \]

**y**

OpenVMS usage: floating_point or complex_number

- F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex
- modify
- by reference, array reference

On entry, array containing the elements to be accessed. All elements of array y are accessed only if the increment argument of y, called incy, is 1. The y argument is the address of a floating-point or floating-point complex number that is this array. The length of this array is at least

\[ 1 + (n - 1) \times |incy| \]

where:

\[ n = \text{number of vector elements specified in } n \]
\[ incy = \text{increment argument for the array } y \text{ specified in } incy \]

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for y</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSAXPY</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDAXPY</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGAXPY</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCAXPY</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZAXPY</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWAXPY</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then y is unchanged. If any element of x shares a memory location with an element of y, the results are unpredictable.

On exit, y contains an array of length at least

\[ 1 + (n - 1) \times |incy| \]

where:

\[ n = \text{number of vector elements specified in } n \]
\[ incy = \text{increment argument for the array } y \text{ specified in } incy \]

After the call to BLAS1$VxAXPY, \( y_i \) is set equal to

\[ y_i + a \times x_i. \]
where:

\[ y = \text{the vector } y \]
\[ i = \text{element of the vector } x \text{ or } y \]
\[ a = \text{scalar multiplier for the vector } x \text{ specified in } a \]
\[ x = \text{the vector } x \]

**incy**

OpenVMS usage longword\_signed
type longword integer (signed)
access read only
mechanism by reference

Increment argument for the array \( y \). The \( incy \) argument is the address of a signed longword integer containing the increment argument. If \( incy \) is greater than or equal to 0, then \( y \) is referenced forward in array \( y \); that is, \( y_i \) is referenced in:

\[ y(1 + (i - 1) \times incy) \]

where:

\( y = \text{array specified in } y \)
\( i = \text{element of the vector } y \)
\( incy = \text{increment argument for the array } y \text{ specified in } incy \)

If \( incy \) is less than 0, then \( y \) is referenced backward in array \( y \); that is, \( y_i \) is referenced in:

\[ y(1 + (n - i) \times |incy|) \]

where:

\( y = \text{array specified in } y \)
\( n = \text{number of vector elements specified in } n \)
\( i = \text{element of the vector } y \)
\( incy = \text{increment argument for the array } y \text{ specified in } incy \)

**Description**

BLAS1$VxAXPY multiplies a vector \( x \) by a scalar, adds to a vector \( y \), and stores the result in the vector \( y \). This is expressed as follows:

\[ y \leftarrow ax + y \]

where \( a \) is a scalar number and \( x \) and \( y \) are real or complex single-precision or double-precision (D and G) \( n \)-element vectors. The vectors can be rows or columns of a matrix. Both forward and backward indexing are permitted. Vectors \( x \) and \( y \) contain \( n \) elements that are accessed from arrays \( x \) and \( y \) by stepping \( incx \) and \( incy \) elements at a time.

The routine name determines the data type you should specify for arguments \( a \), \( x \), and \( y \). Specify the same data type for each of these arguments.

The algorithm does not provide a special case for \( incx = 0 \). Therefore, specifying 0 for \( incx \) has the effect of adding the constant \( a \times x_1 \) to all elements of the vector \( y \) using vector operations.
Example

To compute $y = y + 2.0 \times x$ using SAXPY:

```c
C
C To compute y=y+2.0*x using SAXPY:
C
INTEGER N, INCX, INCY
REAL X(20), Y(20), A
INCX = 1
INCY = 1
A = 2.0
N = 20
CALL BLAS1$VSAXPY(N, A, X, INCX, Y, INCY)
```
BLAS1$VxCOPY

BLAS1$VxCOPY—Copy a Vector

The Copy a Vector routines copy \( n \) elements of the vector \( x \) to the vector \( y \).

Format

BLAS1$VSCOPY \( n, x, incx, y, incy \)
BLAS1$VDCOPY \( n, x, incx, y, incy \)
BLAS1$VCCOPY \( n, x, incx, y, incy \)
BLAS1$VZCOPY \( n, x, incx, y, incy \)

Use BLAS1$VSCOPY for single-precision real operations and BLAS1$VDCOPY for double-precision real (D or G) operations.

Use BLAS1$VCCOPY for single-precision complex operations and BLAS1$VZCOPY for double-precision complex (D or G) operations.

Returns

None.

Arguments

\( n \)

OpenVMS usage floating_point or complex_number

type longword integer (signed)

access read only

mechanism by reference

Number of elements in vector \( x \) to be copied. The \( n \) argument is the address of a signed longword integer containing the number of elements in vector \( x \). If \( n \) is less than or equal to 0, then \( y \) is unchanged.

\( x \)

OpenVMS usage floating_point or complex_number

type F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex

access read only

mechanism by reference, array reference

Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( incx \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least

\[ 1 + (n - 1) \times |incx| \]

where:

\( n \) = number of vector elements specified in \( n \)

\( incx \) = increment argument for the array \( x \) specified in \( incx \)
Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for x</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSCOPY</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDCOPY</td>
<td>D-floating or G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCCOPY</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZCOPY</td>
<td>D-floating or G-floating complex</td>
</tr>
</tbody>
</table>

**incx**

OpenVMS usage: longword_signed

Type: longword integer (signed)

Access: read only

Mechanism: by reference

Increment argument for the array \( x \). The `incx` argument is the address of a signed longword integer containing the increment argument. If `incx` is greater than or equal to 0, then \( x \) is referenced forward in array \( x \); that is, \( x_i \) is referenced in:

\[
x(1 + (i - 1) \times incx)
\]

where:

- \( x \) = array specified in \( x \)
- \( i \) = element of the vector \( x \)
- \( incx \) = increment argument for the array \( x \) specified in \( incx \)

If `incx` is less than 0, then \( x \) is referenced backward in array \( x \); that is, \( x_i \) is referenced in:

\[
x(1 + (n - i) \times |incx|)
\]

where:

- \( x \) = array specified in \( x \)
- \( n \) = number of vector elements specified in \( n \)
- \( i \) = element of the vector \( x \)
- \( incx \) = increment argument for the array \( x \) specified in \( incx \)

**y**

OpenVMS usage: floating_point or complex_number

Type: floating_point, D_floating, G_floating real or F_floating,
D_floating, G_floating complex

Access: write only

Mechanism: by reference, array reference

Array that receives the copied elements. All elements of array \( y \) receive the copied elements only if the increment argument of \( y \), called \( incy \), is 1. The `yncy` argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least

\[
1 + (n - 1) \times |incy|
\]
where:

\[ n \quad = \quad \text{number of vector elements specified in } n \]

\[ incy \quad = \quad \text{increment argument for the array } y \text{ specified in } incy \]

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for y</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSCOPY</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDCOPY</td>
<td>D-floating or G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCCOPY</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZCOPY</td>
<td>D-floating or G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( y \) is unchanged. If \( incx \) is equal to 0, then each \( y_i \) is set to \( x_1 \). If \( incy \) is equal to 0, then \( y_i \) is set to the last referenced element of \( x \). If any element of \( x \) shares a memory location with an element of \( y \), the results are unpredictable. (See the Description section for a special case that does not cause unpredictable results when the same memory location is shared by input and output.)

**incy**

OpenVMS usage: longword_signed  
Type: longword integer (signed)  
Access: read only  
Mechanism: by reference

Increment argument for the array \( y \). The \( incy \) argument is the address of a signed longword integer containing the increment argument. If \( incy \) is greater than or equal to 0, then \( y \) is referenced forward in array \( y \); that is, \( y_i \) is referenced in:

\[ y(1 + (i - 1) \times incy) \]

where:

\[ y \quad = \quad \text{array specified in } y \]

\[ i \quad = \quad \text{element of the vector } y \]

If \( incy \) is less than 0, then \( y \) is referenced backward in array \( y \); that is, \( y_i \) is referenced in:

\[ y(1 + (n - i) \times \text{|incy|}) \]

where:

\[ y \quad = \quad \text{array specified in } y \]

\[ n \quad = \quad \text{number of vector elements specified in } n \]

\[ i \quad = \quad \text{element of the vector } y \]

\[ incy \quad = \quad \text{increment argument for the array } y \text{ specified in } incy \]
**Description**

BLAS1$VSCOPY, BLAS1$VDCOPY, BLAS1$VCCOPY, and BLAS1$VZCOPY copy \( n \) elements of the vector \( x \) to the vector \( y \). Vector \( x \) contains \( n \) elements that are accessed from array \( x \) by stepping \( \text{incx} \) elements at a time. Both \( x \) and \( y \) are real or complex single-precision or double-precision (D and G) \( n \)-element vectors. The vectors can be rows or columns of a matrix. Both forward and backward indexing are permitted.

If you specify 0 for \( \text{incx} \), BLAS1$VxCOPY initializes all elements of \( y \) to a constant.

If you specify \(-\text{incx}\) for \( \text{incy}\), the vector \( z \) is stored in reverse order in \( y \). In this case, the call format is as follows:

```fortran
CALL BLAS1$VxCOPY (N, X, INCX, Y, -INCX)
```

It is possible to move the contents of a vector up or down within itself and not cause unpredictable results even though the same memory location is shared between input and output. To do this when \( i \) is greater than \( j \), call the routine BLAS1$VxCOPY with \( \text{incx} = \text{incy} > 0 \) as follows:

```fortran
CALL BLAS1$VxCOPY (N, X(I), INCX, X(J), INCX)
```

The preceding call to BLAS1$VxCOPY moves:

\[
x(i), x(i + 1 \ast \text{incx}), ... x(i + (n - 1) \ast \text{incx})
\]

to

\[
x(j), x(j + 1 \ast \text{incx}), ... x(j + (n - 1) \ast \text{incx})
\]

If \( i \) is less than \( j \), specify a negative value for \( \text{incx} \) and \( \text{incy} \) in the call to BLAS1$VxCOPY, as follows. The parts that do not overlap are unchanged.

```fortran
CALL BLAS1$VxCOPY (N, X(I), -INCX, X(J), -INCX)
```

---

**Note**

BLAS1$VxCOPY does not perform floating operations on the input data. Therefore, floating reserved operands are not detected by BLAS1$VxCOPY.
Example

To copy a vector $x$ to a vector $y$ using BLAS1$VSCOPY$:

```c
INTEGER N, INCX, INCY
REAL X(20), Y(20)
INCX = 1
INCY = 1
N = 20
CALL BLAS1$VSCOPY(N, X, INCX, Y, INCY)
```

To move the contents of $X(1), X(3), X(5), \ldots, X(2N-1)$
to $X(3), X(5), \ldots, X(2N+1)$ and leave $x$ unchanged:

```c
CALL BLAS1$VSCOPY(N, X, -2, X(3), -2)
```

To move the contents of $X(2), X(3), \ldots, X(100)$ to
$X(1), X(2), \ldots, X(99)$ and leave $x(100)$ unchanged:

```c
CALL BLAS1$VSCOPY(99, X(2), 1, X, 1)
```

To move the contents of $X(1), X(2), X(3), \ldots, X(N)$ to
$Y(N), Y(N-1), \ldots, Y$

```c
CALL BLAS1$VSCOPY(N, X, 1, Y, -1)
```
The Obtain the Inner Product of Two Vectors routines return the dot product of two n-element vectors, x and y.

**Format**

- `BLAS1$VSDOT n, x, incx, y, incy`
- `BLAS1$VDDOT n, x, incx, y, incy`
- `BLAS1$VGDOT n, x, incx, y, incy`
- `BLAS1$VCDOTU n, x, incx, y, incy`
- `BLAS1$VCDOTC n, x, incx, y, incy`
- `BLAS1$VZDOTU n, x, incx, y, incy`
- `BLAS1$VZDOTC n, x, incx, y, incy`
- `BLAS1$VWDOTU n, x, incx, y, incy`
- `BLAS1$VWDOTC n, x, incx, y, incy`

Use BLAS1$VSDOT to obtain the inner product of two single-precision real vectors.

Use BLAS1$VDDOT to obtain the inner product of two double-precision (D-floating) real vectors. Use BLAS1$VGDOT to obtain the inner product of two double-precision (G-floating) real vectors.

Use BLAS1$VCDOTU to obtain the inner product of two single-precision complex vectors (unconjugated).

Use BLAS1$VCDOTC to obtain the inner product of two single-precision complex vectors (conjugated).

Use BLAS1$VZDOTU to obtain the inner product of two double-precision (D-floating) complex vectors (unconjugated). Use BLAS1$VWDOTU to obtain the inner product of two double-precision (G-floating) complex vectors (unconjugated).

Use BLAS1$VZDOTC to obtain the inner product of two double-precision (D-floating) complex vectors (conjugated). Use BLAS1$VWDOTC to obtain the inner product of two double-precision (G-floating) complex vectors (conjugated).

**Returns**

OpenVMS usage floating_point or complex_number
type F_floating, D_floating, G_floating real or F_floating,
D_floating, G_floating complex
access write only
mechanism by value

The function value, called `dotpr`, is the dot product of two n-element vectors, x and y. Specify the same data type for `dotpr` and the argument x.
Arguments

\( \text{n} \)

OpenVMS usage: longword_signed  
Type: longword integer (signed)  
Access: read only  
Mechanism: by reference

Number of elements in vector \( x \). The \( n \) argument is the address of a signed longword integer containing the number of elements. If you specify a value for \( n \) that is less than or equal to 0, then the value of \( \text{dotpr} \) is 0.0.

\( x \)

OpenVMS usage: floating_point or complex_number  
Type: F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex  
Access: read only  
Mechanism: by reference, array reference

Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( \text{incx} \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least

\[
1 + (n - 1) \times |\text{incx}| \]

where:

\( n \) = number of vector elements specified in \( n \)  
\( \text{incx} \) = increment argument for the array \( x \) specified in \( \text{incx} \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for ( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSDOT</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDDOT</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGDOT</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCDOTU and</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VCDOTC</td>
<td></td>
</tr>
<tr>
<td>BLAS1$VZDOTU and</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZDOTC</td>
<td></td>
</tr>
<tr>
<td>BLAS1$VWDOTU and</td>
<td>G-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWDOTC</td>
<td></td>
</tr>
</tbody>
</table>

\( \text{incx} \)

OpenVMS usage: longword_signed  
Type: longword integer (signed)  
Access: read only  
Mechanism: by reference

Increment argument for the array \( x \). The \( \text{incx} \) argument is the address of a signed longword integer containing the increment argument. If \( \text{incx} \) is greater than 0, then \( x \) is referenced forward in array \( x \); that is, \( z_i \) is referenced in:

\[
z(1 + (i - 1) \times \text{incx})
\]
where:

\( x \) = array specified in \( x \)
\( i \) = element of the vector \( x \)
\( incx \) = increment argument for the array \( x \) specified in \( incx \)

If \( incx \) is less than 0, then \( z \) is referenced backward in array \( x \); that is, \( z_i \) is referenced in:

\[ z(1 + (n - i) \cdot |incx|) \]

where:

\( x \) = array specified in \( x \)
\( n \) = number of vector elements specified in \( n \)
\( i \) = element of the vector \( z \)
\( incx \) = increment argument for the array \( x \) specified in \( incx \)

\( y \)

OpenVMS usage floating_point or complex_number

**type**

F_floating, D_floating, G_floating real or F_floating,
D_floating, G_floating complex

**access**

read only

**mechanism**

by reference, array reference

Array containing the elements to be accessed. All elements of array \( y \) are accessed only if the increment argument of \( y \), called \( incy \), is 1. The \( y \) argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least

\[ 1 + (n - 1) \cdot |incy| \]

where:

\( n \) = number of vector elements specified in \( n \)
\( incy \) = increment argument for the array \( y \) specified in \( incy \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for ( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSDOT</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDDOT</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGDOT</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCODOTU and</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VCODOTC</td>
<td>G-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZDOTU and</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZDOTC</td>
<td>G-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWDOTU and</td>
<td></td>
</tr>
<tr>
<td>BLAS1$VWDOTC</td>
<td></td>
</tr>
</tbody>
</table>
BLAS1$VxDOTx

inci
OpenVMS usage  longword_signed
type       longword integer (signed)
access     read only
mechanism  by reference

Increment argument for the array y. The incy argument is the address of a signed longword integer containing the increment argument. If incy is greater than or equal to 0, then y is referenced forward in array y; that is, yi is referenced in:

\[ y(1 + (i - 1) \times \text{incy}) \]

where:

\[ y = \text{array specified in } y \]
\[ i = \text{element of the vector } y \]
\[ \text{incy} = \text{increment argument for the array } y \text{ specified in } \text{incy} \]

If incy is less than 0, then y is referenced backward in array y; that is, yi is referenced in:

\[ y(1 + (n - i) \times |\text{incy}|) \]

where:

\[ y = \text{array specified in } y \]
\[ n = \text{number of vector elements specified in } n \]
\[ i = \text{element of the vector } y \]
\[ \text{incy} = \text{increment argument for the array } y \text{ specified in } \text{incy} \]

Description

The unconjugated versions of this routine, BLAS1$VSDOT, BLAS1$VDDOT, BLAS1$VGDOT, BLAS1$VCMDOT, BLAS1$VZDOTU, and BLAS1$VWDOTU return the dot product of two n-element vectors, x and y, expressed as follows:

\[ x \cdot y = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n \]

The conjugated versions of this routine, BLAS1$VCDOTC, BLAS1$VZDOTC, and BLAS1$VWDOTC return the dot product of the conjugate of the first n-element vector with a second n-element vector, as follows:

\[ \bar{x} \cdot y = \bar{x}_1 y_1 + \bar{x}_2 y_2 + \ldots + \bar{x}_n y_n \]

Vectors x and y contain n elements that are accessed from arrays x and y by stepping incx and incy elements at a time. The vectors x and y can be rows or columns of a matrix. Both forward and backward indexing are permitted.

The routine name determines the data type you should specify for arguments x and y. Specify the same data type for these arguments.

Rounding in BLAS1$VxDOTx occurs in a different order than in a sequential evaluation of the dot product. The final result may differ from the result of a sequential evaluation.
Example

C To compute the dot product of two vectors, x and y, and return the result in DOTPR:
C
INTEGER INCX, INCY
REAL X(20), Y(20), DOTPR
IN CX = 1
IN CY = 1
N = 20
DOTPR = BLAS1$VS DOT(N, X, INCX, Y, INCY)
BLAS1$VxNRM2

BLAS1$VxNRM2—Obtain the Euclidean Norm of a Vector

The Obtain the Euclidean Norm of a Vector routines obtain the Euclidean norm of an n-element vector x, expressed as follows:

\[ \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \]

**Format**

BLAS1$VSNRM2 n,x,incx
BLAS1$VDNRM2 n,x,incx
BLAS1$VGNRM2 n,x,incx
BLAS1$VSCNRM2 n,x,incx
BLAS1$VDZNRM2 n,x,incx
BLAS1$VGWNRM2 n,x,incx

Use BLAS1$VSNRM2 for single-precision real operations. Use BLAS1$VDNRM2 for double-precision real (D-floating) operations and BLAS1$VGNRM2 for double-precision real (G-floating) operations.

Use BLAS1$VSCNRM2 for single-precision complex operations. Use BLAS1$VDZNRM2 for double-precision complex (D-floating) operations and BLAS1$VGWNRM2 for double-precision complex (G-floating) operations.

**Returns**

OpenVMS usage floating_point

type F_floating, D_floating, or G_floating real

access write only

mechanism by value

The function value, called e_norm, is the Euclidean norm of the vector x. The data type of the function value is a real number; for the BLAS1$VSCNRM2, BLAS1$VDZNRM2, and BLAS1$VGWNRM2 routines, the data type of the function value is the real data type corresponding to the complex argument data type.

**Arguments**

n

OpenVMS usage longword_signed

type longword integer (signed)

access read only

mechanism by reference

Number of elements in vector x to be processed. The n argument is the address of a signed longword integer containing the number of elements.

x

OpenVMS usage floating_point or complex_number

type F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex

access read only

mechanism by reference, array reference
Array containing the elements to be accessed. All elements of array $x$ are accessed only if the increment argument of $x$, called $\text{incx}$, is 1. The $x$ argument is the address of a floating-point or floating-point complex number that is this array. This argument is an array of length at least $1 + (n - 1) \cdot \lvert \text{incx} \rvert$

where:

$n$ = number of vector elements specified in $n$
$\text{incx}$ = increment argument for the array $x$ specified in $\text{incx}$

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSNRM2$</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDNRM2$</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGNRM2$</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VSCNRM2$</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VDZNRM2$</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VGWNRM2$</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If $n$ is less than or equal to 0, then $\text{e.norm}$ is 0.0.

$\text{incx}$

OpenVMS usage longword_signed

type longword integer (signed)

access read only

mechanism by reference

Increment argument for the array $x$. The $\text{incx}$ argument is the address of a signed longword integer containing the increment argument. If $\text{incx}$ is greater than or equal to 0, then $z$ is referenced forward in array $x$; that is, $z_i$ is referenced in:

$z(1 + (i - 1) \cdot \text{incx})$

where:

$x$ = array specified in $x$

$i$ = element of the vector $x$

$\text{incx}$ = increment argument for the array $x$ specified in $\text{incx}$

If you specify a negative value for $\text{incx}$, it is interpreted as the absolute value of $\text{incx}$.

**Description**

BLAS1$VxNRM2$ obtains the Euclidean norm of an $n$-element vector $z$, expressed as follows:

$$\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

Vector $z$ contains $n$ elements that are accessed from array $x$ by stepping $\text{incx}$ elements at a time. The vector $z$ is a real or complex single-precision or double-precision (D and G) $n$-element vector. The vector can be a row or a column of a matrix. Both forward and backward indexing are permitted.
The public-domain BLAS Level 1 xNRM2 routines require a positive value for \( \text{incx} \). The Run-Time Library BLAS Level 1 routines interpret a negative value for \( \text{incx} \) as the absolute value of \( \text{incx} \).

The algorithm does not provide a special case for \( \text{incx} = 0 \). Therefore, specifying 0 for \( \text{incx} \) has the effect of using vector operations to set \( e_{\text{norm}} \) as follows:

\[
e_{\text{norm}} = n^{0.5} \cdot |x_1|
\]

For BLAS1$VDNRM2$, BLAS1$VGNRM2$, BLAS1$VDZNRM2$, and BLAS1$VGWNRM2$ (the double-precision routines), the elements of the vector \( x \) are scaled to avoid intermediate overflow or underflow. BLAS1$VSNRM2$ and BLAS1$VSCNRM2$ (the single-precision routines) use a backup data type to avoid intermediate overflow or underflow.

Rounding in BLAS1$VxNRM2$ occurs in a different order than in a sequential evaluation of the Euclidean norm. The final result may differ from the result of a sequential evaluation.

Example

```c
C To obtain the Euclidean norm of the vector x:
C
INTEGER INCX, N
REAL X(20), E_NORM
INCX = 1
N = 20
E_NORM = BLAS1$VSNRM2(N, X, INCX)
```
The Apply a Givens Plane Rotation routines apply a Givens plane rotation to a pair of n-element vectors x and y.

Format

BLAS1$VSROT n ,x ,incx ,y ,incy ,c ,s
BLAS1$VDROT n ,x ,incx ,y ,incy ,c ,s
BLAS1$VGROT n ,x ,incx ,y ,incy ,c ,s
BLAS1$VCSROT n ,x ,incx ,y ,incy ,c ,s
BLAS1$VZDROT n ,x ,incx ,y ,incy ,c ,s
BLAS1$VWGROT n ,x ,incx ,y ,incy ,c ,s

Use BLAS1$VSROT for single-precision real operations. Use BLAS1$VDROT for double-precision real (D-floating) operations and BLAS1$VGROT for double-precision real (G-floating) operations.

Use BLAS1$VCSROT for single-precision complex operations. Use BLAS1$VZDROT for double-precision complex (D-floating) operations and BLAS1$VWGROT for double-precision complex (G-floating) operations.

BLAS1$VCSROT, BLAS1$VZDROT, and BLAS1$VWGROT are real rotations applied to a complex vector.

Returns

None.

Arguments

n
OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference

Number of elements in vectors x and y to be rotated. The n argument is the address of a signed longword integer containing the number of elements to be rotated. If n is less than or equal to 0, then x and y are unchanged.

x
OpenVMS usage floating_point or complex_number
type F.floating, D.floating, G.floating real or F.floating,
D.floating, G.floating complex
access modify
mechanism by reference, array reference

Array containing the elements to be accessed. All elements of array x are accessed only if the increment argument of x, called incx, is 1. The x argument is the address of a floating-point or floating-point complex number that is this array. On entry, this argument is an array of length at least

\[ 1 + (n - 1) \times |incx| \]
where:

\[ n = \text{number of vector elements specified in } n \]

\[ \text{incx} = \text{increment argument for the array } x \text{ specified in incx} \]

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for x</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROT</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDROT</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGROT</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCSROT</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZDROT</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWGROT</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( x \) and \( y \) are unchanged. If \( c \) equals 1.0 and \( s \) equals 0, then \( x \) and \( y \) are unchanged. If any element of \( z \) shares a memory location with an element of \( y \), then the results are unpredictable.

On exit, \( x \) contains the rotated vector \( z \), as follows:

\[ z_i \leftarrow c \cdot x_i + s \cdot y_i \]

where:

\[ x = \text{array } x \text{ specified in } x \]
\[ y = \text{array } y \text{ specified in } y \]
\[ i = i = 1, 2, \ldots, n \]
\[ c = \text{rotation element generated by the BLAS1$VxROTG routines} \]
\[ s = \text{rotation element generated by the BLAS1$VxROTG routines} \]

**incx**

OpenVMS usage: longword

type: longword integer (signed)

access: read only

mechanism: by reference

Increment argument for the array \( x \). The \( \text{incx} \) argument is the address of a signed longword integer containing the increment argument. If \( \text{incx} \) is greater than or equal to 0, then \( z \) is referenced forward in array \( x \); that is, \( z_i \) is referenced in:

\[ z(1 + (i - 1) \cdot \text{incx}) \]

where:

\[ x = \text{array specified in } x \]
\[ i = \text{element of the vector } z \]
\[ \text{incx} = \text{increment argument for the array } x \text{ specified in incx} \]

If \( \text{incx} \) is less than 0, then \( z \) is referenced backward in array \( x \); that is, \( z_i \) is referenced in:

\[ z(1 + (n - i) \cdot \text{|incx|}) \]
where:

\[ x = \text{array specified in } x \]

\[ n = \text{number of vector elements specified in } n \]

\[ i = \text{element of the vector } x \]

\[ incx = \text{increment argument for the array } x \text{ specified in } incx \]

\[ y \]

OpenVMS usage floating_point or complex_number

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for y</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROT</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDROT</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGROT</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCSROT</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZDROT</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWGROT</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( x \) and \( y \) are unchanged. If \( c \) equals 1.0 and \( s \) equals 0, then \( x \) and \( y \) are unchanged. If any element of \( x \) shares a memory location with an element of \( y \), then the results are unpredictable.

On exit, \( y \) contains the rotated vector \( y \), as follows:

\[ y_i \leftarrow s \cdot x_i + c \cdot y_i \]

where:

\[ x = \text{array } x \text{ specified in } x \]

\[ y = \text{array } y \text{ specified in } y \]

\[ i = i = 1, 2, \ldots, n \]

\[ c = \text{real rotation element (can be generated by the BLAS1$VxROTG routines)} \]

\[ s = \text{complex rotation element (can be generated by the BLAS1$VxROTG routines)} \]
**BLAS1$VxROT**

**incy**

OpenVMS usage: longword_signed  
Type: longword integer (signed)  
Access: read only  
Mechanism: by reference

Increment argument for the array $y$. The **incy** argument is the address of a signed longword integer containing the increment argument. If **incy** is greater than or equal to 0, then $y$ is referenced forward in array $y$; that is, $y_i$ is referenced in:

$$y(1 + (i - 1) \times incy)$$

where:

- $y$ = array specified in $y$
- $i$ = element of the vector $y$
- incy = increment argument for the array $y$ specified in **incy**

If **incy** is less than 0, then $y$ is referenced backward in array $y$; that is, $y_i$ is referenced in:

$$y(1 + (n - i) \times |incy|)$$

where:

- $y$ = array specified in $y$
- $n$ = number of vector elements specified in **n**
- $i$ = element of the vector $y$
- incy = increment argument for the array $y$ specified in **incy**

**c**

OpenVMS usage: floating_point  
Type: F_floating, D_floating, or G_floating real  
Access: read only  
Mechanism: by reference

First rotation element, which can be interpreted as the cosine of the angle of rotation. The **c** argument is the address of a floating-point or floating-point complex number that is this vector element. The **c** argument is the first rotation element generated by the BLAS1$VxROTG routines.

Specify the data type (which is always real) as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for <strong>c</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROT and BLAS1$VCSROT</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDROT and BLAS1$VZDROT</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGROT and BLAS1$VWGROT</td>
<td>G-floating real</td>
</tr>
</tbody>
</table>
**Description**

BLAS1$VSROT, BLAS1$VDROT, and BLAS1$VGROT apply a real Givens plane rotation to a pair of real vectors. BLAS1$VCSROT, BLAS1$VZDROT, and BLAS1$VWGROT apply a real Givens plane rotation to a pair of complex vectors. The vectors \(x\) and \(y\) are real or complex single-precision or double-precision (D and G) vectors. The vectors can be rows or columns of a matrix. Both forward and backward indexing are permitted. The routine name determines the data type you should specify for arguments \(x\) and \(y\). Specify the same data type for each of these arguments.

The Givens plane rotation is applied to \(n\) elements, where the elements to be rotated are contained in vectors \(x\) and \(y\) (\(i\) equals 1,2,\(\ldots\),\(n\)). These elements are accessed from arrays \(x\) and \(y\) by stepping \(\text{incx}\) and \(\text{incy}\) elements at a time. The cosine and sine of the angle of rotation are \(c\) and \(s\), respectively. The arguments \(c\) and \(s\) are usually generated by the BLAS Level 1 routine BLAS1$VxROTG, using \(a = x\) and \(b = y\):

\[
\begin{bmatrix}
  x_i \\
  y_i
\end{bmatrix}
= \begin{bmatrix}
  c & s \\
  -s & c
\end{bmatrix}
\begin{bmatrix}
  x_i \\
  y_i
\end{bmatrix}
\]

The BLAS1$VxROT routines can be used to introduce zeros selectively into a matrix.

**Example**

```c
C To rotate the first two rows of a matrix and zero
    C out the element in the first column of the second row:
C
INTEGER INCX,N
REAL X(20,20),A,B,C,S
INCX = 20
N = 20
A = X(1,1)
B = X(2,1)
CALL BLAS1$VSR0TG(A,B,C,S)
CALL BLAS1$VSROT(N,X,INCX,X(2,1),INCX,C,S)
```

MTH-177
BLAS1$VxROTG—Generate the Elements for a Givens Plane Rotation

The Generate the Elements for a Givens Plane Rotation routines construct a Givens plane rotation that eliminates the second element of a two-element vector.

Format

BLAS1$VSROTG  a , b , c , s
BLAS1$VDROTG  a , b , c , s
BLAS1$VGROTG  a , b , c , s
BLAS1$VCROTG  a , b , c , s
BLAS1$VZROTG  a , b , c , s
BLAS1$VWROTG  a , b , c , s

Use BLAS1$VSROTG for single-precision real operations. Use BLAS1$VDROTG for double-precision real (D-floating) operations and BLAS1$VGROTG for double-precision real (G-floating) operations.

Use BLAS1$VCROTG for single-precision complex operations. Use BLAS1$VZROTG for double-precision complex (D-floating) operations and BLAS1$VWROTG for double-precision complex (G-floating) operations.

Returns

None.

Arguments

a

OpenVMS usage floating_point or complex_number
type F_floating, D_floating, G_floating real or F_floating,
D_floating, G_floating complex
access modify
mechanism by reference

On entry, first element of the input vector. On exit, rotated element r. The a argument is the address of a floating-point or floating-point complex number that is this vector element.

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for a</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROTG</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDROTG</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGROTG</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCROTG</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZROTG</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWROTG</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>
**b**

OpenVMS usage floating_point or complex_number

type F_floating, D_floating, G_floating real or F_floating,
     D_floating, G_floating complex

access modify

mechanism by reference

On entry, second element of the input vector. On exit from BLAS1$VSROTG,
BLAS1$VDROTG, and BLAS1$VGROTG, reconstruction element z. (See the
Description section for more information about z.) The b argument is the address
of a floating-point or floating-point complex number that is this vector element.

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for b</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROTG</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDROTG</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGROTG</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCROTG</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZROTG</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWROTG</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

**c**

OpenVMS usage floating_point

type F_floating, D_floating, or G_floating real

access write only

mechanism by reference

First rotation element, which can be interpreted as the cosine of the angle of
rotation. The c argument is the address of a floating-point or floating-point
complex number that is this vector element.

Specify the data type (which is always real) as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for c</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSROTG and</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VCROTG</td>
<td></td>
</tr>
<tr>
<td>BLAS1$VDROTG and</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VZROTG</td>
<td></td>
</tr>
<tr>
<td>BLAS1$VGROTG and</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VWROTG</td>
<td></td>
</tr>
</tbody>
</table>

**s**

OpenVMS usage floating_point or complex_number

type F_floating, D_floating, G_floating real or F_floating,
     D_floating, G_floating complex

access write only

mechanism by reference
Second rotation element, which can be interpreted as the sine of the angle of rotation. The s argument is the address of a floating-point or floating-point complex number that is this vector element.

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for s</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$V\times$ROTG</td>
<td></td>
</tr>
<tr>
<td>F-floating real</td>
<td></td>
</tr>
<tr>
<td>BLAS1$V\times$ROTG</td>
<td>D-floating real</td>
</tr>
<tr>
<td>G-floating real</td>
<td></td>
</tr>
<tr>
<td>BLAS1$V\times$ROTG</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$V\times$ROTG</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$V\times$ROTG</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

Description

BLAS1$V\times$ROTG, BLAS1$V\times$DROTG, and BLAS1$V\times$GROTG construct a real Givens plane rotation. BLAS1$V\times$CROTG, BLAS1$V\times$ZROTG, and BLAS1$V\times$WROTG construct a complex Givens plane rotation. The Givens plane rotation eliminates the second element of a two-element vector. The elements of the vector are real or complex single-precision or double-precision (D and G) numbers. The routine name determines the data type you should specify for arguments a, b, and s. Specify the same data type for each of these arguments.

BLAS1$V\times$ROTG, BLAS1$V\times$DROTG, and BLAS1$V\times$GROTG can use the reconstruction element z to store the rotation elements for future use. There is no counterpart to the term z for BLAS1$V\times$CROTG, BLAS1$V\times$ZROTG, and BLAS1$V\times$WROTG.

The BLAS1$V\times$ROTG routines can be used to introduce zeros selectively into a matrix.

For BLAS1$V\times$DROTG, BLAS1$V\times$GROTG, BLAS1$V\times$ZROTG, and BLAS1$V\times$WROTG (the double-precision routines), the elements of the vector are scaled to avoid intermediate overflow or underflow. BLAS1$V\times$SORTG and BLAS1$V\times$CROTG (the single-precision routines) use a backup data type to avoid intermediate underflow or overflow, which may cause the final result to differ from the original FORTRAN routine.

BLAS1$V\times$SORTG, BLAS1$V\times$DROTG, and BLAS1$V\times$GROTG — Real Givens Plane Rotation

Given the elements a and b of an input vector, BLAS1$V\times$SORTG, and BLAS1$V\times$DROTG, BLAS1$V\times$GROTG calculate the elements c and s of an orthogonal matrix such that:

\[
\begin{bmatrix}
    -s & c \\
    -c & s
\end{bmatrix}
\begin{bmatrix}
    a \\
    b
\end{bmatrix} =
\begin{bmatrix}
    r \\
    0
\end{bmatrix}
\]
A real Givens plane rotation is constructed for values \( a \) and \( b \) by computing values for \( r, c, s, \) and \( z, \) as follows:

\[
r = \sqrt{a^2 + b^2}
\]

where:

\[
p = \text{SIGN}(a) \text{ if } |a| > |b|
\]

\[
p = \text{SIGN}(b) \text{ if } |a| \leq |b|
\]

\[
c = \frac{a}{r} \text{ if } r \neq 0
\]

\[
c = 1 \text{ if } r = 0
\]

\[
s = \frac{b}{r} \text{ if } r \neq 0
\]

\[
s = 0 \text{ if } r = 0
\]

\[
z = s \text{ if } |a| > |b|
\]

\[
z = \frac{1}{c} \text{ if } |a| \leq |b| \text{ and } c \neq 0 \text{ and } r \neq 0
\]

\[
z = 1 \text{ if } |a| \leq |b| \text{ and } c = 0 \text{ and } r \neq 0
\]

\[
z = 0 \text{ if } r = 0
\]

BLAS1$VSROTG, BLAS1$VDROTG, and BLAS1$VGROTG can use the reconstruction element \( z \) to store the rotation elements for future use. The quantities \( c \) and \( s \) are reconstructed from \( z \) as follows:

For \( |z| = 1 \), \( c = 0 \) and \( s = 1.0 \)

For \( |z| < 1 \), \( c = \sqrt{1 - z^2} \) and \( s = z \)

For \( |z| > 1 \), \( c = \frac{1}{z} \) and \( s = \sqrt{1 - c^2} \)

The arguments \( c \) and \( s \) can be passed to the BLAS1$VxROT routines.

**BLAS1$VCROTG, BLAS1$VZROTG, and BLAS1$VWROTG — Complex Givens Plane Rotation**

Given the elements \( a \) and \( b \) of an input vector, BLAS1$VCROTG, BLAS1$VZROTG, and BLAS1$VWROTG calculate the elements \( c \) and \( s \) of an orthogonal matrix such that:

\[
\begin{bmatrix}
    c & s_1 + i \ast s_2 \\
    -s_1 + i \ast s_2 & c
\end{bmatrix}
\begin{bmatrix}
    a_1 + i \ast a_2 \\
    b_1 + i \ast b_2
\end{bmatrix} =
\begin{bmatrix}
    r_1 + i \ast r_2 \\
    0
\end{bmatrix}
\]

There are no BLAS Level 1 routines with which you can use complex \( c \) and \( s \) arguments.

**Example**

```c
C To generate the rotation elements for a vector of C elements a and b:
C
REAL A,B,C,S
CALL SROTG(A,B,C,S)
```
BLAS1$VxSCAL

BLAS1$VxSCAL—Scale the Elements of a Vector

The Scale the Elements of a Vector routines compute $a \cdot x$ where $a$ is a scalar number and $x$ is an $n$-element vector.

Format

BLAS1$VSSCAL n, a, x, incx
BLAS1$VDSCAL n, a, x, incx
BLAS1$VGSCAL n, a, x, incx
BLAS1$VCSCAL n, a, x, incx
BLAS1$VCSSCAL n, a, x, incx
BLAS1$VZSCAL n, a, x, incx
BLAS1$VWSCAL n, a, x, incx
BLAS1$VZDSCAL n, a, x, incx
BLAS1$VWGSCAL n, a, x, incx

Use BLAS1$VSSCAL to scale a real single-precision vector by a real single-precision scalar.

Use BLAS1$VDSCAL to scale a real double-precision (D-floating) vector by a real double-precision (D-floating) scalar. Use BLAS1$VGSCAL to scale a real double-precision (G-floating) vector by a real double-precision (G-floating) scalar.

Use BLAS1$VCSCAL to scale a complex single-precision vector by a complex single-precision scalar. Use BLAS1$VCSSCAL to scale a complex single-precision vector by a real single-precision scalar.

Use BLAS1$VZSCAL to scale a complex double-precision (D-floating) vector by a complex double-precision (D-floating) scalar. Use BLAS1$VWSCAL to scale a complex double-precision (G-floating) vector by a complex double-precision (G-floating) scalar. Use BLAS1$VZDSCAL to scale a complex double-precision (D-floating) vector by a real double-precision (D-floating) scalar. Use BLAS1$VWGSCAL to scale a complex double-precision (G-floating) vector by a real double-precision (G-floating) scalar.

Returns

None.

Arguments

$n$

<table>
<thead>
<tr>
<th>OpenVMS usage</th>
<th>longword_signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>longword integer (signed)</td>
</tr>
<tr>
<td>access</td>
<td>read only</td>
</tr>
<tr>
<td>mechanism</td>
<td>by reference</td>
</tr>
</tbody>
</table>

Number of elements in vector $x$ to be scaled. The $n$ argument is the address of a signed longword integer containing the number of elements to be scaled. If you specify a value for $n$ that is less than or equal to 0, then $x$ is unchanged.
BLAS1$VxSCAL

**a**
OpenVMS usage floating_point or complex_number

**type**
F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex

**access**
read only

**mechanism**
by reference

Scalar multiplier for the elements of vector \( x \). The \( a \) argument is the address of a floating-point or floating-point complex number that is this multiplier.

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for a</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSSCAL</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VCSSCAL</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VDSCAL and</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VZDSCAL</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VGSCAL and</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VWGSCAL</td>
<td>G-floating complex</td>
</tr>
<tr>
<td>BLAS1$VCSCAL</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZSCAL</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$VWSCAL</td>
<td>G-floating complex</td>
</tr>
</tbody>
</table>

If you specify 1.0 for \( a \), then \( x \) is unchanged.

**x**
OpenVMS usage floating_point or complex_number

**type**
F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex

**access**
modify

**mechanism**
by reference, array reference

Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( incx \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. On entry, this argument is an array of length at least

\[
1 + (n - 1) \cdot |incx|
\]

where:

\( n \) = number of vector elements specified in \( n \)

\( incx \) = increment argument for the array \( x \) specified in \( incx \)

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for x</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSSCAL</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDSCAL</td>
<td>D-floating real</td>
</tr>
<tr>
<td>BLAS1$VGSCAL</td>
<td>G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCSCAL and</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZSCAL</td>
<td>D-floating complex</td>
</tr>
</tbody>
</table>
### BLAS1$V$xSCAL

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for x</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$V$ZSCAL and</td>
<td>D-floating complex</td>
</tr>
<tr>
<td>BLAS1$V$ZDSCAL</td>
<td></td>
</tr>
<tr>
<td>BLAS1$V$WSCAL and</td>
<td>G-floating complex</td>
</tr>
<tr>
<td>BLAS1$V$WGSCAL</td>
<td></td>
</tr>
</tbody>
</table>

On exit, $x$ is an array of length at least $1 + (n - 1) \times |incx|$ where:

- $n$ = number of vector elements specified in $n$
- $incx$ = increment argument for the array $x$ specified in $incx$

After the call to BLAS1$V$xSCAL, $x_i$ is replaced by $a \times x_i$. If $a$ shares a memory location with any element of the vector $x$, results are unpredictable.

#### $incx$

- **OpenVMS usage**: longword signed
- **type**: longword integer (signed)
- **access**: read only
- **mechanism**: by reference

Increment argument for the array $x$. The $incx$ argument is the address of a signed longword integer containing the increment argument. If $incx$ is greater than 0, then $x$ is referenced forward in array $x$; that is, $x_i$ is referenced in:

$$ x(i + (i - 1) \times incx) $$

where:

- $x$ = array specified in $x$
- $i$ = element of the vector $x$
- $incx$ = increment argument for the array $x$ specified in $incx$

If you specify a negative value for $incx$, it is interpreted as the absolute value of $incx$. If $incx$ equals 0, the results are unpredictable.

#### Description

BLAS1$V$xSCAL computes $a \times x$ where $a$ is a scalar number and $x$ is an $n$-element vector. The computation is expressed as follows:

$$ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \leftarrow a \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} $$

Vector $x$ contains $n$ elements that are accessed from array $x$ by stepping $incx$ elements at a time. The vector $x$ can be a row or a column of a matrix. Both forward and backward indexing are permitted.

The public-domain BLAS Level 1 $x$SCAL routines require a positive value for $incx$. The Run-Time Library BLAS Level 1 routines interpret a negative value for $incx$ as the absolute value of $incx$.

The algorithm does not provide a special case for $a = 0$. Therefore, specifying 0 for $a$ has the effect of setting to zero all elements of the vector $x$ using vector operations.
Example

C To scale a vector x by 2.0 using SCAL:
C
INTEGER INCX,N
REAL X(20),A
INCX = 1
A = 2
N = 20
CALL BLAS1$VSSCAL(N,A,X,INCX)
BLAS1$VxSWAP

BLAS1$VxSWAP—Swap the Elements of Two Vectors

The Swap the Elements of Two Vectors routines swap \( n \) elements of the vector \( x \) with the vector \( y \).

Format

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSSWAP</td>
<td>( n, x, incx, y, incy )</td>
</tr>
<tr>
<td>BLAS1$VDSWAP</td>
<td>( n, x, incx, y, incy )</td>
</tr>
<tr>
<td>BLAS1$VCSWAP</td>
<td>( n, x, incx, y, incy )</td>
</tr>
<tr>
<td>BLAS1$VZSWAP</td>
<td>( n, x, incx, y, incy )</td>
</tr>
</tbody>
</table>

Use BLAS1$VSSWAP for single-precision real operations and BLAS1$VDSWAP for double-precision real (D or G) operations.

Use BLAS1$VCSWAP for single-precision complex operations and BLAS1$VZSWAP for double-precision complex (D or G) operations.

Returns

None.

Arguments

\( n \)

OpenVMS usage: longword Signed
Type: longword integer (signed)
Access: read only
Mechanism: by reference

Number of elements in vector \( x \) to be swapped. The \( n \) argument is the address of a signed longword integer containing the number of elements to be swapped.

\( x \)

OpenVMS usage: floating point or complex number
Type: F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex
Access: modify
Mechanism: by reference, array reference

Array containing the elements to be accessed. All elements of array \( x \) are accessed only if the increment argument of \( x \), called \( incx \), is 1. The \( x \) argument is the address of a floating-point or floating-point complex number that is this array. On entry, this argument is an array of length at least

\[ 1 + (n - 1) \times |incx| \]

where:

\[ n = \text{number of vector elements specified in } n \]

\[ incx = \text{increment argument for the array } x \text{ specified in } incx \]
Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for x</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSSWAP</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDSWAP</td>
<td>D-floating or G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCZWAP</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZSWAP</td>
<td>D-floating or G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then \( x \) and \( y \) are unchanged. If any element of \( x \) shares a memory location with an element of \( y \), the results are unpredictable.

On exit, \( x \) is an array of length at least

\[
1 + (n - 1) \cdot |incx|
\]

where:

- \( n \) = number of vector elements specified in \( n \)
- \( incx \) = increment argument for the array \( x \) specified in \( incx \)

After the call to BLAS1$VxSWAP, \( n \) elements of the array specified by \( x \) are interchanged with \( n \) elements of the array specified by \( y \).

\( incx \)

OpenVMS usage: longword_signed

- type = longword_integer (signed)
- access = read only
- mechanism = by reference

Increment argument for the array \( x \). The \( incx \) argument is the address of a signed longword integer containing the increment argument. If \( incx \) is greater than or equal to 0, then \( x \) is referenced forward in array \( x \); that is, \( x_i \) is referenced in:

\[
x(1 + (i - 1) \cdot incx)
\]

where:

- \( x \) = array specified in \( x \)
- \( i \) = element of the vector \( z \)
- \( incx \) = increment argument for the array \( x \) specified in \( incx \)

If \( incx \) is less than 0, then \( x \) is referenced backward in array \( x \); that is, \( x_i \) is referenced in:

\[
x(1 + (n - i) \cdot |incx|)
\]

where:

- \( x \) = array specified in \( x \)
- \( n \) = number of vector elements specified in \( n \)
- \( i \) = element of the vector \( z \)
- \( incx \) = increment argument for the array \( x \) specified in \( incx \)
**BLAS1$VxSWAP**

**y**
- OpenVMS usage: floating_point or complex_number
- Type: F_floating, D_floating, G_floating real or F_floating, D_floating, G_floating complex
- Access: modify
- Mechanism: by reference, array reference

Array containing the elements to be accessed. All elements of array y are accessed only if the increment argument of y, called **incy**, is 1. The y argument is the address of a floating-point or floating-point complex number that is this array. On entry, this argument is an array of length at least
\[ 1 + (n - 1) \times |\text{incy}| \]
where:
- \( n \) = number of vector elements specified in \( n \)
- \( incy \) = increment argument for the array y specified in **incy**

Specify the data type as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Data Type for y</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS1$VSSWAP</td>
<td>F-floating real</td>
</tr>
<tr>
<td>BLAS1$VDSWAP</td>
<td>D-floating or G-floating real</td>
</tr>
<tr>
<td>BLAS1$VCSWAP</td>
<td>F-floating complex</td>
</tr>
<tr>
<td>BLAS1$VZSWAP</td>
<td>D-floating or G-floating complex</td>
</tr>
</tbody>
</table>

If \( n \) is less than or equal to 0, then x and y are unchanged. If any element of x shares a memory location with an element of y, the results are unpredictable.

On exit, y is an array of length at least
\[ 1 + (n - 1) \times |\text{incy}| \]
where:
- \( n \) = number of vector elements specified in \( n \)
- \( incy \) = increment argument for the array y specified in **incy**

After the call to BLAS1$VxSWAP, \( n \) elements of the array specified by x are interchanged with \( n \) elements of the array specified by y.

**incy**
- OpenVMS usage: longword_signed
- Type: longword integer (signed)
- Access: read only
- Mechanism: by reference

Increment argument for the array y. The **incy** argument is the address of a signed longword integer containing the increment argument. If **incy** is greater than or equal to 0, then y is referenced forward in array y; that is, \( y_k \) is referenced in:
\[ y(1 + (i - 1) \times \text{incy}) \]
where:

\[ y = \text{array specified in } y \]
\[ i = \text{element of the vector } y \]
\[ incy = \text{increment argument for the array } y \text{ specified in } incy \]

If \( incy \) is less than 0, then \( y \) is referenced backward in array \( y \); that is, \( y_i \) is referenced in:

\[ y(1 + (n - i) \times |incy|) \]

where:

\[ y = \text{array specified in } y \]
\[ n = \text{number of vector elements specified in } n \]
\[ i = \text{element of the vector } y \]
\[ incy = \text{increment argument for the array } y \text{ specified in } incy \]

Description

BLAS1$VSSWAP$, BLAS1$VDSWAP$, BLAS1$VCSWAP$, and BLAS1$VZSWAP$ swap \( n \) elements of the vector \( x \) with the vector \( y \). Vectors \( x \) and \( y \) contain \( n \) elements that are accessed from arrays \( x \) and \( y \) by stepping \( incx \) and \( incy \) elements at a time. Both \( x \) and \( y \) are real or complex single-precision or double-precision (D and G) \( n \)-element vectors. The vectors can be rows or columns of a matrix. Both forward and backward indexing are permitted.

You can use the routine BLAS1$VxSWAP$ to invert the storage of elements of a vector within itself. If \( incx \) is greater than 0, then \( x_i \) can be moved from location \( x(1 + (i - 1) \times incx) \) to \( x(1 + (n - i) \times incx) \)

The following code fragment inverts the storage of elements of a vector within itself:

\[
NN = N/2 \\
LHALF = 1+(N-NN)\times INCX \\
\text{CALL BLAS1$VxSWAP$(NN,X,INCX,X(LHALF),-INCX)}
\]

BLAS1$VxSWAP$ does not check for a reserved operand.

Example

C To swap the contents of vectors \( x \) and \( y \):
C

```
INTEGER INCX, INCY, N
REAL X(20), Y(20)
INCX = 1
INCY = 1
N = 20
CALL BLAS1$VSSWAP(N, X, INCX, Y, INCY)
```

C To invert the order of storage of the elements of \( x \) within itself; that is, to move \( x(1), \ldots, x(100) \) to \( x(100), \ldots, x(1) \):
C

```
INCX = 1
INCY = -1
N = 50
CALL BLAS1$VZSWAP(N, X, INCX, X(51), INCY)
```
MTH$VxFOLRy_MA_V15—First Order Linear Recurrence — Multiplication and Addition

The First Order Linear Recurrence — Multiplication and Addition routines provide a vectorized algorithm for the linear recurrence relation that includes both multiplication and addition operations.

Format

- MTH$VJFOLRP_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VFFOLRP_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VDFFOLRP_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VGFFOLRP_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VJFOLRN_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VFFOLRN_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VDFFOLRN_MA_V15 n,a,inca,b,incb,c,incc
- MTH$VGFFOLRN_MA_V15 n,a,inca,b,incb,c,incc

To obtain one of the preceding formats, substitute the following for x and y in MTH$VxFOLRy_MA_V15:

- x = J for longword integer, F for F-floating, D for D-floating, G for G-floating
- y = P for a positive recursion element, N for a negative recursion element

Returns

None.

Arguments

- n
  - OpenVMS usage: longword_signed
  - type: longword integer (signed)
  - access: read only
  - mechanism: by reference

  Length of the linear recurrence. The n argument is the address of a signed longword integer containing the length.

- a
  - OpenVMS usage: longword_signed or floating_point
  - type: longword integer (signed), F_floating, D_floating, or G_floating
  - access: read only
  - mechanism: by reference, array reference

  Array of length at least
  
  \[ 1 + (n - 1) \cdot \text{inca} \]

  where:

  \[ n = \text{length of the linear recurrence specified in } n \]
inca = increment argument for the array a specified in inca
The a argument is the address of a longword integer or floating-point that is this array.

inca
OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference
Increment argument for the array a. The inca argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for inca.

b
OpenVMS usage longword_signed or floating_point
type longword integer (signed), F_floating, D_floating, or G_floating
access read only
mechanism by reference, array reference
Array of length at least
1 + (n - 1) * incb
where:

n = length of the linear recurrence specified in n
incb = increment argument for the array b specified in incb
The b argument is the address of a longword integer or floating-point number that is this array.

incb
OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference
Increment argument for the array b. The incb argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for incb.

c
OpenVMS usage longword_signed or floating_point
type longword integer (signed), F_floating, D_floating, or G_floating
access modify
mechanism by reference, array reference
Array of length at least
1 + n * incc
where:

n = length of the linear recurrence specified in n
incc = increment argument for the array c specified in incc
The c argument is the address of a longword integer or floating-point number that is this array.
MTH$VxFOLRy_MA_V15

**incc**

OpenVMS usage: longword_signed

Type: longword integer (signed)

Access: read only

Mechanism: by reference

Increment argument for the array c. The **incc** argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for **incc**. Do not specify 0 for **incc**.

**Description**

MTH$VxFOLRy_MA_V15 is a group of routines that provides a vectorized algorithm for computing the following linear recurrence relation:

\[ C(I + 1) = ++/- C(I) \cdot A(I) + B(I) \]

---

**Note**

Save the contents of vector registers V0 through V15 before you call this routine.

Call this routine to utilize vector hardware when computing the recurrence. As an example, the call from VAX FORTRAN is as follows:

K1 = ....
K2 = ....
K3 = ....

CALL MTH$VxFOLRy_MA_V15(N,A(K1),INCA,B(K2),INCB,C(K3),INCC)

The preceding FORTRAN call replaces the following loop:

K1 = ....
K2 = ....
K3 = ....

DO I = 1, N
C(K3+I*INCC) = ++/- C(K3+(I-1)*INCC) \cdot A(K1+(I-1)*INCA) + B(K2+(I-1)*INCB)
ENDDO

The arrays used in a FOLR expression must be of the same data type in order to be vectorized and user callable. The MTH$ FOLR routines assume that all of the arrays are of the same data type.

This group of routines, MTH$VxFOLRy_MA_V15 (and also MTH$VxFOLRy_z_V8) save the result of each iteration of the linear recurrence relation in an array. This is different from the behavior of MTH$VxFOLRLy_MA_V5 and MTH$VxFOLRLy_z_V2, which return only the result of the last iteration of the linear recurrence relation.

For the output array (c), the increment argument (**incc**) cannot be 0. However, you can specify 0 for the input increment arguments (**inca** and **incb**). In that case, the input will be treated as a scalar value and broadcast to a vector input with all vector elements equal to the scalar value.

In MTH$VxFOLRy_MA_V15, array c can overlap array a and array b, or both, as long as the address of array element \( c_x \) is not also the address of an element of \( a \) or \( b \) that will be referenced at a future time in the recurrence relation. For example, in the following code fragment you must ensure that the address of \( c(I + i \cdot incc) \) does not equal the address of either \( a(j \cdot inca) \) or \( b(k \cdot incb) \) for
\begin{verbatim}
1 \leq i \leq n \text{ and } j \geq i + 1.

DO I = 1,N
    C(1+I*INCC) = C(1+(I-1)*INCC) * A(I+(I-1)*INCA) + B(1+(I-1)*INCB)
ENDDO

Examples

1. C
   The following FORTRAN loop computes a linear recurrence.
   INTEGER I
   DIMENSION A(200), B(50), C(50)
   EQUIVALENCE (B,C)
   C(4) = ....
   DO I = 5, 50
       C(I) = C((I-1)) * A(I*3) + B(I)
   ENDDO
   CALL MTH$VFF0LRP_MA_V15(46, A(15), 3, B(5), 1, C(4), 1)

   C
   The following call from FORTRAN to a FOLR routine replaces the preceding loop.
   DIMENSION A(200), B(50), C(50)
   EQUIVALENCE (B,C)
   C(4) = ....
   CALL MTH$VFFOLRP_MA_V15(46, A(INCA), INCA, B(INCB), INCB, C(K), INCC)

2. C
   The following FORTRAN loop computes a linear recurrence.
   INTEGER K,N,INCA,INCB,INCC,I
   DIMENSION A(30), B(6), C(50)
   K = 44
   N = 6
   INCA = 5
   INCB = 1
   INCC = 1
   DO I = 1, N
       C(K+I*INCC) = -C(K+(I-1)*INCC) * A(I*INCA) + B(I*INCB)
   ENDDO
   CALL MTH$VFFOLRN_MA_V15(N, A(INCA), INCA, B(INCB), INCB, C(K), INCC)

   C
   The following call from FORTRAN to a FOLR routine replaces the preceding loop.
   INTEGER K,N,INCA,INCB,INCC
   DIMENSION A(30), B(6), C(50)
   K = 44
   N = 6
   INCA = 5
   INCB = 1
   INCC = 1
   CALL MTH$VFFOLRN_MA_V15(N, A(INCA), INCA, B(INCB), INCB, C(K), INCC)
\end{verbatim}
The First Order Linear Recurrence — Multiplication or Addition routines provide a vectorized algorithm for the linear recurrence relation that includes either a multiplication or an addition operation, but not both.

Format

MTH$VJFOLRP_M_V8  n,a,inca,b,incb
MTH$VFOLRP_M_V8   n,a,inca,b,incb
MTH$VDOLRP_M_V8   n,a,inca,b,incb
MTH$VGOLRP_M_V8   n,a,inca,b,incb
MTH$VJFOLRN_M_V8  n,a,inca,b,incb
MTH$VFOLRN_M_V8   n,a,inca,b,incb
MTH$VDOLRN_M_V8   n,a,inca,b,incb
MTH$VGOLRN_M_V8   n,a,inca,b,incb
MTH$VJFOLRP_A_V8  n,a,inca,b,incb
MTH$VFOLRP_A_V8   n,a,inca,b,incb
MTH$VDOLRP_A_V8   n,a,inca,b,incb
MTH$VGOLRP_A_V8   n,a,inca,b,incb
MTH$VJFOLRN_A_V8  n,a,inca,b,incb
MTH$VFOLRN_A_V8   n,a,inca,b,incb
MTH$VDOLRN_A_V8   n,a,inca,b,incb
MTH$VGOLRN_A_V8   n,a,inca,b,incb

To obtain one of the preceding formats, substitute the following for \( x, y, \) and \( z \) in MTH$VxFOLRy_z_V8:

- \( x = J \) for longword integer, \( F \) for F-floating, \( D \) for D-floating, \( G \) for G-floating
- \( y = P \) for a positive recursion element, \( N \) for a negative recursion element
- \( z = M \) for multiplication, \( A \) for addition

Returns

None.

Arguments

- \( n \)
  - OpenVMS usage: longword_signed
  - type: longword integer (signed)
  - access: read only
  - mechanism: by reference

Length of the linear recurrence. The \( n \) argument is the address of a signed longword integer containing the length.
**a**
OpenVMS usage  longword_signed or floating_point
type  longword integer (signed), F_floating, D_floating, or G_floating
access  read only
mechanism  by reference, array reference

Array of length at least
\[ 1 + (n - 1) \cdot \text{inc}\]

where:
- \( n \) = length of the linear recurrence specified in \( n \)
- \( \text{inc} \) = increment argument for the array \( a \) specified in \( \text{inc} \)

The \( a \) argument is the address of a longword integer or floating-point that is this array.

**inc**
OpenVMS usage  longword_signed
type  longword integer (signed)
access  read only
mechanism  by reference

Increment argument for the array \( a \). The \( \text{inc} \) argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for \( \text{inc} \).

**b**
OpenVMS usage  longword_signed or floating_point
type  longword integer (signed), F_floating, D_floating, or G_floating
access  modify
mechanism  by reference, array reference

Array of length at least
\[ 1 + (n - 1) \cdot \text{inc}\]

where:
- \( n \) = length of the linear recurrence specified in \( n \)
- \( \text{inc} \) = increment argument for the array \( b \) specified in \( \text{inc} \)

The \( b \) argument is the address of a longword integer or floating-point number that is this array.

**inc**
OpenVMS usage  longword_signed
type  longword integer (signed)
access  read only
mechanism  by reference

Increment argument for the array \( b \). The \( \text{inc} \) argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for \( \text{inc} \).
MTH\$VxFOLRy_z_V8

Description

MTH\$VxFOLRy_z_V8 is a group of routines that provide a vectorized algorithm for computing one of the following linear recurrence relations:

\[ B(I) = \pm B(I-1) \ast A(I) \]
or

\[ B(I) = \pm B(I-1) + A(I) \]

For the first relation, specify M for 2 in the routine name to denote multiplication; for the second relation, specify A for 2 in the routine name to denote addition.

---

**Note**

Save the contents of vector registers V0 through V8 before you call this routine.

Call this routine to utilize vector hardware when computing the recurrence. As an example, the call from VAX FORTRAN is as follows:

```
CALL MTH\$VxFOLRy_z_V8(N,A(K1),INCA,B(K2),INCB)
```

The preceding FORTRAN call replaces the following loop:

```
K1 = ....
K2 = ....
DO I = 1, N
B(K2+I*INCB) = \{+/\}B(K2+(I-1)*INCB) \{+/*\} A(K1+(I-1)*INCA)
ENDDO
```

The arrays used in a FOLR expression must be of the same data type in order to be vectorized and user callable. The MTH\$FOLR routines assume that all of the arrays are of the same data type.

This group of routines, MTH\$VxFOLRy_z_V8 (and also MTH\$VxFOLRy_MA_V15) save the result of each iteration of the linear recurrence relation in an array. This is different from the behavior of MTH\$VxFOLRLy_MA_V5 and MTH\$VxFOLRLy_z_V2, which return only the result of the last iteration of the linear recurrence relation.

For the output array (b), the increment argument (incb) cannot be 0. However, you can specify 0 for the input increment argument (inca). In that case, the input will be treated as a scalar and broadcast to a vector input with all vector elements equal to the scalar value.
Examples

1.

The following FORTRAN loop computes a linear recurrence.

```
D_FLOAT
INTEGER N,INCA,INCB,I
DIMENSION A(30), B(13)
N = 6
INCA = 5
INCB = 2
DO I = 1, N
   B(1+I*INCB) = -B(1+(I-1)*INCB) * A(I*INCA)
ENDDO
```

The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
D_FLOAT
INTEGER N,INCA,INCB
REAL*8 A(30), B(13)
N = 6
INCA = 5
INCB = 2
CALL MTHSVDPOLRN_M_V8(N, A(INCA), INCA, B(INCB), INCB)
```

2.

The following FORTRAN loop computes a linear recurrence.

```
G_FLOAT
INTEGER N,INCA,INCB
DIMENSION A(30), B(13)
N = 5
INCA = 5
INCB = 2
DO I = 2, N
   B(1+I*INCB) = B((I-1)*INCB) + A(I*INCA)
ENDDO
```

The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
G_FLOAT
INTEGER N,INCA,INCB
REAL*8 A(30), B(13)
N = 5
INCA = 5
INCB = 2
CALL MTHSVGPOLRP_A_V8(N, A(INCA), INCA, B(INCB), INCB)
```
The First Order Linear Recurrence — Multiplication and Addition — Last Value routines provide a vectorized algorithm for the linear recurrence relation that includes both multiplication and addition operations. Only the last value computed is stored.

**Format**

MTH$VJFOLRLP_MA_V5 n,a,inc_a,inc_b,t
MTH$VFFOLRLP_MA_V5 n,a,inc_a,inc_b,t
MTH$VDFFOLRLP_MA_V5 n,a,inc_a,inc_b,t
MTH$VGFOLRLP_MA_V5 n,a,inc_a,inc_b,t
MTH$VJFFOLRLN_MA_V5 n,a,inc_a,inc_b,t
MTH$VFFOLRLN_MA_V5 n,a,inc_a,inc_b,t
MTH$VDFFOLRLN_MA_V5 n,a,inc_a,inc_b,t
MTH$VGFOLRLN_MA_V5 n,a,inc_a,inc_b,t

To obtain one of the preceding formats, substitute the following for $x$ and $y$ in MTH$VxFOJRLy_MA_V5:

$x = J$ for longword integer, $F$ for F-floating, $D$ for D-floating, $G$ for G-floating

$y = P$ for a positive recursion element, $N$ for a negative recursion element

**Returns**

OpenVMS usage longword_signed or floating_point
type longword integer (signed), F_floating, D_floating or G_floating
access write only
mechanism by value

The function value is the result of the last iteration of the linear recurrence relation. The function value is returned in R0 or R0 and R1.

**Arguments**

$n$

OpenVMS usage longword_signed
type longword integer (signed)
access read only
mechanism by reference

Length of the linear recurrence. The $n$ argument is the address of a signed longword integer containing the length.

$a$

OpenVMS usage longword_signed or floating_point
type longword integer (signed), F_floating, D_floating, or G_floating
access read only
mechanism by reference, array reference
Array of length at least
\[ 1 + (n - 1) \times inca \]
where:
\begin{align*}
  n &= \text{length of the linear recurrence specified in } n \\
  inca &= \text{increment argument for the array } a \text{ specified in } inca
\end{align*}
The \( a \) argument is the address of a longword integer or floating-point that is this array.

\[ \text{inca} \]
OpenVMS usage: longword_signed
Type: longword integer (signed)
Access: read only
Mechanism: by reference
Increment argument for the array \( a \). The \( \text{inca} \) argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for \( \text{inca} \).

\[ b \]
OpenVMS usage: longword_signed or floating_point
Type: longword integer (signed), F_floating, D_floating, or G_floating
Access: read only
Mechanism: by reference, array reference
Array of length at least
\[ 1 + (n - 1) \times incb \]
where:
\begin{align*}
  n &= \text{length of the linear recurrence specified in } n \\
  incb &= \text{increment argument for the array } b \text{ specified in } incb
\end{align*}
The \( b \) argument is the address of a longword integer or floating-point number that is this array.

\[ \text{incb} \]
OpenVMS usage: longword_signed
Type: longword integer (signed)
Access: read only
Mechanism: by reference
Increment argument for the array \( b \). The \( \text{incb} \) argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for \( \text{incb} \).

\[ t \]
OpenVMS usage: longword_signed or floating_point
Type: longword integer (signed), F_floating, D_floating, or G_floating
Access: modify
Mechanism: by reference
Variable containing the starting value for the recurrence; overwritten with the value computed by the last iteration of the linear recurrence relation. The \( t \) argument is the address of a longword integer or floating-point number that is this value.
**MTH$VxFOLRLy_MA_V5**

**Description**

MTH$VxFOLRLy_MA_V5 is a group of routines that provide a vectorized algorithm for computing the following linear recurrence relation. (The $T$ on the right side of the equation is the result of the previous iteration of the loop.)

$$T = +/- T \cdot A(I) + B(I)$$

**Note**

Save the contents of vector registers $V_0$ through $V_5$ before you call this routine.

Call this routine to utilize vector hardware when computing the recurrence. As an example, the call from VAX FORTRAN is as follows:

```fortran
CALL MTH$VxFOLRy_MA_V5(N,A(K1),INCA,B(K2),INCB,T)
```

The preceding FORTRAN call replaces the following loop:

```fortran
K1 = ...
K2 = ...
DO I = 1, N
    T = (+/-) T * A(K1 + (I-1)*INCA) + B(K1 + (I-1)*INCB)
ENDDO
```

The arrays used in a FOLR expression must be of the same data type in order to be vectorized and user callable. The MTH$ FOLR routines assume that all of the arrays are of the same data type.

This group of routines, MTH$VxFOLRLy_MA_V5 (and also MTH$VxFOLRLy_z_V2) returns only the result of the last iteration of the linear recurrence relation. This is different from the behavior of MTH$VxFOLRy_MA_V15 (and also MTH$VxFOLRy_z_V8), which save the result of each iteration of the linear recurrence relation in an array.

If you specify 0 for the input increment arguments (INCA and INCB), the input will be treated as a scalar and broadcast to a vector input with all vector elements equal to the scalar value.

**Examples**

1. C

   ```fortran
   INTEGER N, INCA, INCB, I
   REAL*8 A(30), B(6), T
   N = 6
   INCA = 5
   INCB = 1
   T = 789847562
   DO I = 1, N
       T = -T * A(I*INCA) + B(I*INCB)
   ENDDO
   ```

MTH-200
The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
G_FLOAT
INTEGER N, INCA, INCB
DIMENSION A(30), B(6), T
N = 6
INCA = 5
INCB = 1
T = 78.9847562
T = MTH$VGFLRMA_V5(N, A(INCA), INCA, B(INCB), INCB, T)
```

The following FORTRAN loop computes a linear recurrence.

```
G_FLOAT
INTEGER N, INCA, INCB, I
REAL*8 A(30), B(6), T
N = 6
INCA = 5
INCB = 1
T = 78.9847562
DO I = 1, N
T = T * A(I*INCA) + B(I*INCB)
ENDDO
```

The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
G_FLOAT
INTEGER N, INCA, INCB
DIMENSION A(30), B(6), T
N = 6
INCA = 5
INCB = 1
T = 78.9847562
T = MTH$VGFLRPMA_V5(N, A(INCA), INCA, B(INCB), INCB, T)
```
**MTH$VxFOLRLy_z_V2—First Order Linear Recurrence — Multiplication or Addition — Last Value**

The First Order Linear Recurrence — Multiplication or Addition — Last Value routines provide a vectorized algorithm for the linear recurrence relation that includes either a multiplication or an addition operation. Only the last value computed is stored.

**Format**

MTH$VJFOLRLP_M_V2 n,a,inca,t  
MTH$VFFOLRLP_M_V2 n,a,inca,t  
MTH$VDFOLRLP_M_V2 n,a,inca,t  
MTH$VGFOLRLP_M_V2 n,a,inca,t  
MTH$VJFOLRLN_M_V2 n,a,inca,t  
MTH$VFFOLRLN_M_V2 n,a,inca,t  
MTH$VDFOLRLN_M_V2 n,a,inca,t  
MTH$VGFOLRLN_M_V2 n,a,inca,t  
MTH$VJFOLRLP_A_V2 n,a,inca,t  
MTH$VFFOLRLP_A_V2 n,a,inca,t  
MTH$VDFOLRLP_A_V2 n,a,inca,t  
MTH$VGFOLRLP_A_V2 n,a,inca,t  
MTH$VJFOLRLN_A_V2 n,a,inca,t  
MTH$VFFOLRLN_A_V2 n,a,inca,t  
MTH$VDFOLRLN_A_V2 n,a,inca,t  
MTH$VGFOLRLN_A_V2 n,a,inca,t

To obtain one of the preceding formats, substitute the following for x, y, and z in MTH$VxFOLRLy_z_V2:

- x = J for longword integer, F for F-floating, D for D-floating, G for G-floating
- y = P for a positive recursion element, N for a negative recursion element
- z = M for multiplication, A for addition

**Returns**

OpenVMS usage longword_signed or floating_point  
Type longword integer (signed), F_floating, D_floating or G_floating  
Access write only  
Mechanism by value

The function value is the result of the last iteration of the linear recurrence relation. The function value is returned in R0 or R0 and R1.
Arguments

\( n \)

OpenVMS usage: longword_signed
Type: longword integer (signed)
Access: read only
Mechanism: by reference

Length of the linear recurrence. The \( n \) argument is the address of a signed longword integer containing the length.

\( a \)

OpenVMS usage: longword_signed or floating_point
Type: longword integer (signed), F_floating, D_floating, or G_floating
Access: read only
Mechanism: by reference, array reference

Array of length at least \( n \times inca \)

where:

\( n = \) length of the linear recurrence specified in \( n \)
\( inca = \) increment argument for the array \( a \) specified in \( inca \)

The \( a \) argument is the address of a longword integer or floating-point that is this array.

\( inca \)

OpenVMS usage: longword_signed
Type: longword integer (signed)
Access: read only
Mechanism: by reference

Increment argument for the array \( a \). The \( inca \) argument is the address of a signed longword integer containing the increment argument. For contiguous elements, specify 1 for \( inca \).

\( t \)

OpenVMS usage: longword_signed or floating_point
Type: longword integer (signed), F_floating, D_floating, or G_floating
Access: modify
Mechanism: by reference

Variable containing the starting value for the recurrence; overwritten with the value computed by the last iteration of the linear recurrence relation. The \( t \) argument is the address of a longword integer or floating-point number that is this value.
**Description**

MTH$VxFOLRLy_z_V2 is a group of routines that provide a vectorized algorithm for computing one of the following linear recurrence relations. (The $T$ on the right side of the following equations is the result of the previous iteration of the loop.)

$$T = \pm T \cdot A(I)$$

or

$$T = \pm T + A(I)$$

For the first relation, specify $M$ for $z$ in the routine name to denote multiplication; for the second relation, specify $A$ for $z$ in the routine name to denote addition.

**Note**

Save the contents of vector registers V0, V1, and V2 before you call this routine.

Call this routine to utilize vector hardware when computing the recurrence. As an example, the call from VAX FORTRAN is as follows:

```fortran
CALL MTH$VxFOLRLy_z_V2(N, A(K1), INCA, T)
```

The preceding FORTRAN call replaces the following loop:

```fortran
K1 = ....
DO I = 1, N
  T = (+/-)T (+/*) A(K1+(I-1)*INCA)
ENDDO
```

The arrays used in a FOLR expression must be of the same data type in order to be vectorized and user callable. The MTH$ FOLR routines assume that all of the arrays are of the same data type.

This group of routines, MTH$VxFOLRLy_z_V2 (and also MTH$VxFOLRLy_MA_V5) return only the result of the last iteration of the linear recurrence relation. This is different from the behavior of MTH$VxFOLRy_MA_V15 (and also MTH$VxFOLRy_z_V8), which save the result of each iteration of the linear recurrence relation in an array.

If you specify 0 for the input increment argument (INCA), the input will be treated as a scalar and broadcast to a vector input with all vector elements equal to the scalar value.
Examples

1. 

The following FORTRAN loop computes a linear recurrence.

```
D_FLOAT
INTEGER I, N
REAL*8 A(200), T
T = 78.9847562
N = 20
DO I = 4, N
T = -T * A(I*10)
ENDDO
```

The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
D_FLOAT
INTEGER N
REAL*8 A(200), T
T = 78.9847562
N = 20
T = MTHSVDFOLRLN_M_V2(N-3, A(40), 10, T)
```

2. 

The following FORTRAN loop computes a linear recurrence.

```
D_FLOAT
INTEGER I, N
REAL*8 A(200), T
T = 78.9847562
N = 20
DO I = 4, N
T = T + A(I*10)
ENDDO
```

The following call from FORTRAN to a FOLR routine replaces the preceding loop.

```
D_FLOAT
INTEGER N
REAL*8 A(200), T
T = 78.9847562
N = 20
T = MTHSVDFOLRLP_A_V2(N-3, A(40), 10, T)
```
The following supported MTH$ routines are not included with the routines in the Scalar MTH$ Reference Section because they are rarely used. The majority of these routines serve to satisfy external references when intrinsic functions in FORTRAN and other languages are passed as parameters. Otherwise, the functions are performed by inline code.

Table A–1 lists all of the entry point and argument information for the MTH$ routines not documented in the Scalar MTH$ Reference Section of this manual.

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$ABS</td>
<td>F-floating Absolute Value Routine</td>
</tr>
<tr>
<td>Format: MTH$ABS f-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: floating_point, F_floating, write only, by value</td>
<td></td>
</tr>
<tr>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$DABS</td>
<td>D-floating Absolute Value Routine</td>
</tr>
<tr>
<td>Format: MTH$DABS d-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: floating_point, D_floating, write only, by value</td>
<td></td>
</tr>
<tr>
<td>d-floating: floating_point, D_floating, read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$GABS</td>
<td>G-floating Absolute Value Routine</td>
</tr>
<tr>
<td>Format: MTH$GABS g-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: floating_point, G_floating, write only, by value</td>
<td></td>
</tr>
<tr>
<td>g-floating: floating_point, G_floating, read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$HABS</td>
<td>H-floating Absolute Value Routine</td>
</tr>
<tr>
<td>Format: MTH$HABS h-abs-val, h-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: None</td>
<td></td>
</tr>
<tr>
<td>h-abs-val: floating_point, H_floating, write only, by reference</td>
<td></td>
</tr>
<tr>
<td>h-floating: floating_point, H_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$IIABS</td>
<td>Word Absolute Value Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IIABS word</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>word: word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$JIABS</td>
<td>Longword Absolute Value Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JIABS longword</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword: longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$IIAND</td>
<td>Bitwise AND of Two Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IIAND word1, word2</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_unsigned, word (unsigned), write only, by value</td>
</tr>
<tr>
<td></td>
<td>word1: word_unsigned, word (unsigned), read only, by reference</td>
</tr>
<tr>
<td></td>
<td>word2: word_unsigned, word (unsigned), read only, by reference</td>
</tr>
<tr>
<td>MTH$JIAND</td>
<td>Bitwise AND of Two Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JIAND longword1, longword2</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_unsigned, longword (unsigned), write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword1: longword_unsigned, longword (unsigned), read only, by reference</td>
</tr>
<tr>
<td></td>
<td>longword2: longword_unsigned, longword (unsigned), read only, by reference</td>
</tr>
<tr>
<td>MTH$DBLE</td>
<td>Convert F-floating to D-floating (Exact) Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DBLE f-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$GDBLE</td>
<td>Convert F-floating to G-floating (Exact) Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GDBLE f-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Additional MTHS Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$DIM</td>
<td>Positive Difference of Two F-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DIM f-floating1, f-floating2</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>f-floating1:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>f-floating2:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$DDIM</td>
<td>Positive Difference of Two D-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DDIM d-floating1, d-floating2</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>d-floating1:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>d-floating2:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$GDIM</td>
<td>Positive Difference of Two G-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GDIM g-floating1, g-floating2</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td>g-floating1:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>g-floating2:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$HDIM</td>
<td>Positive Difference of Two H-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HDIM h-floating, h-floating1, h-floating2</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td>h-floating:</td>
<td>floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td>h-floating1:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>h-floating2:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IIDIM</td>
<td>Positive Difference of Two Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IIDIM word1, word2</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td>word1:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>word2:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$JIDIM</td>
<td>Positive Difference of Two Longword Parameters Routine</td>
</tr>
<tr>
<td>Format: MTH$JIDIM longword1, longword2</td>
<td></td>
</tr>
<tr>
<td>Returns: longword_signed, longword (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td>longword1: longword_signed, longword (signed), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>longword2: longword_signed, longword (signed), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$IIEOR</td>
<td>Bitwise Exclusive OR of Two Word Parameters Routine</td>
</tr>
<tr>
<td>Format: MTH$IIEOR word1, word2</td>
<td></td>
</tr>
<tr>
<td>Returns: word_unsigned, word (unsigned), write only, by value</td>
<td></td>
</tr>
<tr>
<td>word1: word_unsigned, word (unsigned), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>word2: word_unsigned, word (unsigned), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$IEOR</td>
<td>Bitwise Exclusive OR of Two Longword Parameters Routine</td>
</tr>
<tr>
<td>Format: MTH$IEOR longword1, longword2</td>
<td></td>
</tr>
<tr>
<td>Returns: longword_unsigned, longword (unsigned), write only, by value</td>
<td></td>
</tr>
<tr>
<td>longword1: longword_unsigned, longword (unsigned), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>longword2: longword_unsigned, longword (unsigned), read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$IIFIX</td>
<td>Convert F-floating to Word (Truncated) Routine</td>
</tr>
<tr>
<td>Format: MTH$IIFIX f-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: word_signed, word (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
<td></td>
</tr>
<tr>
<td>MTH$JIFIX</td>
<td>Convert F-floating to Longword (Truncated) Routine</td>
</tr>
<tr>
<td>Format: MTH$JIFIX f-floating</td>
<td></td>
</tr>
<tr>
<td>Returns: longword_signed, longword (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$FLOATI</td>
<td>Convert Word to F-floating (Exact) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$FLOATI word</td>
</tr>
<tr>
<td></td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>word: word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$DFLOTI</td>
<td>Convert Word to D-floating (Exact) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$DFLOTI word</td>
</tr>
<tr>
<td></td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>word: word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$GFLOTI</td>
<td>Convert Word to G-floating (Exact) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$GFLOTI word</td>
</tr>
<tr>
<td></td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>word: word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$FLOATJ</td>
<td>Convert Longword to F-floating (Rounded) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$FLOATJ longword</td>
</tr>
<tr>
<td></td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword: longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$DFLOTJ</td>
<td>Convert Longword to D-floating (Exact) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$DFLOTJ longword</td>
</tr>
<tr>
<td></td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword: longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$GFLOTJ</td>
<td>Convert Longword to G-floating (Exact) Routine</td>
</tr>
<tr>
<td></td>
<td>MTH$GFLOTJ longword</td>
</tr>
<tr>
<td></td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword: longword_signed, longword (signed), read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
# Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$FLOOR</td>
<td>Convert F-floating to Greatest F-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$FLOOR f-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$FLOOR_R1 f-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_float, write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_float, read only, by reference</td>
</tr>
<tr>
<td>MTH$DFLOOR</td>
<td>Convert D-floating to Greatest D-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DFLOOR d-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$DFLOOR_R3 d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_float, write only, by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_float, read only, by reference</td>
</tr>
<tr>
<td>MTH$GFLOOR</td>
<td>Convert G-floating to Greatest G-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GFLOOR g-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$GFLOOR_R3 g-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_float, write only, by value</td>
</tr>
<tr>
<td>g-floating:</td>
<td>floating_point, G_float, read only, by reference</td>
</tr>
<tr>
<td>MTH$HFLOOR</td>
<td>Convert H-floating to Greatest H-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HFLOOR max-h-float, h-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$HFLOOR_R7 h-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td>max-h-float:</td>
<td>floating_point, H_float, write only, by reference</td>
</tr>
<tr>
<td>h-floating:</td>
<td>floating_point, H_float, read only, by reference</td>
</tr>
<tr>
<td>MTH$AINT</td>
<td>Convert F-floating to Truncated F-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AINT f-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$AINT_R2 f-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_float, write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_float, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MTH$DINT</strong></td>
<td>Convert D-floating to Truncated D-floating Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DINT d-floating</td>
</tr>
<tr>
<td>JSB:</td>
<td>MTH$DINT_R4 d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
</tbody>
</table>

| **MTH$IIDINT** | Convert D-floating to Word (Truncated) Routine |
| Format:       | MTH$IIDINT d-floating |
| Returns:      | word_signed, word (signed), write only, by value |
| d-floating:   | floating_point, D_floating, read only, by reference |

| **MTH$JIDINT** | Convert D-floating to Longword (Truncated) Routine |
| Format:       | MTH$JIDINT d-floating |
| Returns:      | longword_signed, longword (signed), write only, by value |
| d-floating:   | floating_point, D_floating, read only, by reference |

| **MTH$GINT** | Convert G-floating to Truncated G-floating Routine |
| Format:      | MTH$GINT g-floating |
| JSB:         | MTH$GINT_R4 g-floating |
| Returns:     | floating_point, G_floating, write only, by value |
| g-floating:  | floating_point, G_floating, read only, by reference |

| **MTH$IGINT** | Convert G-floating to Word (Truncated) Routine |
| Format:       | MTH$IGINT g-floating |
| Returns:      | word_signed, word (signed), write only, by value |
| g-floating:   | floating_point, G_floating, read only, by reference |

| **MTH$JGINT** | Convert G-floating to Longword (Truncated) Routine |
| Format:       | MTH$JGINT g-floating |
| Returns:      | longword_signed, longword (signed), write only, by value |
| g-floating:   | floating_point, G_floating, read only, by reference |

(continued on next page)
Table A-1 (Cont.)  Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$HINT</td>
<td>Convert H-floating to Truncated H-floating Routine</td>
</tr>
<tr>
<td></td>
<td><strong>Format:</strong> MTH$HINT trunc-h-flt, h-floating</td>
</tr>
<tr>
<td></td>
<td><strong>JSB:</strong> MTH$HINT_R8 h-floating</td>
</tr>
<tr>
<td></td>
<td><strong>Returns:</strong> None</td>
</tr>
<tr>
<td></td>
<td><strong>trunc-h-flt:</strong> floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td></td>
<td><strong>h-floating:</strong> floating_point, H_floating, read only, by reference</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTH$IIHINT</th>
<th>Convert H-floating to Word (Truncated) Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> MTH$IIHINT h-floating</td>
<td></td>
</tr>
<tr>
<td><strong>Returns:</strong> word_signed, word (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td><strong>h-floating:</strong> floating_point, H_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTH$JIHINT</th>
<th>Convert H-floating to Longword (Truncated) Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> MTH$JIHINT h-floating</td>
<td></td>
</tr>
<tr>
<td><strong>Returns:</strong> longword_signed, longword (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td><strong>h-floating:</strong> floating_point, H_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTH$IINT</th>
<th>Convert F-floating to Word (Truncated) Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> MTH$IINT f-floating</td>
<td></td>
</tr>
<tr>
<td><strong>Returns:</strong> word_signed, word (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td><strong>f-floating:</strong> floating_point, F_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTH$JINT</th>
<th>Convert F-floating to Longword (Truncated) Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format:</strong> MTH$JINT f-floating</td>
<td></td>
</tr>
<tr>
<td><strong>Returns:</strong> longword_signed, longword (signed), write only, by value</td>
<td></td>
</tr>
<tr>
<td><strong>f-floating:</strong> floating_point, F_floating, read only, by reference</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$IIOR</td>
<td>Bitwise Inclusive OR of Two Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>( \text{MTH$IIOR \ word}_1, \ \text{word}_2 )</td>
</tr>
<tr>
<td>Returns:</td>
<td>( \text{word} \text{unsigned, word (unsigned), write only, by value} )</td>
</tr>
<tr>
<td></td>
<td>( \text{word}_1 \text{ unsigned, word (unsigned), read only, by reference} )</td>
</tr>
<tr>
<td></td>
<td>( \text{word}_2 \text{ unsigned, word (unsigned), read only, by reference} )</td>
</tr>
<tr>
<td>MTH$JIOR</td>
<td>Bitwise Inclusive OR of Two Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>( \text{MTH$JIOR \ longword}_1, \ \text{longword}_2 )</td>
</tr>
<tr>
<td>Returns:</td>
<td>( \text{longword} \text{unsigned, longword (unsigned), write only, by value} )</td>
</tr>
<tr>
<td></td>
<td>( \text{longword}_1 \text{ unsigned, longword (unsigned), read only, by reference} )</td>
</tr>
<tr>
<td></td>
<td>( \text{longword}_2 \text{ unsigned, longword (unsigned), read only, by reference} )</td>
</tr>
<tr>
<td>MTH$AIMAX0</td>
<td>( F )-floating Maximum of N Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>( \text{MTH$AIMAX0 \ word}, \ldots )</td>
</tr>
<tr>
<td>Returns:</td>
<td>( \text{floating} \text{point, } F_{\text{floating}}, \text{write only, by value} )</td>
</tr>
<tr>
<td></td>
<td>( \text{word} \text{ signed, word (signed), read only, by reference} )</td>
</tr>
<tr>
<td>MTH$AJMAX0</td>
<td>( F )-floating Maximum of N Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>( \text{MTH$AJMAX0 \ longword}, \ldots )</td>
</tr>
<tr>
<td>Returns:</td>
<td>( \text{floating} \text{point, } F_{\text{floating}}, \text{write only, by value} )</td>
</tr>
<tr>
<td></td>
<td>( \text{longword} \text{ signed, longword (signed), read only, by reference} )</td>
</tr>
<tr>
<td>MTH$IMAX0</td>
<td>Word Maximum of N Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>( \text{MTH$IMAX0 \ word}, \ldots )</td>
</tr>
<tr>
<td>Returns:</td>
<td>( \text{word} \text{ signed, word (signed), write only, by value} )</td>
</tr>
<tr>
<td></td>
<td>( \text{word} \text{ signed, word (signed), read only, by reference} ) (continued on next page)</td>
</tr>
</tbody>
</table>
### Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$JMAX0</td>
<td>Longword Maximum of N Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JMAX0 longword, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>longword: longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$AMAX1</td>
<td>F-floating Maximum of N F-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AMAX1 f-floating, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$DMAX1</td>
<td>D-floating Maximum of N D-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DMAX1 d-floating, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>d-floating: floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$GMAX1</td>
<td>G-floating Maximum of N G-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GMAX1 g-floating, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td></td>
<td>g-floating: floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$HMAX1</td>
<td>H-floating Maximum of N H-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HMAX1 h-float-max, h-floating, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>h-float-max: floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td></td>
<td>h-floating: floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IMAX1</td>
<td>Word Maximum of N F-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IMAX1 f-floating, ...</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
# Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$JMAX1</td>
<td>Longword Maximum of N F-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JMAX1 f-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$AIMINO</td>
<td>F-floating Minimum of N Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AIMINO0 word, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>word:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$AJMINO</td>
<td>F-floating Minimum of N Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AJMINO0 longword, . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>longword:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$IMINO0</td>
<td>Word Minimum of N Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IMINO0 word, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td>word:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$JMINO</td>
<td>Longword Minimum of N Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JMINO0 longword, . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td>longword:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$AMIN1</td>
<td>F-floating Minimum of N F-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AMIN1 f-floating, . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$DMIN1</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DMIN1 d-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$GMIN1</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GMIN1 g-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td>g-floating:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$HMIN1</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HMIN1 h-float-max, h-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td>h-float-max:</td>
<td>floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td>h-floating:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IMIN1</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IMIN1 f-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$JMIN1</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JMIN1 f-floating, . . .</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$AMOD</td>
<td></td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$AMOD dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>dividend:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$DMOD</td>
<td>Remainder from Division of Two D-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DMOD dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>dividend:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$GMOD</td>
<td>Remainder from Division of Two G-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GMOD dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td>dividend:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$HMOD</td>
<td>Remainder from Division of Two H-floating Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HMOD h-mod, dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td>h-mod:</td>
<td>floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td>dividend:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IMOD</td>
<td>Remainder from Division of Two Word Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IMOD dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td>dividend:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$JMOD</td>
<td>Remainder of Two Longword Parameters Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JMOD dividend, divisor</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td>dividend:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>divisor:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Additional MTH$ Routines

Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$ANINT</td>
<td>Convert F-floating to Nearest F-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$ANINT f-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only by value</td>
</tr>
<tr>
<td>f-floating:</td>
<td>floating_point, F_floating, read only by reference</td>
</tr>
<tr>
<td>MTH$DNINT</td>
<td>Convert D-floating to Nearest D-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$DNINT d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only by reference</td>
</tr>
<tr>
<td>MTH$IIDNNT</td>
<td>Convert D-floating to Nearest Word Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IIDNNT d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only by reference</td>
</tr>
<tr>
<td>MTH$JIDNNT</td>
<td>Convert D-floating to Nearest Longword Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JIDNNT d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only by reference</td>
</tr>
<tr>
<td>MTH$GNINT</td>
<td>Convert G-floating to Nearest G-floating Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$GNINT g-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only by value</td>
</tr>
<tr>
<td>g-floating:</td>
<td>floating_point, G_floating, read only by reference</td>
</tr>
<tr>
<td>MTH$IIGNNT</td>
<td>Convert G-floating to Nearest Word Integer Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IIGNNT g-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only by value</td>
</tr>
<tr>
<td>g-floating:</td>
<td>floating_point, G_floating, read only by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$JIGNNT</td>
<td>Convert G-floating to Nearest Longword Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$JIGNNT g-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>g-floating: floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$HNINT</td>
<td>Convert H-floating to Nearest H-floating Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$HNINT nearest-h-flt, h-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: None</td>
</tr>
<tr>
<td></td>
<td>nearest-h-flt: floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td></td>
<td>h-floating: floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IIHNNT</td>
<td>Convert H-floating to Nearest Word Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$IIHNNT h-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>h-floating: floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$JIHNNT</td>
<td>Convert H-floating to Nearest Longword Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$JIHNNT h-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>h-floating: floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$ININT</td>
<td>Convert F-floating to Nearest Word Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$ININT f-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$JNINT</td>
<td>Convert F-floating to Nearest Longword Integer Routine</td>
</tr>
<tr>
<td></td>
<td>Format: MTH$JNINT f-floating</td>
</tr>
<tr>
<td></td>
<td>Returns: longword_signed, longword (signed), write only, by value</td>
</tr>
<tr>
<td></td>
<td>f-floating: floating_point, F_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
### Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MTH$INOT</strong></td>
<td><em>Bitwise Complement of Word Parameter Routine</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$INOT word</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>word_unsigned, word (unsigned), write only, by value</code></td>
</tr>
<tr>
<td></td>
<td><code>word: word_unsigned, word (unsigned), read only, by reference</code></td>
</tr>
<tr>
<td><strong>MTH$JNOT</strong></td>
<td><em>Bitwise Complement of Longword Parameter Routine</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$JNOT longword</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>longword_unsigned, longword (unsigned), write only, by value</code></td>
</tr>
<tr>
<td></td>
<td><code>longword: longword_unsigned, longword (unsigned), read only, by reference</code></td>
</tr>
<tr>
<td><strong>MTH$DPROD</strong></td>
<td><em>D-floating Product of Two F-floating Parameters Routine</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$DPROD f-floating1, f-floating2</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>floating_point, D_floating, write only, by value</code></td>
</tr>
<tr>
<td></td>
<td><code>f-floating1: floating_point, F_floating, read only, by reference</code></td>
</tr>
<tr>
<td></td>
<td><code>f-floating2: floating_point, F_floating, read only, by reference</code></td>
</tr>
<tr>
<td><strong>MTH$GPROD</strong></td>
<td><em>G-floating Product of Two F-floating Parameters Routine</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$GPROD f-floating1, f-floating2</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>floating_point, G_floating, write only, by value</code></td>
</tr>
<tr>
<td></td>
<td><code>f-floating1: floating_point, F_floating, read only, by reference</code></td>
</tr>
<tr>
<td></td>
<td><code>f-floating2: floating_point, F_floating, read only, by reference</code></td>
</tr>
<tr>
<td><strong>MTH$SGN</strong></td>
<td><em>F-floating Sign Function</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$SGN f-floating</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>longword_signed, longword (signed), write only, by reference</code></td>
</tr>
<tr>
<td></td>
<td><code>f-floating: floating_point, F_floating, read only, by reference</code></td>
</tr>
<tr>
<td><strong>MTH$SGN</strong></td>
<td><em>D-floating Sign Function</em></td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td><code>MTH$SGN d-floating</code></td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td><code>longword_signed, longword (signed), write only, by reference</code></td>
</tr>
<tr>
<td></td>
<td><code>d-floating: floating_point, D_floating, read only, by reference</code></td>
</tr>
</tbody>
</table>

(continued on next page)
### Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MTH$IIISHFT</strong></td>
<td>Bitwise Shift of Word Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>\texttt{MTH$IIISHFT} word, shift-cnt</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_unsigned, word (unsigned), write only, by value</td>
</tr>
<tr>
<td>word:</td>
<td>word_unsigned, word (unsigned), read only, by reference</td>
</tr>
<tr>
<td>shift-cnt:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td><strong>MTH$JISHFT</strong></td>
<td>Bitwise Shift of Longword Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>\texttt{MTH$JISHFT} longword, shift-cnt</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_unsigned, longword (unsigned), write only, by value</td>
</tr>
<tr>
<td>longword:</td>
<td>longword_unsigned, longword (unsigned), read only, by reference</td>
</tr>
<tr>
<td>shift-cnt:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td><strong>MTH$SIGN</strong></td>
<td>F-floating Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>\texttt{MTH$SIGN} f_float-x, f_float-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>f_float-x:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td>f_float-y:</td>
<td>floating_point, F_floating, read only, by reference</td>
</tr>
<tr>
<td><strong>MTH$DSIGN</strong></td>
<td>D-floating Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>\texttt{MTH$DSIGN} d_float-x, d_float-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, D_floating, write only, by value</td>
</tr>
<tr>
<td>d_float-x:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>d_float-y:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td><strong>MTH$GSIGN</strong></td>
<td>G-floating Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>\texttt{MTH$GSIGN} g_float-x, g_float-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, G_floating, write only, by value</td>
</tr>
<tr>
<td>g_float-x:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
<tr>
<td>g_float-y:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
</tbody>
</table>

(continued on next page)
Table A-1 (Cont.) Additional MTH$ Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Entry Point Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTH$HSIGN</td>
<td>H-floating Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$HSIGN h-result, h-float-x, h-float-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>None</td>
</tr>
<tr>
<td>h-result:</td>
<td>floating_point, H_floating, write only, by reference</td>
</tr>
<tr>
<td>h-float-x:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>h-float-y:</td>
<td>floating_point, H_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$IISIGN</td>
<td>Word Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$IISIGN word-x, word-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>word_signed, word (signed), write only, by value</td>
</tr>
<tr>
<td>word-x:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>word-y:</td>
<td>word_signed, word (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$JISIGN</td>
<td>Longword Transfer of Sign of Y to Sign of X Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$JISIGN longwrd-x, longwrd-y</td>
</tr>
<tr>
<td>Returns:</td>
<td>longword_signed, longword (signed), write only, by reference</td>
</tr>
<tr>
<td>longwrd-x:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>longwrd-y:</td>
<td>longword_signed, longword (signed), read only, by reference</td>
</tr>
<tr>
<td>MTH$SNGL</td>
<td>Convert D-floating to F-floating (Rounded) Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$SNGL d-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>d-floating:</td>
<td>floating_point, D_floating, read only, by reference</td>
</tr>
<tr>
<td>MTH$SNGLG</td>
<td>Convert G-floating to F-floating (Rounded) Routine</td>
</tr>
<tr>
<td>Format:</td>
<td>MTH$SNGLG g-floating</td>
</tr>
<tr>
<td>Returns:</td>
<td>floating_point, F_floating, write only, by value</td>
</tr>
<tr>
<td>g-floating:</td>
<td>floating_point, G_floating, read only, by reference</td>
</tr>
</tbody>
</table>
Table B-1 contains all of the vector MTH$ routines that you can call from
VAX MACRO. Be sure to read Section 2.3.3 and Section 2.3.4 before using the
information in this table.

<table>
<thead>
<tr>
<th>Scalar Name</th>
<th>Call or JSB</th>
<th>Vector Input Registers</th>
<th>Vector Output Registers</th>
<th>Vector Name (Underflows Not Signaled)</th>
<th>Vector Name (Underflows Signaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AINT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VAINT_R0_V1</td>
<td></td>
</tr>
<tr>
<td>DINT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDINT_R3_V3</td>
<td></td>
</tr>
<tr>
<td>GINT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGINT_R3_V3</td>
<td></td>
</tr>
<tr>
<td>DPROD</td>
<td>Call</td>
<td>V0,VI</td>
<td>V0</td>
<td>MTH$VVDPROD_R1_V1</td>
<td></td>
</tr>
<tr>
<td>GPROD</td>
<td>Call</td>
<td>V0,VI</td>
<td>V0</td>
<td>MTH$VVGPROD_R1_V1</td>
<td></td>
</tr>
<tr>
<td>ACOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VACOS_R6_V7</td>
<td></td>
</tr>
<tr>
<td>DACOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDACOS_R2_V7</td>
<td></td>
</tr>
<tr>
<td>GACOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGACOS_R2_V7</td>
<td></td>
</tr>
<tr>
<td>ACOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VACOSD_R6_V7</td>
<td></td>
</tr>
<tr>
<td>DACOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDACOSD_R2_V7</td>
<td></td>
</tr>
<tr>
<td>GACOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGACOSD_R2_V7</td>
<td></td>
</tr>
<tr>
<td>ASIN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VASIN_R2_V6</td>
<td></td>
</tr>
<tr>
<td>DASIN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDASIN_R2_V6</td>
<td></td>
</tr>
<tr>
<td>GASIN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGASIN_R2_V6</td>
<td></td>
</tr>
<tr>
<td>ASIND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VASIND_R2_V6</td>
<td></td>
</tr>
<tr>
<td>DASIND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDASIND_R2_V6</td>
<td></td>
</tr>
<tr>
<td>GASIND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGASIND_R2_V6</td>
<td></td>
</tr>
<tr>
<td>ATAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VATAN_R0_V4</td>
<td></td>
</tr>
<tr>
<td>DATAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDATAN_R0_V6</td>
<td></td>
</tr>
<tr>
<td>GATAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGATAN_R0_V6</td>
<td></td>
</tr>
<tr>
<td>ATAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VATAND_R0_V4</td>
<td></td>
</tr>
<tr>
<td>DATAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDATAND_R0_V6</td>
<td></td>
</tr>
<tr>
<td>GATAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGATAND_R0_V6</td>
<td></td>
</tr>
<tr>
<td>ATAN2</td>
<td>JSB</td>
<td>V0,VI</td>
<td>V0</td>
<td>MTH$VVATAN2_R4_V7</td>
<td></td>
</tr>
<tr>
<td>DATAN2</td>
<td>JSB</td>
<td>V0,VI</td>
<td>V0</td>
<td>MTH$VVDATAN2_R4_V9</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
## Vector MTH$ Routine Entry Points

<table>
<thead>
<tr>
<th>Scalar Name</th>
<th>Call or JSB</th>
<th>Vector Input Registers</th>
<th>Vector Output Registers</th>
<th>Vector Name (Underflows Not Signaled)</th>
<th>Vector Name (Underflows Signaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATAN2</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VGATAN2_R4_V9</td>
<td></td>
</tr>
<tr>
<td>ATAND2</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VATAND2_R4_V7</td>
<td></td>
</tr>
<tr>
<td>DATAND2</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VDATAND2_R4_V9</td>
<td></td>
</tr>
<tr>
<td>GATAND2</td>
<td>JSB</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VGATAND2_R4_V9</td>
<td></td>
</tr>
<tr>
<td>CABS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VCABS_R1_V5</td>
<td></td>
</tr>
<tr>
<td>CDABS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VCDBS_R1_V6</td>
<td></td>
</tr>
<tr>
<td>CGABS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0</td>
<td>MTH$VCGABS_R1_V6</td>
<td></td>
</tr>
<tr>
<td>CCOS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCOS_R4_V7</td>
<td></td>
</tr>
<tr>
<td>CDCOS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VDCOS_R4_V6</td>
<td></td>
</tr>
<tr>
<td>CGCOS</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VGCOS_R1_V11</td>
<td></td>
</tr>
<tr>
<td>COS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VCOS_R4_V7</td>
<td></td>
</tr>
<tr>
<td>DCOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDCOS_R4_V8</td>
<td></td>
</tr>
<tr>
<td>GCOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGCOS_R4_V8</td>
<td></td>
</tr>
<tr>
<td>COSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VCOSD_R4_V6</td>
<td></td>
</tr>
<tr>
<td>DCOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDCOSD_R4_V6</td>
<td></td>
</tr>
<tr>
<td>GCOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGCOSD_R4_V6</td>
<td></td>
</tr>
<tr>
<td>CEXP</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCEXP_R1_V8</td>
<td></td>
</tr>
<tr>
<td>CDEXP</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCDEXP_R1_V10</td>
<td></td>
</tr>
<tr>
<td>CGEXP</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCGEXP_R1_V10</td>
<td></td>
</tr>
<tr>
<td>CLOG</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCLOG_R1_V8</td>
<td></td>
</tr>
<tr>
<td>CDLOG</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCDLOG_R1_V10</td>
<td></td>
</tr>
<tr>
<td>CGLOG</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCGL0G_R1_V10</td>
<td></td>
</tr>
<tr>
<td>AMOD</td>
<td>JSB</td>
<td>V0, R0</td>
<td>V0</td>
<td>MTH$VAMOD_R4_V5</td>
<td></td>
</tr>
<tr>
<td>DMOD</td>
<td>JSB</td>
<td>V0, R0</td>
<td>V0</td>
<td>MTH$VDMOD_R7_V6</td>
<td></td>
</tr>
<tr>
<td>GMOD</td>
<td>JSB</td>
<td>V0, R0</td>
<td>V0</td>
<td>MTH$VGMOD_R7_V6</td>
<td></td>
</tr>
<tr>
<td>CSIN</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCSIN_R1_V11</td>
<td></td>
</tr>
<tr>
<td>CDSIN</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VDSIN_R1_V11</td>
<td></td>
</tr>
<tr>
<td>CGSIN</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCGSIN_R1_V11</td>
<td></td>
</tr>
<tr>
<td>CQSRT</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCSQRT_R1_V7</td>
<td></td>
</tr>
<tr>
<td>CDQRT</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VDCQRT_R1_V8</td>
<td></td>
</tr>
<tr>
<td>CGSQRT</td>
<td>Call</td>
<td>V0, V1</td>
<td>V0, V1</td>
<td>MTH$VCSQRT_R1_V8</td>
<td></td>
</tr>
<tr>
<td>COSH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VCOSH_R5_V8</td>
<td></td>
</tr>
<tr>
<td>DCOSH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDCOSH_R5_V8</td>
<td></td>
</tr>
<tr>
<td>GCOSH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGCOSH_R5_V8</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VEXP_R3_V6</td>
<td></td>
</tr>
<tr>
<td>DEXP</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDEXP_R3_V6</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
Table B-1 (Cont.) Vector MTH$ Routines

<table>
<thead>
<tr>
<th>Scalar Name</th>
<th>Call or JSB</th>
<th>Vector Input Registers</th>
<th>Vector Output Registers</th>
<th>Vector Name (Underflows Not Signaled)</th>
<th>Vector Name (Underflows Signaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEXP</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGEXP_R3_V6</td>
<td>MTH$VGEXP_E_R3_V6</td>
</tr>
<tr>
<td>ALOG</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VALOG_R3_V5</td>
<td></td>
</tr>
<tr>
<td>DLOG</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDLOG_R3_V7</td>
<td></td>
</tr>
<tr>
<td>GLOG</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGLOG_R3_V7</td>
<td></td>
</tr>
<tr>
<td>ALOG10</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VALOG10_R3_V5</td>
<td></td>
</tr>
<tr>
<td>DLOG10</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDLOG10_R3_V7</td>
<td></td>
</tr>
<tr>
<td>GLOG10</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGLOG10_R3_V7</td>
<td></td>
</tr>
<tr>
<td>ALOG2</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VALOG2_R3_V5</td>
<td></td>
</tr>
<tr>
<td>DLOG2</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDLOG2_R3_V7</td>
<td></td>
</tr>
<tr>
<td>GLOG2</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGLOG2_R3_V7</td>
<td></td>
</tr>
<tr>
<td>RANDOM</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VRANDOM_R2_V0</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VSIN_R4_V6</td>
<td></td>
</tr>
<tr>
<td>DSI</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDSDI_R4_V6</td>
<td></td>
</tr>
<tr>
<td>GSIN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGSID_R4_V6</td>
<td></td>
</tr>
<tr>
<td>SIND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VSIND_R4_V6</td>
<td></td>
</tr>
<tr>
<td>DSIIND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDSDIIND_R4_V6</td>
<td>MTH$VDSDIIND_E_R6_V6</td>
</tr>
<tr>
<td>GSININD</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGSIDIND_R4_V6</td>
<td>MTH$VGSIDIND_E_R6_V6</td>
</tr>
<tr>
<td>SINCOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VSINCOS_R4_V7</td>
<td>MTH$VSINCOS_E_R6_V7</td>
</tr>
<tr>
<td>DSIINCOS</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VDSIINCOS_R4_V8</td>
<td></td>
</tr>
<tr>
<td>GSINCO</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VGSINCO_R4_V8</td>
<td></td>
</tr>
<tr>
<td>SINCOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VSINCOSD_R4_V6</td>
<td>MTH$VSINCOSD_E_R6_V6</td>
</tr>
<tr>
<td>DSIINCOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VDSIINCOSD_R4_V7</td>
<td>MTH$VDSIINCOSD_E_R6_V7</td>
</tr>
<tr>
<td>GSINCO</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VGSINCO_R4_V8</td>
<td></td>
</tr>
<tr>
<td>SINCOSD</td>
<td>JSB</td>
<td>V0</td>
<td>V0,V1</td>
<td>MTH$VSINCOSD_R4_V7</td>
<td>MTH$VSINCOSD_E_R6_V7</td>
</tr>
<tr>
<td>SINH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VSINH_R5_V9</td>
<td>MTH$VSINH_E_R5_V9</td>
</tr>
<tr>
<td>DSIINH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDSDIINH_R5_V9</td>
<td></td>
</tr>
<tr>
<td>GSINH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGSINH_R5_V9</td>
<td></td>
</tr>
<tr>
<td>SQRT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VSQRT_R2_V4</td>
<td>MTH$VSQRT_E_R2_V4</td>
</tr>
<tr>
<td>D SQRT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDSQRT_R2_V5</td>
<td></td>
</tr>
<tr>
<td>GSQRT</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VSQRT_R2_V5</td>
<td></td>
</tr>
<tr>
<td>TAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VTAN_R4_V5</td>
<td>MTH$VTAN_E_R4_V5</td>
</tr>
<tr>
<td>DTAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDTAN_R4_V5</td>
<td></td>
</tr>
<tr>
<td>GTAN</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGTA N_R4_V5</td>
<td>MTH$VGTA N_E_R4_V5</td>
</tr>
<tr>
<td>TAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VTAND_R4_V5</td>
<td>MTH$VTAND_E_R4_V5</td>
</tr>
<tr>
<td>DTAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDTAND_R4_V5</td>
<td>MTH$VDTAND_E_R4_V5</td>
</tr>
<tr>
<td>GTAND</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGTA N_R4_V5</td>
<td>MTH$VGTA N_E_R4_V5</td>
</tr>
<tr>
<td>TANH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VTANH_R3_V10</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
Vector MTH$ Routine Entry Points

Table B-1 (Cont.) Vector MTH$ Routines

<table>
<thead>
<tr>
<th>Scalar Name</th>
<th>Call or JSB</th>
<th>Vector Input Registers</th>
<th>Vector Output Registers</th>
<th>Vector Name (Underflows Not Signaled)</th>
<th>Vector Name (Underflows Signaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTANH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VDTANH_R3_V10</td>
<td></td>
</tr>
<tr>
<td>GTANH</td>
<td>JSB</td>
<td>V0</td>
<td>V0</td>
<td>MTH$VGTANH_R3_V10</td>
<td></td>
</tr>
<tr>
<td>DIVC</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VDIVC_R1_V6</td>
<td>OTS$VDIVC_E_R1_V6</td>
</tr>
<tr>
<td>DIVCD</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VDIVCD_R1_V7</td>
<td>OTS$VDIVCD_E_R1_V7</td>
</tr>
<tr>
<td>DIVCG</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VDIVCG_R1_V7</td>
<td>OTS$VDIVCG_E_R1_V7</td>
</tr>
<tr>
<td>MULC</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VMULC_R1_V4</td>
<td>OTS$VMULC_E_R1_V4</td>
</tr>
<tr>
<td>MULCD</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VMULCD_R1_V4</td>
<td>OTS$VMULCD_E_R1_V4</td>
</tr>
<tr>
<td>MULCG</td>
<td>Call</td>
<td>V0,V1,V2,V3</td>
<td>V0,V1</td>
<td>OTS$VMULCG_R1_V4</td>
<td>OTS$VMULCG_E_R1_V4</td>
</tr>
<tr>
<td>POWJJ</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWJJ_R1_V1</td>
<td>OTS$VPOWJJ_E_R1_V1</td>
</tr>
<tr>
<td>POWRJ</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWRJ_R1_V2</td>
<td>OTS$VPOWRJ_E_R1_V2</td>
</tr>
<tr>
<td>POWDJ</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWDJ_R1_V2</td>
<td>OTS$VPOWDJ_E_R1_V2</td>
</tr>
<tr>
<td>POWGJ</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWGJ_R1_V2</td>
<td>OTS$VPOWGJ_E_R1_V2</td>
</tr>
<tr>
<td>POWRR</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWRR_R1_V4</td>
<td>OTS$VPOWRR_E_R1_V4</td>
</tr>
<tr>
<td>POWDD</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWDD_R1_V8</td>
<td>OTS$VPOWDD_E_R1_V8</td>
</tr>
<tr>
<td>POWGG</td>
<td>Call</td>
<td>V0,R0</td>
<td>V0</td>
<td>OTS$VPOWGG_R1_V9</td>
<td>OTS$VPOWGG_E_R1_V9</td>
</tr>
</tbody>
</table>
Index

A
Absolute value, 1–4
  of complex number, MTH-23
Additional routines
  list of, 1–4 to 1–9
Algorithm, 1–3
Arc cosine
  in degrees, MTH-6, MTH-70
  in radians, MTH-3, MTH-68
Arc sine
  in degrees, MTH-11, MTH-74
  in radians, MTH-9, MTH-72
Arc tangent
  hyperbolic, MTH-21, MTH-84
  in degrees, MTH-15, MTH-19, MTH-78,
  MTH-82
  in radians, MTH-13, MTH-17, MTH-76,
  MTH-80
Arrays
  conversion of, MTH-63

B
Backward indexing, 2–6
Bitwise AND operator, 1–4
Bitwise complement operator, 1–8
Bitwise exclusive OR operator, 1–5
Bitwise inclusive OR operator, 1–6
Bitwise shift, 1–8
BLAS
  definition of, 2–1
BLAS Level 1
  BLAS1$VxAMAX, MTH–149
  BLAS1$VxASUM, MTH–152
  BLAS1$VxAXPY, MTH–155
  BLAS1$VxCOPY, MTH–160
  BLAS1$VxDOT, MTH–165
  BLAS1$VxNRM2, MTH–170
  BLAS1$VxROT, MTH–173
  BLAS1$VxROTG, MTH–178
  BLAS1$VxSCAL, MTH–182
  BLAS1$VxSWAP, MTH–186

C
Calling convention, 1–2
Complex numbers, 1–3, MTH–56, MTH–58,
  MTH–111, MTH–121
  absolute value of, MTH–23
  complex exponential of, MTH–30, MTH–32
  conjugate of, MTH–43, MTH–44
  cosine of, MTH–26, MTH–28
  made from floating-point, MTH–39, MTH–41
  natural logarithm of, MTH–34, MTH–36
  sine of, MTH–52, MTH–53
Condition handling, 1–3
Conjugate of complex number, MTH–43, MTH–44
Conversion of double to single floating-point value, 1–9
Conversion to greatest floating-point integer, 1–5
Copying
  vector, MTH–160
Cosind
  in radians, MTH–125
Cosine
  hyperbolic, MTH–50, MTH–88
  in degrees, MTH–48, MTH–87, MTH–128
  in radians, MTH–46, MTH–86
  of complex number, MTH–26, MTH–28

D
Double-precision value
  converting, MTH–61
  converting an array of, MTH–63

E
Entry point name, 1–1
Error checking
  in FOLR routines, 2–6
Euclidean norm
  of a vector, MTH–170
Exceptions
  recovering from, 2–7
Exponential, MTH–65, MTH–90
  of complex number, MTH–30, MTH–32
F-floating conversion, 1-4
First Order Linear Recurrence, MTH-190,
MTH-194, MTH-198, MTH-202
Floating-point
conversion to nearest value, 1-7
multiplication, 1-8
positive difference, 1-5
sign function, 1-8
POLR
definition of, 2-6
POLR routines, MTH-190, MTH-194, MTH-198,
MTH-202
error checking, 2-6
naming conventions, 2-6
FORTRAN
/BLAS qualifier, 2-2
Forward indexing, 2-6
G
Givens plane rotation
applying to a vector, MTH-173
generating the elements for, MTH-178
H
Hyperbolic arc tangent, MTH-21, MTH-84
Hyperbolic cosine, MTH-50, MTH-88
Hyperbolic sine, MTH-101, MTH-133
Hyperbolic tangent, MTH-109, MTH-142
I
Index
of a vector, MTH-149
Indexing
backward, 2-6
forward, 2-6
Inner product
of a vector, MTH-165
Integer to floating-point conversion, 1-5
J
JSB entry point, 1-2
L
Linear recurrence
definition of, 2-6
Logarithm
base 2, MTH-94, MTH-115
common, MTH-96, MTH-117
natural, MTH-92, MTH-113
natural complex, MTH-34, MTH-36
M
Mathematics routine
additional routines, A-1 to A-18
Maximum value, 1-6
Minimum value, 1-7
MTH$ACOS, MTH-3
MTH$ACOSD, MTH-6
MTH$AIMAG, MTH-111
MTH$ALOG, MTH-113
MTH$ALOG10, MTH-117
MTH$ALOG2, MTH-115
MTH$ASIN, MTH-9
MTH$ASIND, MTH-11
MTH$ATAN, MTH-13
MTH$ATAND, MTH-15
MTH$ATAND2, MTH-19
MTH$ATANH, MTH-21
MTH$CABS, MTH-23
MTH$CCOS, MTH-26
MTH$CDABS, MTH-23
MTH$CDCOS, MTH-28
MTH$CDEXP, MTH-32
MTH$CDLOG, MTH-36
MTH$CDSIN, MTH-53
MTH$CDSQR, MTH-58
MTH$CEXP, MTH-30
MTH$CGABS, MTH-23
MTH$CGCOS, MTH-28
MTH$CGEXP, MTH-32
MTH$CGLOG, MTH-36
MTH$CGSIN, MTH-53
MTH$CGSQR, MTH-58
MTH$CLOG, MTH-34
MTH$CMPLX, MTH-39
MTH$CONJG, MTH-43
MTH$COS, MTH-46
MTH$COSD, MTH-48
MTH$COSH, MTH-50
MTH$CSIN, MTH-52
MTH$CSQR, MTH-56
MTH$CVT_DA_GA, MTH-63
MTH$CVT_D_G, MTH-61
MTH$CVT_GA_DA, MTH-63
MTH$CVT_G_D, MTH-61
MTH$DACOS, MTH-3
MTH$DACOSD, MTH-6
MTH$DASIN, MTH-9
MTH$DASIND, MTH-11
MTH$DATAN, MTH-13
MTH$DATAND, MTH-17
MTH$DATAND2, MTH-19
MTH$DATANH, MTH-21
Multiplying
vector, MTH-155

N
Naming conventions
FOLR routines, 2-6
vector routines, 2-8
Norm
Euclidean
of a vector, MTH-170

O
Overflow detection, 2-8
Plane rotation
applying Givens plane rotation to a vector, MTH-173
generating the elements for a Givens plane rotation, MTH-178
Product
of a vector, MTH-165

Random number generator, MTH-119
Recurrence
linear
definition of, 2–6
Remainder, 1–7
Rotation
applying to a vector, MTH-173

Scaling
vector, MTH-182
Sine
hyperbolic, MTH-101, MTH-133
in degrees, MTH-99, MTH-128, MTH-131
in radians, MTH-98, MTH-123, MTH-125
of complex number, MTH-52, MTH-53
Square root, MTH-103, MTH-136
Sum of absolute values
of a vector, MTH-152
Swapping
vector, MTH-186

Tangent, MTH-105, MTH-107, MTH-138,
MTH-140
hyperbolic, MTH-109, MTH-142
Truncation of floating-point value, 1–6

Underflow detection, 2–8

VAX FORTRAN
/BLAS qualifier, 2–2
VAX FORTRAN-HPO compiler, 2–2, 2–9
Vectorization of a loop
preventing, MTH-190, MTH-194, MTH-198,
MTH-202
Vectorizing FORTRAN compiler, 2–7
NOTES
How to Order Additional Documentation

Technical Support
If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825) and press 2 for technical assistance.

Electronic Orders
If you wish to place an order through your account at the Electronic Store, dial 800-234-1998, using a modem set to 2400- or 9600-baud. You must be using a VT terminal or terminal emulator set at 8 bits, no parity. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825) and ask for an Electronic Store specialist.

Telephone and Direct Mail Orders

<table>
<thead>
<tr>
<th>From</th>
<th>Call</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>U.S.A.</td>
<td>DECdirect</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td></td>
<td>Phone: 800-DIGITAL</td>
<td>P.O. Box CS2008</td>
</tr>
<tr>
<td></td>
<td>(800-344-4825)</td>
<td>Nashua, NH 03061</td>
</tr>
<tr>
<td></td>
<td>FAX: (603) 884-5597</td>
<td></td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>Phone: (809) 781-0505</td>
<td>Digital Equipment Caribbean, Inc.</td>
</tr>
<tr>
<td></td>
<td>FAX: (809) 749-8377</td>
<td>3 Digital Plaza, 1st Street</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suite 200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Metro Office Park</td>
</tr>
<tr>
<td></td>
<td></td>
<td>San Juan, Puerto Rico 00920</td>
</tr>
<tr>
<td>Canada</td>
<td>Phone: 800-267-6215</td>
<td>Digital Equipment of Canada Ltd.</td>
</tr>
<tr>
<td></td>
<td>FAX: (613) 592-1946</td>
<td>100 Herzberg Road</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kanata, Ontario, Canada K2K 2A6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Attn: DECdirect Sales</td>
</tr>
<tr>
<td>International</td>
<td></td>
<td>Local Digital subsidiary or approved distributor</td>
</tr>
<tr>
<td>Internal Orders</td>
<td>DTN: 241-3023</td>
<td>Software Supply Business (SSB)</td>
</tr>
<tr>
<td></td>
<td>(508) 874-3023</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Digital Drive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Westminster, MA 01473</td>
</tr>
<tr>
<td>Internal Orders</td>
<td>DTN: 234-4325</td>
<td>Publishing &amp; Circulation Services</td>
</tr>
<tr>
<td></td>
<td>(508) 351-4325</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td></td>
<td>FAX: (508) 351-4467</td>
<td>NR02-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>444 Whitney Street</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Northboro, MA 01532</td>
</tr>
</tbody>
</table>

¹Call to request an Internal Software Order Form (EN-01740-07).
How to Create a Professional Resume

Technical Skills

Software Proficiency:

- Microsoft Office Suite
- Adobe Creative Suite
- Google Analytics
- SQL
- Python

Equipment Knowledge:

- Digital Cameras
- Video Editing Tools
- Audio Recording Equipment
- 3D Modeling Software

Education:

- Bachelor of Science in Computer Science
- Master of Business Administration

Certifications:

- Cisco Certified Network Associate (CCNA)
- Certified Public Accountant (CPA)
- Project Management Professional (PMP)

Languages:

- Fluent in English
- Proficient in Spanish
- Basic knowledge of French

Professional Experience:

[Job Title] at [Company Name] (2015 - Present)

- Managed a team of 10 employees
- Increased sales by 25%

[Job Title] at [Company Name] (2010 - 2015)

- Coordinated projects across multiple departments
- Led a successful product launch

References:

Available upon request.
Reader's Comments

Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:  

Accuracy (product works as manual says)  □  □  □  □  □
Completeness (enough information)  □  □  □  □  □
Clarity (easy to understand)  □  □  □  □  □
Organization (structure of subject matter)  □  □  □  □  □
Figures (useful)  □  □  □  □  □
Examples (useful)  □  □  □  □  □
Index (ability to find topic)  □  □  □  □  □
Page layout (easy to find information)  □  □  □  □  □

I would like to see more/less

What I like best about this manual is

What I like least about this manual is

I found the following errors in this manual:

Page Description

Additional comments or suggestions to improve this manual:

For software manuals, please indicate which version of the software you are using:

Name/Title ___________________________ Dept. __________
Company ___________________________ Date __________
Mailing Address ___________________________ Phone ___________________________
Your comments and suggestions help us improve the quality of our publications.

Thank you for your assistance.

I rate this manual's:

<table>
<thead>
<tr>
<th></th>
<th>Excellent</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (product works as manual says)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Completeness (enough information)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Clarity (easy to understand)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Organization (structure of subject matter)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Figures (useful)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Examples (useful)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Index (ability to find topic)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Page layout (easy to find information)</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

I would like to see more/less

What I like best about this manual is

What I like least about this manual is

I found the following errors in this manual:

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additional comments or suggestions to improve this manual:

For software manuals, please indicate which version of the software you are using:

Name/Title ___________________________________________ Dept. __________

Company ____________________________________________ Date ____________

Mailing Address ______________________________________ Phone __________