intel

i860™ Microprocessor
Math Library
Reference Manual

Version 2

January 1990

464411-002

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, BITBUS, COMMputer, CREDIT, Data Pipeline, DVI, ETOX, FaxBACK, Genius, i, $\overline{\text{i}}$, i486, i750, i860, ICE, iCEL, ICEVIEW, iCS, iDBP, iDIS, I^2ICE, iLBX, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, Intel386, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPAT, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, Pro750, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, ToolTALK, UPI, Visual Edge, VLSiCEL, and ZapCode, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

MULTIBUS is a patented Intel bus.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

UNIX is a trademark of AT&T Bell Labs.

MS-DOS and XENIX are a trademarks of Microsoft Corporation.

OS/2 is a trademark of International Business Machines Corporation.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

CG-010290

Preface

The Intel i860™ Microprocessor delivers supercomputing performance in a single VLSI component. The 64-bit design of the i860 Microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches, million-transistor design, and fast one-micron CHMOS IV silicon technology.

The Math Library is a package of procedures (called "primitives") designed for use when writing programs for the i860 Microprocessor that use high-level mathematical functions.

This manual is a complete source of information about the use of the Math Library. It is useful both to software managers and software engineers. Software managers will learn how this product can accelerate development of applications for the i860 Microprocessor. Software engineers will find detailed descriptions of the use of the product.

How to Use This Manual

- Chapter 1, "Introduction" presents an overview of the Math Library and defines the notation of Chapter 2. Read this chapter for information common to all primitives.
- Chapter 2, "Math Primitives" specifies each of the primitives.

Related Documentation

The following literature contains additional information concerning the i860 Microprocessor:

- *i860 64-Bit Microprocessor* (Data Sheet), order number 240296
- *i860 64-Bit Microprocessor Programmer's Reference Manual,* order number 240329
- *i860 64-Bit Microprocessor Simulator and Debugger Reference Manual,* order number 240437
- *i860 64-Bit Microprocessor Assembler and Linker Reference Manual,* order number 240436

Notation and Conventions

italic	In mathematical notation, scalar variables are printed in roman italic type; for example, *scalar*.
roman bold	In mathematical notation, vectors are identified by roman bold type; for example, **X**, **Z**(*i*).
gothic bold	When a name, symbol, or other sequence of characters is used in exactly the same way that it is intended to be entered into the Assembler, a compiler, or other software product, it is printed in a contrasting type style; for example, **r20**.
gothic italic	Gothic italic type indicates a metasymbol that is to be replaced with an item that fulfills the rules for that symbol.
[ ]	Brackets indicate optional arguments or parameters.
[ ]	Brackets, when printed in the contrasting type style, are required, and must be entered as shown.
{ }	One and only one of the enclosed entries must be selected unless the field is also surrounded by brackets, in which case it is optional.
{ }...	At least one of the enclosed entries must be selected unless the field is also surrounded by brackets, in which case it is optional. The items may be used in any order, unless otherwise noted.
...	Ellipses indicate that the preceding argument or parameter may be repeated. When an ellipsis follows a right bracket or brace, the entire unit enclosed within the pair of brackets or braces may be repeated.
←	The notation $a \leftarrow b$ means that the value of b is assigned to a.
SMALLCAPS	Small capital letters indicate the ASCII name of one of the nondisplayable characters; for example, LF is the linefeed character.
.,/$	Punctuation other than brackets must be entered as shown.
π	pi = 3.141592654...

Table of Contents

Tables

Chapter 1
Introduction

1.1 Descriptions of Primitives

Chapter 2 presents detailed descriptions of each primitive. Each description is divided into sections as follows.

1.1.1 Description of Algorithm

With every primitive is given a brief description of the operation performed. The description uses standard mathematical notation, with the symbol ← indicating assignment.

1.1.2 Syntax

Primitives are defined in both FORTRAN and C syntax; however, because they adhere to standard calling conventions, all primitives are callable by other high-level languages as well as by the i860 Microprocessor Assembler. Parameters (except complex) are passed by value. Complex parameters are passed by reference.

The keyword complex is used as a type in the C examples for complex primitives. Because C does not have complex as a primitive type, the following **typedef** is assumed:

```
typedef struct {float re, im} complex;
```

...where **re** is the real part and **im** is the imaginary part[1].

In keeping with this definition of complex variables, when it is necessary to refer to the real and imaginary parts separately, the suffixes **.re** and **.im** are used. For example: **x.re** refers to the real part of the variable **x, x.im** refers to the imaginary part of the variable **x**, and **x = (x.re,x.im).**

Assembly-language programmers can derive register usage from the high-level language syntax given. All parameter passing conforms to the standards specified in *i860™ 64-Bit Microprocessor Programmer's Reference Manual*. Single-precision primitives return values in **f16**; double-precision primitives return values in **f17:f16**; complex primitives leave results in a location specified by an address in **r16**.

1.1.3 Input

The syntax section defines the number and type of input parameters; however, any additional information about the inputs (units and valid range of values, for example) is presented in this section. All angles are expressed in radians.

In order to achieve maximum performance, inputs are not checked for validity. Invalid inputs may produce exceptions, in which case the results depend on the exception handler used.

1.1.4 Output

The syntax section defines the type of the output; however, any additional information about the output (range of possible values, for example) is presented in this section. All angles are expressed in radians.

1. For programming in C, an include file fmath.h is provided containing this typedef declaration.

1.1.5 Precision

Precision is specified as the number of significant decimal digits.

1.1.6 Timing Estimates

All clock counts shown are approximate timing estimates and are subject to change.

1.2 Primitives by Function

Table 1-1 provides a cross-reference of primitives organized by function. Use this table to find the identifier of the primitive that satisfies the required function and type of operands. Then look up the primitive in Chapter 2, which is arranged in alphabetical order by primitive identifier.

Table 1-1. Primitives by Function

Function	Type of Arguments		
	Single	Double	Complex
Square Root	sqrt	dsqrt	csqrt
Cosine	cos	dcos	ccos
Sine	sin	dsin	csin
Tangent	tan	dtan	
Sine and Cosine	sincos	dsincos	
Arc Cosine	acos	dacos	
Arc Sin	asin	dasin	
Arc Tangent	atan	datan	
Arc Tangent of Quotient	atan2	datan2	
Hyperbolic Cosine	cosh	dcosh	
Hyperbolic Sine	sinh	dsinh	
Hyperbolic Tangent	tanh	dtanh	
Exponential	exp	dexp	cexp
Natural Logarithm	alog	dlog	clog
Log Base 10	alog10	dlog10	

1.3 Linking to the Math Library

The name of the library file is **libfm.a**. When programming in the C Language link this library file in the following order:

```
libfm.a {C Libraries}
```

When programming in the Fortran Language link it in the following order:

```
libfm.a {Fortran Libraries}
```

Chapter 2
Math Primitives

2.1 acos ..... Inverse Cosine

Computes...

$$y \leftarrow \text{Arc cos } x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = acos*f* (x);	REAL x, y y = acos (x)

Input $-1 \leq x \leq 1$; zeros and denormals return $\pi/2$; infinities are invalid.

Output $0 \leq y \leq \pi$

Precision 6.52 digits

Timing 63, 111 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.2 alog ..... Natural Logarithm

Computes...

$$y \leftarrow \log_e x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = logf (x);	REAL x, y y = log (x)

Input $x > 0.$

Output $-\infty < y < +\infty$

Precision 6.71 digits

Timing 76 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.3 alog10 ..... Log Base 10

Computes...

$$y \leftarrow \log_{10} x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = log10*f* (x);	REAL x, y y = log10 (x)

Input $x > 0.$

Output $-\infty < y < +\infty$

Precision 7.07 digits

Timing 76 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.4 asin ..... Inverse Sine

Computes...

$$y \leftarrow \text{Arc sin } x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = asin*f* (x);	REAL x, y y = asin (x)

Input $-1 \leq x \leq 1$; zeros and denormals return $\pi/2$; infinities are invalid.

Output $-\pi/2 \leq y \leq \pi/2$

Precision 6.72 digits

Timing 60, 108 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.5 atan .....Inverse Tangent

Computes...

$$y \leftarrow \text{Arc tan } x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = atan*f* (x);	REAL x, y y = atan (x)

Input $-\infty \leq x \leq +\infty$

Output $-\pi/2 \leq y \leq \pi/2$; $-\infty$ returns $-\pi/2$

Precision 6.94 digits

Timing 57, 85 clocks

Exceptions None.

2.6 atan2 .....Inverse Tangent of Quotient

Computes...

$$z \leftarrow \text{Arc tan}(x/y)$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y, z; z = atan2*f* (x, y);	REAL x, y, z z = atan2 (x, y)

Input x and y not both 0.

Output In the range of π (points just below the negative x-axis) to π (points at or just above the negative x-axis). The range of the result depends on the relationship between the operands according to Table 2-1.

Precision 6.94 digits

Timing 11, 12, 101, 130 clocks

Table 2-1. Results of atan2

Sign(y)	Sign(x)	|y| < |x|?	Final Result
+	+	Yes	$0 \le \text{atan}(y/x) \le \pi/4$
+	+	No	$\pi/4 \le \text{atan}(y/x) \le \pi/2$
+	−	No	$\pi/2 \le \text{atan}(y/x) \le 3^*\pi/4$
+	−	Yes	$3^*\pi/4 < \text{atan}(y/x) < \pi$
−	+	Yes	$-\pi/4 \le \text{atan}(y/x) \le 0$
−	+	No	$-\pi/2 \le \text{atan}(y/x) \le -\pi/4$
−	−	No	$-3^*\pi/4 \le \text{atan}(y/x) \le -\pi/2$
−	−	Yes	$-\pi \le \text{atan}(y/x) \le -3^*\pi/4$

Exceptions

Type	Condition	Masked Response
I	$x = \pm\infty$ or $y = \pm\infty$	QNaN *indefinite*
X	$x = y = 0$	QNaN*indefinite*
I	invalid inputs	QNaN *indefinite*

2.7 ccos ..... Complex Cosine

Computes...

$$y \leftarrow \cos x$$

...where x and y are complex numbers.

$$y.re = \cos(x.re) * \cosh(x.im)$$
$$y.im = -\sin(x.re) * \sinh(x.im)$$

Syntax

C	FORTRAN
complex x, y; y = c_cos (x);	COMPLEX x, y y = ccos (x)

Input $-\infty < x.re < +\infty$; $-89.415985 \leq x.im \leq +89.415985$

Output

Precision 7.00 digits for the real part, 7.00 digits for the imaginary part

Timing 231–285 clocks

Exceptions

Type	Condition	Masked Response
O	$\|x.im\| > 89.415985$	$y.re = \pm\infty$, $y.im = \pm\infty$
I	invalid inputs	QNaN*indefinite*

2.8 cexp ..... Complex Exponential

Computes...

$$y \leftarrow e^x$$

...where x and y are complex numbers.

$$y.re = \exp(x.re) * \cos(x.im)$$
$$y.im = \exp(x.re) * \sin(x.im)$$

Syntax

C	FORTRAN
complex x, y; y = c_exp (x);	COMPLEX x, y y = CALL cexp (x)

Input $-87.336545 < x.re < 88.722839$; $-\infty < x.im < +\infty$

Output

Precision 6.70 digits

Timing 130–150 clocks

Exceptions

Type	Condition	Masked Response
O	$x.re > 88.722839$	$+\infty$
U	$x.re < -87.336544$	zero
I	invalid inputs	QNaN *indefinite*

2.9 clog ..... Complex Natural Logarithm

Computes...

$$y \leftarrow \log_e x$$

...where x and y are complex numbers.

$$y.re = 0.5 * \text{alog}(x.re^2 + x.im^2)$$
$$y.im = \text{atan2}(x.im, x.re)$$

Syntax

C	FORTRAN
complex x, y; y = c_log (x);	COMPLEX x, y y = clog (x)

Input $x.re \neq 0$; $x.im \neq 0$; $x.re$ and $x.im$ not both zero

Output

Precision 7.00 digits for the imaginary part; 6.00 digits for the real part

Timing 187–216 clocks

Exceptions Trap if $x.re \neq 0$ or $x.im \neq 0$. ***{What kind of}*** trap if $x.re = x.im = 0$.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.10 cos .....Cosine

Computes...

$$y \leftarrow \cos x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = cos*f* (x);	REAL x, y y = cos (x)

Input Infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 6.92 digits

Timing 30, 55, 58 clocks

Exceptions Trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.11 cosh ..... Hyperbolic Cosine

Computes...

$$y \leftarrow \cosh x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = cosh*f* (x);	REAL x, y y = cosh (x)

Input $-89.415985 < x < 89.415985$

Output

Precision 6.60 digits

Timing 96 clocks

Exceptions

Type	Condition	Masked Response
O	$\|x\| \geq 89.4159865$	$+\infty$
I	invalid inputs	QNaN*indefinite*

2.12 csin .....Complex Sine

Computes...

$$y \leftarrow \sin x$$

...where x and y are complex numbers.

$$y.re = \sin(x.re) * \cosh(x.im)$$
$$y.im = \cos(x.re) * \sinh(x.im)$$

Syntax

C	FORTRAN
complex x, y; y = c_sin (x);	COMPLEX x, y y = csin (x)

Input $-\infty < x.re < +\infty$; $-89.415985 \leq x.im \leq +89.415985$

Output

Precision 7.00 digits for the real part; 7.00 digits for the imaginary part

Timing 131–285 clocks

Exceptions

Type	Condition	Masked Response
O	$\|x.im\| > 89.415985$	$y.re = \pm\infty$, $y.im = \pm\infty$
I	invalid inputs	QNaN *indefinite*

2.13 csqrt ..... Complex Square Root

Computes...

$$y \leftarrow \text{SQRT}(x)$$

...where x and y are complex numbers.

Let...

$$r = \text{sqrt}(x.re^2 + x.im^2)$$
$$t = \text{atan2}(x.im, x.re)*0.5$$

Then...

$$y.re = r * \cos(t)$$
$$y.im = r * \sin(t)$$

Syntax

C	FORTRAN
complex x, y; y = c_sqrt (x);	COMPLEX x, y y = csqrt (x)

Input $x.re \neq 0$; $x.im \neq 0$; $x.re$ and $x.im$ not both zero

Output

Precision 6.89 digits for the imaginary part; 6.97 digits for the real part

Timing 220–269 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.14 dacos ..... Inverse Cosine

Computes...

$$y \leftarrow \text{Arc cos } x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = acos (x);	DOUBLE PRECISION x, y y = acos (x)

Input $-1 \leq x \leq 1$

Output $0 \leq y \leq \pi$

Precision 15.54 digits

Timing 122, 197 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.15 dasin ..... Inverse Sine

Computes...

$$y \leftarrow \text{Arc sin } x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = asin (x);	DOUBLE PRECISION x, y y = asin (x)

Input $-1 \leq x \leq 1$; zeros and denormals return $\pi/2$; infinities are invalid.

Output $-\pi/2 \leq y \leq \pi/2$

Precision 15.45 digits

Timing 119, 194 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.16 datan ..... Inverse Tangent

Computes...

$$y \leftarrow \text{Arc tan } x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = atan (x);	DOUBLE PRECISION x, y y = atan (x)

Input $-\infty < x < +\infty$

Output $-\pi/2 \leq y \leq \pi/2$; $-\infty$ returns $-\pi/2$; $+\infty$ returns $+\pi/2$.

Precision 15.84 digits

Timing 137, 145 clocks

Exceptions None.

2.17 datan2 ..... Inverse Tangent of Quotient

Computes...

$$z \leftarrow \text{Arc tan}(x/y)$$

...where x, y, and z are double-precision real numbers.

Syntax

C	FORTRAN
double x, y, z; z = atan2 (x, y);	DOUBLE PRECISION x, y, z z = atan2 (x, y)

Input x and y not both 0.

Output In the range of π (points just below the negative x-axis) to π (points at or just above the negative x-axis). The range of the result depends on the relationshp between the operands according to Table 2-1.

Precision 15.35 digits

Timing 11, 12, 190, 198 clocks

Exceptions

Type	Condition	Masked Response
I	$x = \pm\infty$ or $y = \pm\infty$	QNaN *indefinite*
X	$x = 0$ and $y = 0$	QNaN*indefinite*
I	invalid inputs	QNaN *indefinite*

2.18 dcos ..... Cosine

Computes...

$$y \leftarrow \cos x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = cos (x);	DOUBLE PRECISION x, y y = cos (x)

Input Infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 15.48 digits

Timing 55, 77, 80 clocks

Exceptions Trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.19 dcosh .....Hyperbolic Cosine

Computes...

$$y \leftarrow \cosh x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = cosh (x);	DOUBLE PRECISION x, y y = cosh (x)

Input $-710.4758600 < x < 710.4758600$

Output

Precision 6.52 digits

Timing 129 clocks

Exceptions

Type	Condition	Masked Response
O	$\|x\| \geq 710.4758600$	$+\infty$
I	invalid inputs	QNaN*indefinite*

2.20 dexp ..... Exponential

Computes...

$$y \leftarrow e^x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = exp (x);	DOUBLE PRECISION x, y y = exp (x)

Input $-707.7032713 < x < 709.7827128$

Output

Precision 15.44 digits

Timing 111 clocks

Exceptions

Type	Condition	Masked Response
O	$x > 709.7827128$	$+\infty$
U	$x < -707.7032713$	zero
I	invalid inputs	QNaN *indefinite*

2.21 dlog ..... Natural Logarithm

Computes...

$$y \leftarrow \log_e x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = log (x);	DOUBLE PRECISION x, y y = log (x)

Input $x > 0$

Output

Precision 15.65 digits

Timing 123 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.22 dlog10 ..... Log Base 10

Computes...

$$y \leftarrow \log_{10} x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = log10 (x);	DOUBLE PRECISION x, y y = log10 (x)

Input $x > 0$

Output

Precision 15.65 digits

Timing 123 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.23 dsin ..... Sine

Computes...

$$y \leftarrow \sin x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = sin (x);	DOUBLE PRECISION x, y y = sin (x)

Input $-\infty < x < +\infty$; infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 15.41 digits

Timing 55, 77, 80 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.24 dsincos ..... Sine and Cosine

Define...

struct pair {double y, double z}

Computes...

$pair.y \leftarrow \sin x$
$pair.z \leftarrow \cos x$

...where x, y, and z are double-precision real numbers.

Syntax

C	FORTRAN
double x pair = sincos (x);	Not Applicable

Input $-\infty < x < +\infty$; infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 15.41, 15.48 digits respectively

Timing 106, 128 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.25 dsinh ..... Hyperbolic Sine

Computes...

$$y \leftarrow \sinh x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = sinh (x);	DOUBLE PRECISION x, y y = sinh (x)

Input In the range $-710.4758600 < x < 710.4758600$

Output

Precision 15.27 digits

Timing 57, 130 clocks

Exceptions Overflow trap if x out of range, which returns positive infinity*{?}*.

Type	Condition	Masked Response
O	$x \geq 710.4758600$	$+\infty$
O	$x \geq -710.4758600$	$-\infty$
I	invalid inputs	QNaN *indefinite*

2.26 dsqrt .....Square Root

Computes...

$$y \leftarrow \text{SQRT}(x)$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = sqrt (x);	DOUBLE PRECISION x, y y = sqrt (x)

Input $x \geq 0$

Output $y \geq 0$; dsqrt(–0) = –0; dsqrt(+∞) = +∞

Precision 15.98 digits

Timing 60 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.25 dsinh ..... Hyperbolic Sine

Computes...

$$y \leftarrow \sinh x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = sinh (x);	DOUBLE PRECISION x, y y = sinh (x)

Input In the range $-710.4758600 < x < 710.4758600$

Output

Precision 15.27 digits

Timing 57, 130 clocks

Exceptions Overflow trap if x out of range, which returns positive infinity*{?}*.

Type	Condition	Masked Response
O	$x \geq 710.4758600$	$+\infty$
O	$x \geq -710.4758600$	$-\infty$
I	invalid inputs	QNaN *indefinite*

2.26 dsqrt .....Square Root

Computes...

$$y \leftarrow \text{SQRT}(x)$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = sqrt (x);	DOUBLE PRECISION x, y y = sqrt (x)

Input $x \geq 0$

Output $y \geq 0$; dsqrt(–0) = –0; dsqrt(+∞) = +∞

Precision 15.98 digits

Timing 60 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.27 dtan ..... Tangent

Computes...

$$y \leftarrow \tan x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = tan (x);	DOUBLE PRECISION x, y y = tan (x)

Input $-\infty < x < +\infty$, in radians

Output

Precision 15.35 digits

Timing 115, 127, 163, 175 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.28 dtanh .....Hyperbolic Tangent

Computes...

$$y \leftarrow \tanh x$$

...where x and y are double-precision real numbers.

Syntax

C	FORTRAN
double x, y; y = tanh (x);	DOUBLE PRECISION x, y y = tanh (x)

Input

Output

Precision 15.35 digits

Timing 20, 59, 128 clocks

Exceptions None.

2.29 exp ..... Exponential

Computes...

$$y \leftarrow e^x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = expf (x);	REAL x, y y = exp (x)

Input $-87.336545 < x < 88.722839$

Output

Precision 6.72 digits

Timing 63 clocks

Exceptions

Type	Condition	Masked Response
O	$x > 88.722839$	$+\infty$
U	$x < -87.336544$	zero
I	invalid inputs	QNaN *indefinite*

2.30 sin ..... Sine

Computes...

$$y \leftarrow \sin x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = sin*f* (x);	REAL x, y y = sin (x)

Input $-\infty < x < +\infty$; infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 6.92 digits

Timing 30, 55, 58 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.31 sincos ..... Sine and Cosine

Define...

struct pair {float y, float z}

Computes...

$pair.y \leftarrow \sin x$
$pair.z \leftarrow \cos x$

...where x, y, and z are single-precision real numbers.

Syntax

C	FORTRAN
float x; pair = sincos*f* (x);	Not Applicable

Input $-\infty < x < +\infty$; infinities are invalid.

Output $-1 \leq y \leq 1$

Precision 6.92 digits

Timing 54, 74 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.32 sinh ..... Hyperbolic Sine

Computes...

$$y \leftarrow \sinh x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = sinh*f* (x);	REAL x, y y = sinh (x)

Input In the range $-89.41598629 < x < 89.41598629$

Output

Precision 6.92 digits

Timing 67, 101 clocks

Exceptions Overflow trap if x out of range, which returns positive infinity.

Type	Condition	Masked Response
O	$x \geq 89.41598629$	$+\infty$
O	$x \geq -89.41598629$	$-\infty$
I	invalid inputs	QNaN *indefinite*

2.33 sqrt .....Square Root

Computes...

$$y \leftarrow \text{SQRT}(x)$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = sqrt*f* (x);	REAL x, y y = sqrt (x)

Input $x \geq 0$

Output $y \geq 0$; sqrt(–0) = –0; sqrt(+∞) = +∞

Precision 7.31 digits

Timing 37 clocks

Exceptions

Type	Condition	Masked Response
I	x out of range	QNaN *indefinite*

2.34 tan ..... Tangent

Computes...

$$y \leftarrow \tan x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = tan*f* (x);	REAL x, y y = tan (x)

Input $-\infty < x < +\infty$, in radians

Output $-\pi/2 \leq y \leq +\pi/2$

Precision 6.53 digits

Timing 69, 81, 90, 102 clocks

Exceptions Invalid operation trap on infinity.

Type	Condition	Masked Response
I	invalid inputs	QNaN *indefinite*

2.35 tanh .....Hyperbolic Tangent

Computes...

$$y \leftarrow \tanh x$$

...where x and y are single-precision real numbers.

Syntax

C	FORTRAN
float x, y; y = tanh*f* (x);	REAL x, y y = tanh (x)

Input $-\infty \leq x \leq +\infty$

Output $-1 \leq y \leq +1$

Precision 6.92 digits

Timing 20, 29, 103 clocks

Exceptions None.