

Libraries

COLLABORATORS

	<i>TITLE :</i> Libraries		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Libraries	1
1.1	Amiga® RKM Libraries: 35 Math Libraries	1
1.2	35 Math Libraries / Math Libraries and Functions	2
1.3	35 Math Libraries / FFP Floating Point Data Format	2
1.4	35 Math Libraries / FFP Basic Mathematics Library	3
1.5	35 / FFP Basic Mathematics Library / FFP Basic Functions	4
1.6	35 Math Libraries / FFP Transcendental Mathematics Library	7
1.7	35 / Transcendental Mathematics Library / FFP Transcendental Functions	8
1.8	35 Math Libraries / FFP Mathematics Conversion Library	12
1.9	35 / FFP Mathematics Conversion Library / Math Support Functions	13
1.10	35 Math Libraries / IEEE Single-Precision Data Format	13
1.11	35 Math Libraries / IEEE Single-Precision Basic Math Library	14
1.12	35 / SP Basic Math Library / SP IEEE Basic Functions (V36 or Greater)	15
1.13	35 Math Libraries / IEEE Single-Precision Transcendental Math Library	17
1.14	35 // SP IEEE Transcendental Functions (V36 Or Greater)	18
1.15	35 Math Libraries / IEEE Double-Precision Data Format	21
1.16	35 Math Libraries / IEEE Double-Precision Basic Math Library	22
1.17	35 / IEEE Double-Precision Basic Math Library / DP IEEE Basic Functions	22
1.18	35 Math Libraries / IEEE Double-Precision Transcendental Math Library	25
1.19	35 // DP IEEE Transcendental Functions	26
1.20	35 Math Libraries / Function Reference	29
1.21	35 Math Libraries / Compile and Link Commands for SAS C 5.10	32

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: 35 Math Libraries

This chapter describes the structure and calling sequences required to access the Motorola Fast Floating Point (FFP), the IEEE single-precision math libraries and the IEEE double-precision math libraries via the Amiga-supplied interfaces.

In its present state, the FFP library consists of three separate entities: the basic math library, the transcendental math library, and C and assembly-language interfaces to the basic math library plus FFP conversion functions. The IEEE single-precision, introduced in Release 2, and the double-precision libraries each presently consists of two entities: the basic math library and the transcendental math library.

Open Each Library Separately.

Each Task using an IEEE math library must open the library itself. Library base pointers to these libraries may not be shared. Libraries can be context sensitive and may use the Task structure to keep track of the current context. Sharing of library bases by Tasks may seem to work in some systems. This is true for any of the IEEE math libraries.

Depending on the compiler used, it is not always necessary to explicitly call the library functions for basic floating point operations as adding, subtracting, dividing, etc. Consult the manual supplied with the compiler for information regarding the compiler options for floating point functions.

Math Libraries and Functions
FFP Floating Point Data Format
FFP Basic Mathematics Library
FFP Transcendental Mathematics Library
FFP Mathematics Conversion Library
IEEE Single-Precision Data Format
IEEE Single-Precision Basic Math Library
IEEE Single-Precision Transcendental Math Library
IEEE Double-Precision Data Format
IEEE Double-Precision Basic Math Library
IEEE Double-Precision Transcendental Math Library

Function Reference
 Compile and Link Commands for SAS C 5.10

1.2 35 Math Libraries / Math Libraries and Functions

There are six math libraries providing functions ranging from adding two floating point numbers to calculating a hyperbolic cosine. They are:

```
mathffp.library
    the basic function library

mathtrans.library
    the FFP transcendental math library

mathieeesingbas.library
    the IEEE single-precision library

mathieeesingtrans.library
    the IEEE single-precision transcendental library

mathieeedoubbas.library
    the IEEE double-precision library

mathieeesingtrans.library
    the IEEE double-precision transcendental library
```

1.3 35 Math Libraries / FFP Floating Point Data Format

FFP floating-point variables are defined within C by the float or FLOAT directive. In assembly language they are simply defined by a DC.L/DS.L statement. All FFP floating-point variables are defined as 32-bit entities (longwords) with the following format:

MMMMMMMM	MMMMMMMM	MMMMMMMM	EEEEEEE
31	23	15	7

The mantissa is considered to be a binary fixed-point fraction; except for 0, it is always normalized (the mantissa is shifted over and the exponent adjusted, so that the mantissa has a 1 bit in its highest position). Thus, it represents a value of less than 1 but greater than or equal to 1/2.

The sign bit is reset (0) for a positive value and set (1) for a negative value.

The exponent is the power of two needed to correctly position the mantissa to reflect the number's true arithmetic value. It is held in excess-64 notation, which means that the two's-complement values are adjusted upward

by 64, thus changing \$40 (-64) through \$3F (+63) to \$00 through \$7F. This facilitates comparisons among floating-point values.

The value of 0 is defined as all 32 bits being 0s. The sign, exponent, and mantissa are entirely cleared. Thus, 0s are always treated as positive.

The range allowed by this format is as follows:

DECIMAL:

$$9.22337177 * 10^{18} > +\text{VALUE} > 5.42101070 * 10^{-20}$$

$$-9.22337177 * 10^{18} < -\text{VALUE} < -2.71050535 * 10^{-20}$$

BINARY (HEXADECIMAL):

$$.FFFFFF * 2^{63} > +\text{VALUE} > .800000 * 2^{-63}$$

$$-.FFFFFF * 2^{63} < -\text{VALUE} < -.800000 * 2^{-64}$$

Remember that you cannot perform any arithmetic on these variables without using the fast floating-point libraries. The formats of the variables are incompatible with the arithmetic format of C-generated code; hence, all floating-point operations are performed through function calls.

1.4 35 Math Libraries / FFP Basic Mathematics Library

The FFP basic math library contains entries for the basic mathematics functions such as add, subtract and divide. It resides in ROM and is opened by calling `OpenLibrary()` with "mathffp.library" as the argument.

```
#include <exec/types.h>
#include <libraries/mathffp.h>

#include <clib/mathffp_protos.h>

struct Library *MathBase;

VOID main()
{
  if (MathBase = OpenLibrary("mathffp.library", 0))
  {
    . . .

    CloseLibrary(MathBase);
  }
  else
    printf("Can't open mathffp.library\n");
}
```

The global variable `MathBase` is used internally for all future library references.

FFP Basic Functions

1.5 35 / FFP Basic Mathematics Library / FFP Basic Functions

SPAbs() FLOAT SPAbs(FLOAT parm);
Take absolute value of FFP variable.

SPAdd() FLOAT SPAdd(FLOAT leftParm, FLOAT rightParm);
Add two FFP variables.

SPCeil() FLOAT SPCeil(FLOAT parm);~
Computer largest integer less than or equal to variable.

SPCmp() LONG SPCmp(FLOAT leftParm, FLOAT rightParm);
Compare two FFP variables.

SPDiv() FLOAT SPDiv(FLOAT leftParm, FLOAT rightParm);
Divide two FFP variables.

SPFix() LONG SPFix(FLOAT parm);
Convert FFP variable to integer.

SPFloor() FLOAT SPFloor(FLOAT parm);
Compute least integer greater than or equal to variable.

SPFlt() FLOAT SPFlt(long integer);
Convert integer variable to FFP.

SPMul() FLOAT SPMul(FLOAT leftParm, FLOAT rightParm);
Multiply two FFP variables.

SPNeg() FLOAT SPNeg(FLOAT parm);
Take two's complement of FFP variable.

SPSub() FLOAT SPSub(FLOAT leftParm, FLOAT rightParm);
Subtract two FFP variables.

SPTst() LONG SPTst(FLOAT parm);
Test an FFP variable against zero.

Be sure to include the proper data type definitions shown below.

```
#include <exec/types.h>
#include <libraries/mathffp.h>

#include <clib/mathffp_protos.h>

struct Library *MathBase;

VOID main()
{
    FLOAT f1, f2, f3;
    LONG  i1;

    if (MathBase = OpenLibrary("mathffp.library", 0))
```

```

{
i1 = SPFix(f1);          /* Call SPFix entry */
f1 = SPFlt(i1);         /* Call SPFlt entry */

if (SPCmp(f1,f2)) {};   /* Call SPCmp entry */
if (!(SPTst(f1))) {};   /* Call SPTst entry */

f1 = SPAbs(f2);         /* Call SPAbs entry */
f1 = SPNeg(f2);         /* Call SPNeg entry */
f1 = SPAdd(f2, f3);     /* Call SPAdd entry */
f1 = SPSub(f2, f3);     /* Call SPSub entry */
f1 = SPMul(f2, f3);     /* Call SPMul entry */
f1 = SPDiv(f2, f3);    /* Call SPDiv entry */
f1 = SPCEil(f2);        /* Call SPCEil entry */
f1 = SPFloor(f2);       /* Call SPFLOOR entry */

CloseLibrary(MathBase);
}
else
printf("Can't open mathffp.library\n");
}

```

The assembly language interface to the FFP basic math routines is shown below, including some details about how the system flags are affected by each operation. The access mechanism is:

```

MOVEA.L _MathBase,A6
JSR _LVOSPFix(A6)

```

FFP Basic Assembly Functions			
Function	Input	Output	Condition Codes
<code>_LVOSPABs</code>	D0=FFP arg	D0=FFP absolute value	N=0 Z=1 if result is zero V=0 C=undefined X=undefined
<code>_LVOSPAdd</code>	D1=FFP arg1 D0=FFP arg2	D0=FFP addition of arg1 + arg2	N=1 if result is negative Z=1 if result is zero V=1 if result overflowed C=undefined Z=undefined
<code>_LVOSPCeil</code>	D0=FFP arg	D0=least integer >=arg	N=1 if result is negative Z=1 if result is zero V=undefined

			C=undefined
			Z=undefined
-----	-----	-----	-----
_LVOSPCmp	D1=FFP arg1 D0=FFP arg2	D0=+1 if arg1>arg2 D0=-1 if arg1<arg2 D0=0 if arg1=arg2	N=0 Z=1 if result is zero V=0 C=undefined X=undefined GT=arg2>arg1 GE=arg2>=arg1 EQ=arg2=arg1 NE=arg2<>arg1 LT=arg2<arg1 LE=arg2<=arg1
-----	-----	-----	-----
_LVOSPDiv	D1=FFP arg1 D0=FFP arg2	D0=FFP division of arg2/arg1	N=1 if result is negative Z=1 if result is zero V=1 if result overflowed C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPFix	D0=FFP arg	D0=Integer (two's complement)	N=1 if result is negative Z=1 if result is zero V=1 if overflow occurred C=undefined X=undefined
-----	-----	-----	-----
_LVOSPFloor	D0=FFP arg	D0=largest integer <= arg	N=1 if result is negative Z=1 if result is zero V=undefined C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPFlt	D0=Integer (two's complement)	D0=FFP result	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined
-----	-----	-----	-----
_LVOSPMul	D0=FFP arg1 D1=FFP arg2	D0=FFP multiplication of arg1*arg2	N=1 if result is negative Z=1 if result is zero V=1 if result overflowed C=undefined

			Z=undefined
-----	-----	-----	-----
_LVOSPNeg	D0=FFP arg	D0=FFP negated	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined
-----	-----	-----	-----
_LVOSPSub	D1=FFP arg1 D0=FFP arg2	D0=FFP subtraction of arg2-arg1	N=1 if result is negative Z=1 if result is zero V=1 if result overflowed C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPTst	D1=FFP arg	D0=+1 if arg>0.0 D0=-1 if arg<0.0 D0=0 if arg=0.0	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined EQ=arg=0.0 NE=arg<>0.0 PL=arg>=0.0 MI=arg<0.0
	Note: This routine trashes the arg in D1.		

1.6 35 Math Libraries / FFP Transcendental Mathematics Library

The FFP transcendental math library contains entries for the transcendental math functions sine, cosine, and square root. It resides on disk and is opened by calling `OpenLibrary()` with `"mathtrans.library"` as the argument.

```
#include <exec/types.h>
#include <libraries/mathffp.h>

#include <clib/mathffp_protos.h>
#include <clib/mathtrans_protos.h>

struct Library *MathTransBase;

VOID main()
{
  if (MathTransBase = OpenLibrary("mathtrans.library",0))
  {
    .
    .
    .
  }
}
```

```

        CloseLibrary(MathTransBase);
    }
else
    printf("Can't open mathtrans.library\n");
}

```

The global variable `MathTransBase` is used internally for all future library references. Note that the transcendental math library is dependent upon the basic math library, which it will open if it is not open already. If you want to use the basic math functions in conjunction with the transcendental math functions however, you have to specifically open the basic math library yourself.

FFP Transcendental Functions

1.7 35 / Transcendental Mathematics Library / FFP Transcendental Functions

```
SPAsin()    FLOAT SPAsin( FLOAT parm );
Return arccosine of FFP variable.
```

```
SPAcos()    FLOAT SPAcos( FLOAT parm );
Return arctangent of FFP variable.
```

```
SPAtan()    FLOAT SPAtan( FLOAT parm );
Return arcsine of FFP variable.
```

```
SPSin()     FLOAT SPSin( FLOAT parm );
Return sine of FFP variable. This function accepts an FFP radian
argument and returns the trigonometric sine value. For extremely
large arguments where little or no precision would result, the
computation is aborted and the "V" condition code is set. A direct
return to the caller is made.
```

```
SPCos()     FLOAT SPCos( FLOAT parm );
Return cosine of FFP variable. This function accepts an FFP radian
argument and returns the trigonometric cosine value. For extremely
large arguments where little or no precision would result, the
computation is aborted and the "V" condition code is set. A direct
return to the caller is made.
```

```
SPTan()     FLOAT SPTan( FLOAT parm );
Return tangent of FFP variable. This function accepts an FFP radian
argument and returns the trigonometric tangent value. For extremely
large arguments where little or no precision would result, the
computation is aborted and the "V" condition code is set. A direct
return to the caller is made.
```

```
SPSincos()  FLOAT SPSincos( FLOAT *cosResult, FLOAT parm);
Return sine and cosine of FFP variable. This function accepts an FFP
radian argument and returns the trigonometric sine as its result and
the trigonometric cosine in the first parameter. If both the sine
and cosine are required for a single radian value, this function will
result in almost twice the execution speed of calling the SPSin() and
SPCos() functions independently. For extremely large arguments where
little or no precision would result, the computation is aborted and
```

the "V" condition code is set. A direct return to the caller is made.

SPSinh() FLOAT SPSinh(FLOAT parm);
Return hyperbolic sine of FFP variable.

SPCosh() FLOAT SPCosh(FLOAT parm);
Return hyperbolic cosine of FFP variable.

SPTanh() FLOAT SPTanh(FLOAT parm);
Return hyperbolic tangent of FFP variable.

SPExp() FLOAT SPExp(FLOAT parm);
Return e to the FFP variable power. This function accepts an FFP argument and returns the result representing the value of e (2.71828...) raised to that power.

SPLog() FLOAT SPLog(FLOAT parm);
Return natural log (base e) of FFP variable.

SPLog10() FLOAT SPLog10(FLOAT parm);
Return log (base 10) of FFP variable.

SPPow() FLOAT SPPow(FLOAT power, FLOAT arg);
Return FFP arg2 to FFP arg1.

SPSqrt() FLOAT SPSqrt(FLOAT parm);
Return square root of FFP variable.

SPTieee() FLOAT SPTieee(FLOAT parm);
Convert FFP variable to IEEE format

SPFieee() FLOAT SPFieee(FLOAT parm);
Convert IEEE variable to FFP format.

Be sure to include proper data type definitions, as shown in the example below.

mathtrans.c

The Amiga assembly language interface to the FFP transcendental math routines is shown below, including some details about how the system flags are affected by the operation. This interface resides in the library file amiga.lib and must be linked with the user code. Note that the access mechanism from assembly language is:

```
MOVEA.L _MathTransBase,A6
JSR     _LVOSPAsin(A6)
```

FFP Transcendental Assembly Functions			
Function	Input	Output	Condition Codes
_LVOSPAsin	D0=FFP arg	D0=FFP arcsine radian	N=0 Z=1 if result

			is zero
			V=0
			C=undefined
			X=undefined
-----	-----	-----	-----
_LVOSPACos	D0=FFP arg	D0=FFP arccosine radian	N=0 Z=1 if result is zero V=1 if overflow occurred C=undefined X=undefined
-----	-----	-----	-----
_LVOSPAtan	D0=FFP arg	D0=FFP arctangent radian	N=0 Z=1 if result is zero V=0 C=undefined X=undefined
-----	-----	-----	-----
_LVOSPSin	D0=FFP arg in radians	D0=FFP sine	N=1 if result is negative Z=1 if result is zero V=1 if result is meaningless (input magnitude too large) C=undefined X=undefined
-----	-----	-----	-----
_LVOSPCos	D0=FFP arg in radians	D0=FFP cosine	N=1 if result is negative Z=1 if result is zero V=1 if result is meaningless (input magnitude too large) C=undefined X=undefined
-----	-----	-----	-----
_LVOSPTan	D0=FFP arg in radians	D0=FFP tangent	N=1 if result is negative Z=1 if result is zero V=1 if result is meaningless (input magnitude too large) C=undefined X=undefined
-----	-----	-----	-----
_LVOSPSincos	D0=FFP arg in radians D1=Address to store	D0=FFP sine (D1)=FFP cosine	N=1 if result is negative Z=1 if result is zero

	cosine result		V=1 if result is meaningless (input magnitude too large) C=undefined X=undefined
<hr/>			
<code>_LVOSPSinh</code>	D0=FFP arg in radians	D0=FFP hyperbolic sine	N=1 if result is negative Z=1 if result is zero V=1 if overflow occurred C=undefined X=undefined
<hr/>			
<code>_LVOSPCosh</code>	D0=FFP arg in radians	D0=FFP hyperbolic cosine	N=1 if result is negative Z=1 if result is zero V=1 if overflow occurred C=undefined X=undefined
<hr/>			
<code>_LVOSPtanh</code>	D0=FFP arg in radians	D0=FFP hyperbolic tangent	N=1 if result is negative Z=1 if result is zero V=1 if overflow occurred C=undefined X=undefined
<hr/>			
<code>_LVOSPExp</code>	D0=FFP arg	D0=FFP exponential	N=0 Z=1 if result is zero V=1 if overflow occurred C=undefined Z=undefined
<hr/>			
<code>_LVOSPLog</code>	D0=FFP arg	D0=FFP natural logarithm	N=1 if result is negative Z=1 if result is zero V=1 if arg is negative or zero C=undefined Z=undefined
<hr/>			
<code>_LVOSPLog10</code>	D0=FFP arg	D0=FFP logarithm (base 10)	N=1 if result is negative Z=1 if result is zero V=1 if arg is negative or zero

			C=undefined
			Z=undefined
-----	-----	-----	-----
_LVOSPPow	D0=FFP exponent value D1=FFP arg value	D0=FFP result of arg taken to exp power	N=0 Z=1 if result is zero V=1 if result overflowed or arg < 0 C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPSqrt	D0=FFP arg	D0=FFP square root	N=0 Z=1 if result is zero V=1 if arg was negative C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPTieee	D0=FFP format arg	D0=IEEE floating-point format	N=1 if result is negative Z=1 if result is zero V=undefined C=undefined Z=undefined
-----	-----	-----	-----
_LVOSPFieee	D0=IEEE floating-point format arg	D0=FFP format	N=undefined Z=1 if result is zero V=1 if result overflowed C=undefined Z=undefined
-----	-----	-----	-----

1.8 35 Math Libraries / FFP Mathematics Conversion Library

The FFP mathematics conversion library provides functions to convert ASCII strings to their FFP equivalents and vice versa.

It is accessed by linking code into the executable file being created. The name of the file to include in the library description of the link command line is `amiga.lib`. When this is included, direct calls are made to the conversion functions. Only a C interface exists for the conversion functions; there is no assembly language interface. The basic math library is required in order to access these functions.

```
#include <exec/types.h>
#include <libraries/mathffp.h>

#include <clib/mathffp_protos.h>

struct Library *MathBase;
```

```

VOID main()
{
if (MathBase = OpenLibrary("mathffp.library", 33))
    {
        . . .

        CloseLibrary(MathBase);
    }
else
    printf("Can't open mathffp.library\n");
}

```

Math Support Functions

1.9 35 / FFP Mathematics Conversion Library / Math Support Functions

```

afp()    FLOAT afp( BYTE *string );
         Convert ASCII string into FFP equivalent.

arnd()   VOID arnd( LONG place, LONG exp, BYTE *string);
         Round ASCII representation of FFP number.

dbf()    FLOAT dbf( ULONG exp, ULONG mant);
         Convert FFP dual-binary number to FFP equivalent.

fpa()    LONG fpa( FLOAT fnum, BYTE *string);
         Convert FFP variable into ASCII equivalent.

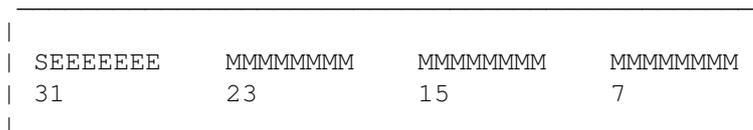
```

Be sure to include proper data type definitions, as shown in the example below. Print statements have been included to help clarify the format of the math conversion function calls.

mathffp.c

1.10 35 Math Libraries / IEEE Single-Precision Data Format

The IEEE single-precision variables are defined as 32-bit entities with the following format:



Hidden Bit In The Mantissa.

There is a "hidden" bit in the mantissa part of the IEEE numbers. Since all numbers are normalized, the integer (high) bit of the mantissa is dropped off. The IEEE single-precision range is 1.3E-38

(1.4E-45 de-normalized) to 3.4E+38.

The exponent is the power of two needed to correctly position the mantissa to reflect the number's true arithmetic value. If both the exponent and the mantissa have zero in every position, the value is zero. If only the exponent has zero in every position, the value is an unnormal (extremely small). If all bits of the exponent are set to 1 the value is either a positive or negative infinity or a Not a Number (NaN). NaN is sometimes used to indicate an uninitialized variable.

1.11 35 Math Libraries / IEEE Single-Precision Basic Math Library

The ROM-based IEEE single-precision basic math library was introduced in V36. This library contains entries for the basic IEEE single-precision mathematics functions, such as add, subtract, and divide. (Note, registered developers can license a disk-based version of this library from CATS, for usage with V33).

The library is opened by making calling `OpenLibrary()` with `"mathieeesingbas.library"` as the argument. Do not share the library base pointer between tasks -- see note at beginning of chapter for details.

```
#include <exec/types.h>
#include <libraries/mathieeesp.h>

#include <clib/mathsingbas_protos.h>

struct Library *MathIeeeSingBasBase;

VOID main()
{
    /* do not share base pointer between tasks. */
    if (MathIeeeSingBasBase = OpenLibrary("mathieeesingbas.library", 37))
    {
        .
        .
        .
        CloseLibrary(MathIeeeSingBasBase);
    }
    else
        printf("Can't open mathieeesingbas.library\n");
}
```

The global variable `MathIeeeSingBasBase` is used internally for all future library references.

If an 680X0/68881/68882 processor combination is available, it will be used by the IEEE single-precision basic library instead of the software emulation. Also, if an autoconfigured math resource is available, that will be used. Typically this is a 68881 designed as a 16 bit I/O port, but it could be another device as well.

SP IEEE Basic Functions (V36 or Greater)

1.12 35 / SP Basic Math Library / SP IEEE Basic Functions (V36 or Greater)

```

IEEESPAbs()    FLOAT ( FLOAT parm );
    Take absolute value of IEEE single-precision variable.

IEEESPAdd()    FLOAT IEEESPAdd( FLOAT leftParm, FLOAT rightParm);
    Add two IEEE single-precision variables.

IEEESPCeil()   FLOAT IEEESPCeil( FLOAT parm );
    Compute least integer greater than or equal to variable.

IEEESPCmp()    LONG  IEEESPCmp( FLOAT leftParm, FLOAT rightParm );
    Compare two IEEE single-precision variables.

IEEESPDiv()    FLOAT IEEESPDiv( FLOAT dividend, FLOAT divisor );
    Divide two IEEE single-precision variables.

IEEESPFix()    LONG  IEEESPFix( FLOAT parm );
    Convert IEEE single-precision variable to integer.

IEEESPFloor()  FLOAT IEEESPFloor( FLOAT parm );
    Compute largest integer less than or equal to variable.

IEEESPFlt()    FLOAT IEEESPFlt( long integer );
    Convert integer variable to IEEE single-precision.

IEEESPMul()    FLOAT IEEESPMul( FLOAT leftParm, FLOAT rightParm );
    Multiply two IEEE single-precision variables.

IEEESPNeg()    FLOAT IEEESPNeg( FLOAT parm );
    Take two's complement of IEEE single-precision variable.

IEEESPSub()    FLOAT IEEESPSub( FLOAT leftParm, FLOAT rightParm );
    Subtract two IEEE single-precision variables.

IEEESPtst()    LONG  IEEESPtst( FLOAT parm );
    Test an IEEE single-precision variable against zero.

```

Be sure to include proper data type definitions, as shown in the example below.

```
mathieeesingbas.c
```

The Amiga assembly language interface to the IEEE single-precision basic math routines is shown below, including some details about how the system flags are affected by each operation. Note that the access mechanism from assembly language is as shown below:

```

MOVEA.L _MathIeeeSingBasBase,A6
JSR     _LVOIEEESPFix(A6)

```

Function	Input	Output	Condition Codes
----------	-------	--------	-----------------

<code>_LVOIEEEESPFix</code>	D0=IEEE arg double-precision	D0=Integer (two's complement)	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPFlt</code>	D0=Integer arg (two's complement)	D0=IEEE single-precision	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPCmp</code>	D0=IEEE arg1 single-precision D1=IEEE arg2 single-precision	D0=+1 if arg1>arg2 D0=-1 if arg1<arg2 D0=0 if arg1=arg2	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined GT=arg2>arg1 GE=arg2>=arg1 EQ=arg2=arg1 NE=arg2<>arg1 LT=arg2<arg1 E= arg2<=arg1
<code>_LVOIEEEESPTst</code>	D0=IEEE arg single-precision	D0=+1 if arg>0.0 D0=-1 if arg<0.0 D0=0 if arg=0.0	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined EQ=arg=0.0 NE=arg<>0.0 PL=arg>=0.0 MI=arg<0.0
<code>_LVOIEEEESPAbs</code>	D0=IEEE arg single-precision	D0=IEEE single-precision absolute value	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPNeg</code>	D0=IEEE arg single-precision	D0=IEEE single-precision negated	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPAAdd</code>	D0=IEEE arg1 single-precision D1=IEEE arg2 single-precision	D0=IEEE single-precision addition of arg1+arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined

<code>_LVOIEEEESPSub</code>	D0=IEEE arg1 single-precision D1=IEEE arg2 single-precision	D0=IEEE single-precision subtraction of arg1-arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPMul</code>	D0=IEEE arg1 single-precision D1=IEEE arg2 single-precision	D0=IEEE single-precision multiplication of arg1*arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPDiv</code>	D0=IEEE arg1 single-precision D1=IEEE arg2 single-precision	D0=IEEE single-precision division of arg1/arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPCeil</code>	D0=IEEE variable single-precision	D0=least integer >= variable	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEESPFloor</code>	D0=IEEE variable single-precision	D0=largest integer <= arg	N=undefined Z=undefined V=undefined C=undefined X=undefined

1.13 35 Math Libraries / IEEE Single-Precision Transcendental Math Library

The IEEE single-precision transcendental math library was introduced in V36. It contains entries for transcendental math functions such as sine, cosine, and square root.

This library resides on disk and is opened by calling `OpenLibrary()` with `"mathieeesingtrans.library"` as the argument. Do not share the library base pointer between tasks -- see note at beginning of chapter.

```
#include <exec/types.h>
#include <libraries/mathieeesp.h>

struct Library *MathIeeeSingTransBase;

#include <clib/mathsingtrans_protos.h>

VOID main()
{
  if (MathIeeeSingTransBase = OpenLibrary("mathieeesingtrans.library", 37))
  {
    . . .
  }
}
```

```

        CloseLibrary(MathIeeeSingTransBase);
    }
    else printf("Can't open mathieeesingtrans.library\n");
}

```

The global variable `MathIeeeSingTransBase` is used internally for all future library references.

The IEEE single-precision transcendental math library is dependent upon the IEEE single-precision basic math library, which it will open if it is not open already. If you want to use the IEEE single-precision basic math functions in conjunction with the transcendental math functions however, you have to specifically open the basic math library yourself.

Just as the IEEE single-precision basic math library, the IEEE single-precision transcendental math library will take advantage of a 680X0/68881 combination or another math resource, if present.

SP IEEE Transcendental Functions (V36 Or Greater)

1.14 35 // SP IEEE Transcendental Functions (V36 Or Greater)

```

IEEEESPAsin()  FLOAT IEEEESPAsin( FLOAT parm );
    Return arcsine of IEEE single-precision variable.

IEEEESPACos()  FLOAT IEEEESPACos( FLOAT parm );
    Return arccosine of IEEE single-precision variable.

IEEEESPAtan()  FLOAT IEEEESPAtan( FLOAT parm );
    Return arctangent of IEEE single-precision variable.

IEEEESPsin()   FLOAT IEEEESPsin( FLOAT parm );
    Return sine of IEEE single-precision variable. This function accepts
    an IEEE radian argument and returns the trigonometric sine value.

IEEEESPCos()   FLOAT IEEEESPCos( FLOAT parm );
    Return cosine of IEEE single-precision variable. This function
    accepts an IEEE radian argument and returns the trigonometric cosine
    value.

IEEEESPTan()   FLOAT IEEEESPTan( FLOAT parm );
    Return tangent of IEEE single-precision variable. This function
    accepts an IEEE radian argument and returns the trigonometric tangent
    value.

IEEESPSincos() FLOAT IEEESPSincos( FLOAT *cosptr, FLOAT parm );
    Return sine and cosine of IEEE single-precision variable. This
    function accepts an IEEE radian argument and returns the
    trigonometric sine as its result and the ?cosine in the first
    parameter.

IEEEESPsinh()  FLOAT IEEEESPsinh( FLOAT parm );
    Return hyperbolic sine of IEEE single-precision variable.

IEEEESPCosh()  FLOAT IEEEESPCosh( FLOAT parm );

```

Return hyperbolic cosine of IEEE single-precision variable.

```
IEEEESPTanh()  FLOAT IEEEESPTanh( FLOAT parm );
```

Return hyperbolic tangent of IEEE single-precision variable.

```
IEEEESPExp()   FLOAT IEEEESPExp( FLOAT parm );
```

Return e to the IEEE variable power. This function accept an IEEE single-precision argument and returns the result representing the value of e (2.712828...) raised to that power.

```
IEEEESPFieee()  FLOAT IEEEESPFieee( FLOAT parm );
```

Convert IEEE single-precision number to IEEE single-precision number. The only purpose of this function is to provide consistency with the double-precision math IEEE library.

```
IEEEESPLog()   FLOAT IEEEESPLog( FLOAT parm );
```

Return natural log (base e of IEEE single-precision variable.

```
IEEEESPLog10()  FLOAT IEEEESPLog10( FLOAT parm );
```

Return log (base 10) of IEEE single-precision variable.

```
IEEEESPpow()   FLOAT IEEEESPpow( FLOAT exp, FLOAT arg );
```

Return IEEE single-precision arg2 to IEEE single-precision arg1.

```
IEEEESPSqrt()  FLOAT IEEEESPSqrt( FLOAT parm );
```

Return square root of IEEE single-precision variable.

```
IEEEESPTieee()  FLOAT IEEEESPTieee( FLOAT parm );
```

Convert IEEE single-precision number to IEEE single-precision number. The only purpose of this function is to provide consistency with the double-precision math IEEE library.

Be sure to include the proper data type definitions as shown below.

```
mathieeesingtrans.c
```

The section below describes the Amiga assembly interface to the IEEE single-precision transcendental math library. The access mechanism from assembly language is:

```
MOVEA.L _MathIeeeSingTransBase,A6
JSR     _LVOIEEEESPAsin(A6)
```

SP IEEE Transcendental Assembly Functions			
Function	Input	Output	Condition Codes
_LVOIEEEESPAsin	D0=IEEE arg	D0=IEEE arcsine radian	N=undefined Z=undefined V=undefined C=undefined X=undefined

_LVOIEEEESPAcos	D0=IEEE arg single-precision	D0=IEEE arccosine radian	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPAtan	D0=IEEE arg single-precision	D0=IEEE arctangent radian	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPSin	D0=IEEE arg in radians single-precision	D0=IEEE sine	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPCos	D0=IEEE arg in radians single-precision	D0=IEEE cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPTan	D0=IEEE arg in radians single-precision	D0=IEEE tangent	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPSincos	A0=Addr to store cosine result D0=IEEE arg in radians	D0=IEEE sine (A0)=IEEE cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPSinh	D0=IEEE arg in radians single-precision	D0=IEEE hyperbolic sine	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPCosh	D0=IEEE arg in radians single-precision	D0=IEEE hyperbolic cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPTanh	D0=IEEE arg in radians single-precision	D0=IEEE hyperbolic tangent	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEESPExp	D0=IEEE arg single-precision	D0=IEEE exponential	N=undefined Z=undefined V=undefined

			C=undefined
			X=undefined
-----	-----	-----	-----
_LVOIEEEESPLog	D0=IEEE arg single-precision	D0=IEEE natural logarithm	N=undefined Z=undefined V=undefined C=undefined X=undefined
-----	-----	-----	-----
_LVOIEEEESPLog10	D0=IEEE arg single-precision	D0=IEEE logarithm (base 10)	N=undefined Z=undefined V=undefined C=undefined X=undefined
-----	-----	-----	-----
_LVOIEEEESPPow	D0=IEEE exponent value single-precision D1=IEEE arg value single-precision	D0=IEEE result of arg taken to exp power	N=undefined Z=undefined V=undefined C=undefined X=undefined
-----	-----	-----	-----
_LVOIEEEESPSqrt	D0=IEEE arg single-precision	D0=IEEE square root	N=undefined Z=undefined V=undefined C=undefined X=undefined
-----	-----	-----	-----

1.15 35 Math Libraries / IEEE Double-Precision Data Format

The IEEE double-precision variables are defined as 64-bit entities with the following format:

SEEEEEEE	EEEEIMM	MMMMMMMM	MMMMMMMM
63	55	47	39
-----	-----	-----	-----
MMMMMMMM	MMMMMMMM	MMMMMMMM	MMMMMMMM
31	23	15	7
-----	-----	-----	-----

Hidden Bit In The Mantissa.

There is a "hidden" bit in the mantissa part of the IEEE numbers. Since all numbers are normalized, the integer (high) bit of the mantissa is dropped off. The IEEE double-precision range is 2.2E-308 (4.9E-324 de-normalized) to 1.8E+307.

The exponent is the power of two needed to correctly position the mantissa

to reflect the number's true arithmetic value. If both the exponent and the mantissa have zero in every position, the value is zero. If only the exponent has zero in every position, the value is an unnormal (extremely small). If all bits of the exponent are set to 1 the value is either a positive or negative infinity or a Not a Number (NaN). NaN is sometimes used to indicate an uninitialized variable.

1.16 35 Math Libraries / IEEE Double-Precision Basic Math Library

The IEEE double-precision basic math library contains entries for the basic IEEE mathematics functions, such as add, subtract, and divide. This library resides on disk and is opened by calling `OpenLibrary()` with `"mathieeedoubbas.library"` as the argument. Do not share the library base pointer between tasks -- see note at beginning of chapter for details.

```
#include <exec/types.h>
#include <libraries/mathieeedp.h>

#include <clib/mathdoubbas_protos.h>

struct Library *MathIeeeDoubBasBase;

VOID main()
{
    /* do not share base pointer between tasks. */
    if (MathIeeeDoubBasBase = OpenLibrary("mathieeedoubbas.library", 34))
    {
        . . .

        CloseLibrary(MathIeeeDoubBasBase);
    }
    else printf("Can't open mathieeedoubbas.library\n");
}
```

The global variable `MathIeeeDoubBasBase` is used internally for all future library references.

If an 680X0/68881/68882 processor combination is available, it will be used by the IEEE basic library instead of the software emulation. Also, if an autoconfigured math resource is available, that will be used. Typically this is a 68881 designed as a 16 bit I/O port, but it could be another device as well.

DP IEEE Basic Functions

1.17 35 /IEEE Double-Precision Basic Math Library / DP IEEE Basic Functions

```
IEEEEDPAbs()    DOUBLE IEEEEDPAbs( DOUBLE parm );
    Take absolute value of IEEE double-precision variable.
```

```
IEEEEDPAdd()    DOUBLE IEEEEDPAdd( DOUBLE leftParm, DOUBLE rightParm );
    Add two IEEE double-precision variables.
```

```

IEEEEDPCeil()    DOUBLE IEEEEDPCeil( DOUBLE parm );
    Compute least integer greater than or equal to variable.

IEEEEDPCmp()    LONG IEEEEDPCmp( DOUBLE leftParm, DOUBLE rightParm );
    Compare two IEEE double-precision variables.

IEEEEDPDiv()    DOUBLE IEEEEDPDiv( DOUBLE dividend, DOUBLE divisor );
    Divide two IEEE double-precision variables.

IEEEEDPFix()    LONG IEEEEDPFix( DOUBLE parm );
    Convert IEEE double-precision variable to integer.

IEEEEDPFloor()  DOUBLE IEEEEDPFloor( DOUBLE parm );
    Compute largest integer less than or equal to variable.

IEEEEDPFlt()    DOUBLE IEEEEDPFlt( long integer );
    Convert integer variable to IEEE double-precision.

IEEEEDPMul()    DOUBLE IEEEEDPMul( DOUBLE factor1, DOUBLE factor2 );
    Multiply two IEEE double-precision variables.

IEEEEDPNeg()    DOUBLE IEEEEDPNeg( DOUBLE parm );
    Take two's complement of IEEE double-precision variable.

IEEEEDPSub()    DOUBLE IEEEEDPSub( DOUBLE leftParm, DOUBLE rightParm );
    Subtract two IEEE double-precision variables.

IEEEEDPTst()    LONG IEEEEDPTst( DOUBLE parm );
    Test an IEEE double-precision variable against zero.

```

Be sure to include proper data type definitions, as shown in the example below.

```
mathieeeedoubbas.c
```

The Amiga assembly language interface to the IEEE double-precision floating-point basic math routines is shown below, including some details about how the system flags are affected by each operation. The access mechanism from assembly language is:

```

MOVEA.L _MathIeeeDoubBasBase,A6
JSR     _LVOIEEEEDPFix(A6)

```

DP IEEE Basic Assembly Functions			
Function	Input	Output	Condition Codes
<code>_LVOIEEEEDPFix</code>	D0/D1=IEEE arg double-precision	D0=Integer (two's complement)	N=undefined Z=undefined V=undefined C=undefined X=undefined

<code>_LVOIEEEDPF1</code>	D0=Integer arg (two's complement)	D0/D1=IEEE double-precision	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEDPCmp</code>	D0/D1=IEEE arg1 double-precision D2/D3=IEEE arg2 double-precision	D0=+1 if arg1>arg2 D0=-1 if arg1<arg2 D0=0 if arg1=arg2	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined GT=arg2>arg1 GE=arg2>=arg1 EQ=arg2=arg1 NE=arg2<>arg1 LT=arg2<arg1 LE=arg2<=arg1
<code>_LVOIEEEDPTst</code>	D0/D1=IEEE arg double-precision	D0=+1 if arg>0.0 D0=-1 if arg<0.0 D0=0 if arg=0.0	N=1 if result is negative Z=1 if result is zero V=0 C=undefined X=undefined EQ=arg=0.0 NE=arg<>0.0 PL=arg>=0.0 MI=arg<0.0
<code>_LVOIEEEDPAbs</code>	D0/D1=IEEE arg double-precision	D0/D1=IEEE double-precision absolute value	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEDPNeg</code>	D0/D1=IEEE arg double-precision	D0/D1=IEEE double-precision negated	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEDPAdd</code>	D0/D1=IEEE arg1 double-precision D2/D3=IEEE arg2 double-precision	D0/D1=IEEE double-precision addition of arg1+arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEDPSub</code>	D0/D1=IEEE arg1 double-precision D2/D3=IEEE arg2 double-precision	D0/D1=IEEE double-precision subtraction of arg1-arg2	N=undefined Z=undefined V=undefined C=undefined X=undefined

<code>_LVOIEEEEDPMul</code>	<code>D0/D1=IEEE arg1</code>	<code>D0/D1=IEEE</code>	<code>N=undefined</code>
	<code>double-precision</code>	<code>double-precision</code>	<code>Z=undefined</code>
		<code>multiplication of</code>	<code>V=undefined</code>
	<code>D2/D3=IEEE arg2</code>	<code>arg1*arg2</code>	<code>C=undefined</code>
	<code>double-precision</code>		<code>X=undefined</code>
-----	-----	-----	-----
<code>_LVOIEEEEDPDiv</code>	<code>D0/D1=IEEE arg1</code>	<code>D0/D1=IEEE</code>	<code>N=undefined</code>
	<code>double-precision</code>	<code>double-precision</code>	<code>Z=undefined</code>
		<code>division of</code>	<code>V=undefined</code>
	<code>D2/D3=IEEE arg2</code>	<code>arg1/arg2</code>	<code>C=undefined</code>
	<code>double-precision</code>		<code>X=undefined</code>
-----	-----	-----	-----
<code>_LVOIEEEEDPCeil</code>	<code>D0/D1=IEEE arg</code>	<code>D0/D1=least</code>	<code>N=undefined</code>
	<code>double-precision</code>	<code>integer</code>	<code>Z=undefined</code>
		<code>>= arg</code>	<code>V=undefined</code>
			<code>C=undefined</code>
			<code>X=undefined</code>
-----	-----	-----	-----
<code>_LVOIEEEEDPFloor</code>	<code>D0/D1=IEEE arg</code>	<code>D0/D1=largest</code>	<code>N=undefined</code>
	<code>double-precision</code>	<code>integer</code>	<code>Z=undefined</code>
		<code><= arg</code>	<code>V=undefined</code>
			<code>C=undefined</code>
			<code>X=undefined</code>

1.18 35 Math Libraries / IEEE Double-Precision Transcendental Math Library

The IEEE double-precision transcendental math library contains entries for the transcendental math functions such as sine, cosine, and square root. The library resides on disk and is opened by calling `OpenLibrary()` with `"mathieeedoubtrans.library"` as the argument. Do not share the library base pointer between tasks -- see note at beginning of chapter for details.

```
#include <exec/types.h>
#include <libraries/mathieeedp.h>

#include <clib/mathdoubtrans_protos.h>

struct Library *MathIeeeDoubTransBase;

VOID main()
{
  if (MathIeeeDoubTransBase = OpenLibrary("mathieeedoubtrans.library", 34))
  {
    . . .

    CloseLibrary(MathIeeeDoubTransBase);
  }
  else printf("Can't open mathieeedoubtrans.library\n");
}
```

The global variable `MathIeeeDoubTransBase` is used internally for all future library references.

The IEEE double-precision transcendental math library is dependent upon

the IEEE double-precision basic math library, which it will open if it is not open already. If you want to use the IEEE double-precision basic math functions in conjunction with the transcendental math functions however, you have to specifically open the basic math library yourself.

Just as the IEEE double-precision basic math library, the IEEE double-precision transcendental math library will take advantage of a 680X0/68881 combination or another math resource, if present.

DP IEEE Transcendental Functions

1.19 35 // DP IEEE Transcendental Functions

IEEEEDPAsin() DOUBLE IEEEEDPAsin(DOUBLE parm);
Return arcsine of IEEE variable.

IEEEEDPAcos() DOUBLE IEEEEDPAcos(DOUBLE parm);
Return arccosine of IEEE variable.

IEEEEDPAtan() DOUBLE IEEEEDPAtan(DOUBLE parm);
Return arctangent of IEEE variable.

IEEEEDPSin() DOUBLE IEEEEDPSin(DOUBLE parm);
Return sine of IEEE variable. This function accepts an IEEE radian argument and returns the trigonometric sine value.

IEEEEDPCos() DOUBLE IEEEEDPCos(DOUBLE parm);
Return cosine of IEEE variable. This function accepts an IEEE radian argument and returns the trigonometric cosine value.

IEEEEDPTan() DOUBLE IEEEEDPTan(DOUBLE parm);
Return tangent of IEEE variable. This function accepts an IEEE radian argument and returns the trigonometric tangent value.

IEEEEDPSincos() DOUBLE IEEEEDPSincos(DOUBLE *pf2, DOUBLE parm);
Return sine and cosine of IEEE variable. This function accepts an IEEE radian argument and returns the trigonometric sine as its result and the trigonometric cosine in the first parameter.

IEEEEDPSinh() DOUBLE IEEEEDPSinh(DOUBLE parm);
Return hyperbolic sine of IEEE variable.

IEEEEDPCosh() DOUBLE IEEEEDPCosh(DOUBLE parm);
Return hyperbolic cosine of IEEE variable.

IEEEEDPTanh() DOUBLE IEEEEDPTanh(DOUBLE parm);
Return hyperbolic tangent of IEEE variable.

IEEEEDPExp() DOUBLE IEEEEDPExp(DOUBLE parm);
Return e to the IEEE variable power. This function accept an IEEE argument and returns the result representing the value of e (2.712828...) raised to that power.

IEEEEDPFieee() DOUBLE IEEEEDPFieee(FLOAT single);
Convert IEEE single-precision number to IEEE double-precision number.

```

IEEEEDPLog()    DOUBLE IEEEEDPLog( DOUBLE parm );
    Return natural log (base e of IEEE variable.

IEEEEDPLog10()  DOUBLE IEEEEDPLog10( DOUBLE parm );
    Return log (base 10) of IEEE variable.

IEEEEDPPow()    DOUBLE IEEEEDPPow( DOUBLE exp, DOUBLE arg );
    Return IEEE arg2 to IEEE arg1.

IEEEEDPSqrt()   DOUBLE IEEEEDPSqrt( DOUBLE parm );
    Return square root of IEEE variable.

IEEEEDPTieee()  FLOAT IEEEEDPTieee( DOUBLE parm );
    Convert IEEE double-precision number to IEEE single-precision number.

```

Be sure to include proper data type definitions as shown below.

```
mathieeeedoubtrans.c
```

The section below describes the Amiga assembly interface to the IEEE double-precision transcendental math library. The access mechanism from assembly language is:

```

MOVEA.L _MathIeeeDoubTransBase,A6
JSR     _LVOIEEEEDPAsin(A6)

```

DP IEEE Transcendental Assembly Functions			
Function	Input	Output	Condition Codes
_LVOIEEEEDPAsin	D0/D1=IEEE arg	D0/D1=IEEE arcsine radian	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEEDPAcos	D0/D1=IEEE arg	D0/D1=IEEE arccosine radian	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEEDPAtan	D0/D1=IEEE arg	D0/D1=IEEE arctangent radian	N=undefined Z=undefined V=undefined C=undefined X=undefined
_LVOIEEEEDPSin	D0/D1=IEEE arg in radians	D0/D1=IEEE sine	N=undefined Z=undefined V=undefined C=undefined X=undefined

<code>_LVOIEEEEDPCos</code>	D0/D1=IEEE arg in radians	D0/D1=IEEE cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPTan</code>	D0/D1=IEEE arg in radians	D0/D1=IEEE tangent	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPSincos</code>	A0=Address to store cosine result D0/D1=IEEE arg in radians	D0/D1=IEEE sine (A0)=IEEE cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPSin</code>	D0/D1=IEEE arg in radians	D0/D1=IEEE hyperbolic sine	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPCosh</code>	D0/D1=IEEE arg in radians	D0/D1=IEEE hyperbolic cosine	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPTanh</code>	D0/D1=IEEE arg in radians	D0/D1=IEEE hyperbolic tangent	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPExp</code>	D0/D1=IEEE arg	D0/D1=IEEE exponential	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPLog</code>	D0/D1=IEEE arg	D0/D1=IEEE natural logarithm	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPLog10</code>	D0/D1=IEEE arg	D0/D1=IEEE logarithm (base 10)	N=undefined Z=undefined V=undefined C=undefined X=undefined
<code>_LVOIEEEEDPPow</code>	D0/D1=IEEE exp D2/D3=IEEE arg	D0/D1=IEEE of arg taken to	N=undefined Z=undefined

		exp power	V=undefined
			C=undefined
			X=undefined

_LVOIEEEDPSqrt	D0/D1=IEEE arg	D0/D1=IEEE square root	N=undefined
			Z=undefined
			V=undefined
			C=undefined
			X=undefined

_LVOIEEEDPTieee	D0/D1=IEEE arg	D0=single-precision IEEE floating-point format	N=undefined
			Z=undefined
			V=undefined
			C=undefined
			X=undefined

1.20 35 Math Libraries / Function Reference

Here's a brief summary of the functions covered in this chapter. Refer to the Amiga ROM Kernel Reference Manual: Includes and Autodocs for additional information.

FFP Basic Functions	
=====	
SPAbs()	Take absolute value of FFP variable
SPAdd()	Add two FFP variables
SPCeil()	Compute least integer greater than or equal to variable.
SPCmp()	Compare two FFP variables
SPDiv()	Divide two FFP variables
SPFix()	Convert FFP variable to integer
SPFloor()	Computer largest integer less than or equal to variable.
SPFlt()	Convert integer variable to FFP
SPMul()	Multiply two FFP variables
SPNeg()	Take two's complement of FFP variable
SPSub()	Subtract two FFP variables
SPTst()	Test an FFP variable against zero

FFP Transcendental Functions	
=====	
SPAcos()	Return arccosine of FFP variable.
SPAsin()	Return arcsine of FFP variable.
SPAtan()	Return arctangent of FFP variable.
SPCos()	Return cosine of FFP variable.
SPCosh()	Return hyperbolic cosine of FFP variable.
SPExp()	Return e to the FFP variable power.
SPFieee()	Convert IEEE variable to FFP format.
SPLog()	Return natural log (base e) of FFP variable.
SPLog10()	Return log (base 10) of FFP variable.
SPPow()	Return FFP arg2 to FFP arg1.

	SPSin()	Return sine of FFP variable.	
	SPSincos()	Return sine and cosine of FFP variable.	
	SPSinh()	Return hyperbolic sine of FFP variable.	
	SPSqrt()	Return square root of FFP variable.	
	SPTan()	Return tangent of FFP variable.	
	SPTanh()	Return hyperbolic tangent of FFP variable.	
	SPTieee()	Convert FFP variable to IEEE format	

	Math Support Functions		
	=====		
	afp()	Convert ASCII string into FFP equivalent.	
	fpa()	Convert FFP variable into ASCII equivalent.	
	arnd()	Round ASCII representation of FFP number.	
	dbf()	Convert FFP dual-binary number to FFP equivalent.	

	SP IEEE Basic Functions (V36)		
	=====		
	IEEESPAbs()	Take absolute value of IEEE single-precision variable.	
	IEEESPAdd()	Add two IEEE single-precision variables.	
	IEEESPCeil()	Compute least integer greater than or equal to variable.	
	IEEESPCmp()	Compare two IEEE single-precision variables.	
	IEEESPDiv()	Divide two IEEE single-precision variables.	
	IEEESPFix()	Convert IEEE single-precision variable to integer.	
	IEEESPFloor()	Compute largest integer less than or equal to variable.	
	IEEESPFlt()	Convert integer variable to IEEE single-precision.	
	IEEESPMul()	Multiply two IEEE single-precision variables.	
	IEEESPNeg()	Take two's complement of IEEE single-precision variable.	
	IEEESPSub()	Subtract two IEEE single-precision variables.	
	IEEESPtst()	Test an IEEE single-precision variable against zero.	

	SP IEEE Transcendental Functions (V36)		
	=====		
	IEEESPACos()	Return arccosine of IEEE single-precision variable.	
	IEEESPASin()	Return arcsine of IEEE single-precision variable.	
	IEEESPAtan()	Return arctangent of IEEE single-precision variable.	
	IEEESPCos()	Return cosine of IEEE single-precision variable.	
	IEEESPCosh()	Return hyperbolic cosine of IEEE single-precision variable.	
	IEEESPExp()	Return e to the IEEE variable power.	
	IEEESPLog()	Return natural log (base e of IEEE single-precision variable.	
	IEEESPLog10()	Return log (base 10) of IEEE single-precision variable.	
	IEEESPPow()	Return power of IEEE single-precision variable.	
	IEEESPSin()	Return sine of IEEE single-precision variable.	

IEEE SPSincos()	Return sine and cosine of IEEE single-precision variable.	
IEEE SPSinh()	Return hyperbolic sine of IEEE single-precision variable.	
IEEE SPSqrt()	Return square root of IEEE single-precision variable.	
IEEE SPTan()	Return tangent of IEEE single-precision variable.	
IEEE SPTanh()	Return hyperbolic tangent of IEEE single-precision variable.	

DP IEEE Basic Functions

IEEE DP Abs()	Take absolute value of IEEE double-precision variable.	
IEEE DP Add()	Add two IEEE double-precision variables.	
IEEE DP Ceil()	Compute least integer greater than or equal to variable.	
IEEE DP Cmp()	Compare two IEEE double-precision variables.	
IEEE DP Div()	Divide two IEEE double-precision variables.	
IEEE DP Fix()	Convert IEEE double-precision variable to integer.	
IEEE DP Floor()	Compute largest integer less than or equal to variable.	
IEEE DP Flt()	Convert integer variable to IEEE double-precision.	
IEEE DP Mul()	Multiply two IEEE double-precision variables.	
IEEE DP Neg()	Take two's complement of IEEE double-precision variable.	
IEEE DP Sub()	Subtract two IEEE single-precision variables.	
IEEE DP Tst()	Test an IEEE double-precision variable against zero.	

DP IEEE Transcendental Functions

IEEE DP ACos()	Return arccosine of IEEE double-precision variable.	
IEEE DP ASin()	Return arcsine of IEEE double-precision variable.	
IEEE DP ATan()	Return arctangent of IEEE double-precision variable.	
IEEE DPCos()	Return cosine of IEEE double-precision variable.	
IEEE DPCosh()	Return hyperbolic cosine of IEEE double-precision variable.	
IEEE DPExp()	Return e to the IEEE variable power.	
IEEE DPFieee()	Convert IEEE single-precision number to IEEE double-precision number.	
IEEE DPLog()	Return natural log (base e) of IEEE double-precision variable.	
IEEE DPLog10()	Return log (base 10) of IEEE double-precision variable.	
IEEE DPPow()	Return power of IEEE double-precision variable.	
IEEE DPSin()	Return sine of IEEE double-precision variable.	
IEEE DPSincos()	Return sine and cosine of IEEE double-precision variable.	
IEEE DPSinh()	Return hyperbolic sine of IEEE double-precision variable.	
IEEE DPSqrt()	Return square root of IEEE double-precision variable.	
IEEE DPTan()	Return tangent of IEEE double-precision variable.	
IEEE DPTanh()	Return hyperbolic tangent of IEEE double-precision	

```
|          variable. |
| IEEEEDPTieee()   Convert IEEE double-precision number to IEEE |
|          single-precision number. |
|-----|
```

1.21 35 Math Libraries / Compile and Link Commands for SAS C 5.10

FFP Basic, Transcendental and Math Support functions

```
-----
lc -bl -cfistq -ff -v -y <filename>.c
blink lib:c.o + <filename>.o TO
    <filename> LIB lib:lcmffp.lib + lib:lc.lib + lib:amiga.lib
```

IEEE Single-Precision and Double-Precision Basic and Transcendental Functions

```
-----
lc -bl -cfistq -fi -v -y <filename>.c
blink lib:c.o + <filename>.o TO
    <filename> LIB lib:lcmieee.lib + lib:lc.lib + lib:amiga.lib
```