SEGGER Floating-point Library

User Guide & Reference Manual

Document: UM12008
Software Version: 2.10
Revision: 0
Date: May 5, 2020

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com*
Internet:	www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: May 5, 2020

Software	Revision	Date	By	Description
2.20	0	200505	PC	Updated to latest software version.
2.12	0	191220	PC	Chapter "C library API" • Added expm1f().
2.10	0	190307	PC	Release version.
1.00	0	190204	PC	Internal version.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

Chapter 1

Introduction

This section presents an overview of SEGGER Floating-point Library, its structure, and its capabilities.

1.1 What is the SEGGER Floating-point Library?

SEGGER Floating-point Library is an optimized C and assembly language library to implement common floating-point operations on Arm and RISC-V processors.

1.2 Features

SEGGER Floating-point Library is written in standard ANSI C and Arm and RISC-V assembly language and can run on any Arm or RV32I CPU. Here’s a list summarising the main features of SEGGER Floating-point Library:

- Clean ISO/ANSI C source code.
- Fast assembly language floating point support.
- Conforms to standard runtime ABIs for the Arm and RISC-V architectures.
- Simple configuration.
- Royalty free.

1.3 Recommended project structure

We recommend keeping SEGGER Floating-point Library separate from your application files. It is good practice to keep all the program files (including the header files) together in the `LIB` subdirectory of your project's root directory. This practice has the advantage of being very easy to update to newer versions of SEGGER Floating-point Library by simply replacing the `LIB` directory. Your application files can be stored anywhere.

> **Note**
>
> When updating to a newer SEGGER Floating-point Library version: as files may have been added, moved or deleted, the project directories may need to be updated accordingly.

1.4 Package content

SEGGER Floating-point Library is provided in source code and contains everything needed. The following table shows the content of the SEGGER Floating-point Library Package:

Directory	Description
Doc	SEGGER Floating-point Library documentation.
LIB	SEGGER Floating-point Library source code.

1.4.1 Include directories

You should make sure that the system include path contains the following directory:

- LIB

> **Note**
>
> Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of SEGGER Floating-point Library if you have old files included and therefore mix different versions. If you keep SEGGER Floating-point Library in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the LIB directories before to updating.

Chapter 2

Compiling the SEGGER Floating-point Library

2.1 User-facing source files

The standard C library is exposed to the user by a set of header files that provide an interface to the library. In addition, there must be additional "invisible" functions added to provide C language support, such as software floating point and integer mathematics, that the C compiler calls.

The user-facing interface files are:

File	Description
{<__SEGGER_RTL_F-P.h>}	Interface to the SEGGER floating-point library.

In addition some private header files are required:

File	Description
__SEGGER_RTL_F-P_Conf.h	Configuration of the library.
__SEGGER_RTL_F-P_ConfDefaults.h	Default configuration of the library.

2.2 Implementation source files

SEGGER Floating-point Library is delivered in a small number of files that must be added to your project before building:

File	Description
`floatops.c`	Support for high-level floating point functions.
`floatasmops_arm.s`	Support for low-level floating point functions (ARM).
`floatasmops_rv.s`	Support for low-level floating point functions (RISC-V).

2.3 Configuring the library

All source files should be added to the project and the following preprocessor symbols set correctly to select the particular variant of the library:

Symbol	Description
__SEGGER_RTL_BYTE_ORDER	Select the target's byte order.
__SEGGER_RTL_OPTIMIZE	Prefer size-optimized or speed-optimized code.

In many cases these can be configured automatically. For ARM the default configuration of the library is derived from these preprocessor symbols:

Symbol	Description
__thumb__	Assemble targeting the Thumb instruction set (as opposed to ARM).
__thumb2__	Assemble targeting the Thumb-2 instruction set.
__ARCH_V3	Assemble targeting ARMv3.
__ARCH_V4	Assemble targeting ARMv4.
__ARCH_V4T	Assemble targeting ARMv4T.
__ARCH_V5TE	Assemble targeting for ARMv5TE.
__ARCH_V6M	Assemble targeting for ARMv6-M.
__ARCH_V7M	Assemble targeting for ARMv7-M.
__ARCH_V7EM	Assemble targeting for ARMv7E-M.
__FP_ABI_HARD__	Assemble targeting the hard floating point ABI.
__FPU_VFP__	Assemble targeting the vector FPU.
__FPV4_SP_D16__	Assemble targeting the single-precision v4 FPU.
__FPV5_SP_D16__	Assemble targeting the single-precision v5 FPU.
__LITTLE_ENDIAN	Target little-endian processors.
__BIG_ENDIAN	Target big-endian processors.

For RISC-V the default configuration of the library is derived from these preprocessor symbols:

Symbol	Description
__riscv	Target is RISC-V.
__riscv_abi_rve	Assemble targeting RV32E.
__riscv_compressed	Target has C extension.
__riscv_float_abi_soft	Target has neither F nor D extension.
__riscv_float_abi_single	Target has F extension.
__riscv_float_abi_double	Target has D and F extensions.
__riscv_mul	Target has M extension.
__riscv_muldiv	Target has M extension with divide support.
__riscv_xlen	Distinguish RV32 and RV64 targets.

2.3.1 The SEGGER Runtime Library configuration file

The configuration of SEGGER Floating-point Library is defined by the content of `__libc_conf.h` which is included by all C and assembly language source files.

2.3.2 __SEGGER_RTL_BYTE_ORDER

Set this to -1 for little-endian byte order and +1 for big-endian byte order.

> **Note**
>
> This library does not support big-endian RVI32 targets

2.3.3 __SEGGER_RTL_OPTIMIZE

Define the preprocessor symbol `__SEGGER_RTL_OPTIMIZE` to select size-optimized implementations for both C and assembly language code.

If this preprocessor symbol is undefined (the default) the library is configured to select balanced implementations.

Value	Description
-2	Favor size at the expense of speed.
-1	Favor size over speed.
0	Balanced.
+1	Favor speed over size.
+2	Favor speed at the expense of size.
+3	As +2 and inline all functions that make sense as inlined.

2.4 Example command lines for compilation

The following GCC command lines a re sufficient to build the floating-point library for RV32I-MAC with an ILP32 ABI, assuming gcc is the compiler driver:

```
gcc -mabi=ilp32 -march=rv32imac -c -x assembler-with-cpp floatasmops_rv.s
gcc -mabi=ilp32 -march=rv32imac -c -O3 floatops.c
```

Chapter 3

Standard C library API

3.1 Exponential and logarithm functions

Function	Description
sqrt()	Compute square root, double.
sqrtf()	Compute square root, float.
cbrt()	Compute cube root, double.
cbrtf()	Compute cube root, float.
exp()	Compute base-e exponential, double.
expf()	Compute base-e exponential, float.
exp2()	Compute base-2 exponential, double.
exp2f()	Compute base-2 exponential, float.
exp10()	Compute base-10 exponential, double.
exp10f()	Compute base-10 exponential, float.
frexp()	Set exponent, double.
frexpf()	Set exponent, float.
hypot()	Compute magnitude of complex, double.
hypotf()	Compute magnitude of complex, float.
log()	Compute natural logarithm, double.
logf()	Compute natural logarithm, float.
log10()	Compute common logarithm, double.
log10f()	Compute common logarithm, float.
ldexp()	Scale by power of two, double.
ldexpf()	Scale by power of two, float.
pow()	Raise to power, double.
powf()	Raise to power, float.
scalbn()	Scale, double.
scalbnf()	Scale, float.

3.1.1 sqrt()

Description

Compute square root, double.

Prototype

```
double sqrt(double x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return square root of x.

Additional information

`sqrt()` computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, `sqrt()` deviates and always uses IEC 60559 semantics.

3.1.2 sqrtf()

Description

Compute square root, float.

Prototype

```
float sqrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If x < 0, return NaN.
- Else, return square root of x.

Additional information

`sqrt()` computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, `sqrt()` deviates and always uses IEC 60559 semantics.

3.1.3 cbrt()

Description

Compute cube root, double.

Prototype

```
double cbrt(double x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

3.1.4 cbrtf()

Description

Compute cube root, float.

Prototype

```
float cbrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

3.1.5 exp()

Description

Compute base-e exponential, double.

Prototype

```
double exp(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.6 expf()

Description

Compute base-e exponential, float.

Prototype

```
float expf(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.7 exp2()

Description

Compute base-2 exponential, double.

Prototype

```
double exp2(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.8 exp2f()

Description

Compute base-2 exponential, float.

Prototype

```
float exp2f(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.9 exp10()

Description

Compute base-10 exponential, double.

Prototype

```
double exp10(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.10 exp10f()

Description

Compute base-10 exponential, float.

Prototype

```
float exp10f(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

3.1.11 frexp()

Description

Set exponent, double.

Prototype

```
double frexp(double   x,
             int    * exp);
```

Parameters

Parameter	Description
x	Floating value top operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f * pow(2, *exp)

Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

3.1.12 frexpf()

Description

Set exponent, float.

Prototype

```
float frexpf(float   x,
             int   * exp);
```

Parameters

Parameter	Description
x	Floating value top operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f * pow(2, *exp)

Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

3.1.13 hypot()

Description

Compute magnitude of complex, double.

Prototype

```
double hypot(double x,
             double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrt(x*x + y*y).

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

3.1.14 hypotf()

Description

Compute magnitude of complex, float.

Prototype

```
float hypotf(float x,
             float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrt(x*x + y*y).

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

3.1.15 log()

Description

Compute natural logarithm, double.

Prototype

```
double log(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x.

3.1.16 logf()

Description

Compute natural logarithm, float.

Prototype

```
float logf(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-e logarithm of x.

3.1.17 log10()

Description

Compute common logarithm, double.

Prototype

```
double log10(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

3.1.18 log10f()

Description

Compute common logarithm, float.

Prototype

```
float log10f(float x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return negative infinity.
- If x is positively infinite, return infinity.
- ELse, return base-10 logarithm of x.

3.1.19 ldexp()

Description

Scale by power of two, double.

Prototype

```
double ldexp(double x,
             int    n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

Return value

- If x is zero, return x;
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * 2 ^ n.

Additional information

Multiplies a floating-point number by an integral power of two.

See also

scalbn()

3.1.20 ldexpf()

Description

Scale by power of two, float.

Prototype

```
float ldexpf(float x,
             int   n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

Return value

- If x is zero, return x;
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * 2^n.

Additional information

Multiplies a floating-point number by an integral power of two.

See also

`scalbnf()`

3.1.21 pow()

Description

Raise to power, double.

Prototype

```
double pow(double x,
           double y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

3.1.22 powf()

Description

Raise to power, float.

Prototype

```
float powf(float x,
           float y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

3.1.23 scalbn()

Description

Scale, double.

Prototype

```
double scalbn(double x,
              int    n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of DBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * DBL_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of DBL_RADIX.

As floating-point arithmetic conforms to IEC 60559, DBL_RADIX is 2 and scalbn() is (in this implementation) identical to ldexp().

See also

ldexpf()

3.1.24 scalbnf()

Description

Scale, float.

Prototype

```
float scalbnf(float x,
              int   n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of FLT_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x * FLT_RADIX ^ n.

Additional information

Multiplies a floating-point number by an integral power of FLT_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT_RADIX is 2 and scalbnf() is (in this implementation) identical to ldexpf().

See also

ldexpf()

3.2 Trigonometric functions

Function	Description
sin()	Calculate sine, double.
sinf()	Calculate sine, float.
cos()	Calculate cosine, double.
cosf()	Calculate cosine, float.
tan()	Compute tangent, double.
tanf()	Compute tangent, float.
sinh()	Compute hyperbolic sine, double.
sinhf()	Compute hyperbolic sine, float.
cosh()	Compute hyperbolic cosine, double.
coshf()	Compute hyperbolic cosine, float.
tanh()	Compute hyperbolic tangent, double.
tanhf()	Compute hyperbolic tangent, float.

3.2.1 sin()

Description

Calculate sine, double.

Prototype

```
double sin(double x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

3.2.2 sinf()

Description

Calculate sine, float.

Prototype

```
float sinf(float x);
```

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

3.2.3 cos()

Description

Calculate cosine, double.

Prototype

```
double cos(double x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

3.2.4 cosf()

Description

Calculate cosine, float.

Prototype

```
float cosf(float x);
```

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

3.2.5 tan()

Description

Compute tangent, double.

Prototype

```
double tan(double x);
```

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

3.2.6 tanf()

Description

Compute tangent, float.

Prototype

```
float tanf(float x);
```

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

3.2.7 sinh()

Description

Compute hyperbolic sine, double.

Prototype

```
double sinh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

3.2.8 sinhf()

Description

Compute hyperbolic sine, float.

Prototype

```
float sinhf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

3.2.9 cosh()

Description

Compute hyperbolic cosine, double.

Prototype

```
double cosh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return +Inf.
- Else, return hyperbolic cosine of x.

3.2.10 coshf()

Description

Compute hyperbolic cosine, float.

Prototype

```
float coshf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return +Inf.
- Else, return hyperbolic cosine of x.

3.2.11 tanh()

Description

Compute hyperbolic tangent, double.

Prototype

```
double tanh(double x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

3.2.12 tanhf()

Description

Compute hyperbolic tangent, float.

Prototype

```
float tanhf(float x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

3.3 Inverse trigonometric functions

Function	Description
asin()	Compute inverse sine, double.
asinf()	Compute inverse sine, float.
acos()	Compute inverse cosine, double.
acosf()	Compute inverse cosine, float.
atan()	Compute inverse tangent, double.
atanf()	Compute inverse tangent, float.
atan2()	Compute inverse tangent, with quadrant, double.
atan2f()	Compute inverse tangent, with quadrant, float.
asinh()	Compute inverse hyperbolic sine, double.
asinhf()	Compute inverse hyperbolic sine, float.
acosh()	Compute inverse hyperbolic cosine, double.
acoshf()	Compute inverse hyperbolic cosine, float.
atanh()	Compute inverse hyperbolic tangent, double.
atanhf()	Compute inverse hyperbolic tangent, float.

3.3.1 asin()

Description

Compute inverse sine, double.

Prototype

```
double asin(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular sine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

3.3.2 asinf()

Description

Compute inverse sine, float.

Prototype

```
float asinf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular sine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

3.3.3 acos()

Description

Compute inverse cosine, double.

Prototype

```
double acos(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [0, Pi] radians.

3.3.4 acosf()

Description

Compute inverse cosine, float.

Prototype

```
float acosf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval [0, Pi] radians.

3.3.5 atan()

Description

Compute inverse tangent, double.

Prototype

```
double atan(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

3.3.6 atanf()

Description

Compute inverse tangent, float.

Prototype

```
float atanf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval [-Pi/2, Pi/2] radians.

3.3.7 atan2()

Description

Compute inverse tangent, with quadrant, double.

Prototype

```
double atan2(double y,
             double x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-Pi, +Pi] radians.

3.3.8 atan2f()

Description

Compute inverse tangent, with quadrant, float.

Prototype

```
float atan2f(float y,
             float x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval [-Pi, +Pi] radians.

3.3.9 asinh()

Description

Compute inverse hyperbolic sine, double.

Prototype

```
double asinh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

Additional information

Calculates the inverse hyperbolic sine of x.

3.3.10 asinhf()

Description

Compute inverse hyperbolic sine, float.

Prototype

```
float asinhf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

Additional information

Calculates the inverse hyperbolic sine of x.

3.3.11 acosh()

Description

Compute inverse hyperbolic cosine, double.

Prototype

```
double acosh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

- If x < 1, return NaN.
- If x is NaN, return x.
- Else, return inverse hyperbolic cosine of x.

Additional information

Calculates the non-negative inverse hyperbolic cosine of x.

3.3.12 acoshf()

Description

Compute inverse hyperbolic cosine, float.

Prototype

```
float acoshf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

- If x < 1, return NaN.
- If x is NaN, return x.
- Else, return inverse hyperbolic cosine of x.

Additional information

Calculates the non-negative inverse hyperbolic cosine of x.

3.3.13 atanh()

Description

Compute inverse hyperbolic tangent, double.

Prototype

```
double atanh(double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- If x = +/-1, return +/-infinity.
- Else, return inverse hyperbolic tangent of x.

Additional information

Calculates the non-negative inverse hyperbolic tangent of x.

3.3.14 atanhf()

Description

Compute inverse hyperbolic tangent, float.

Prototype

```
float atanhf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If |x| > 1, return NaN.
- If x = +/-1, return +/-infinity.
- Else, return inverse hyperbolic tangent of x.

Additional information

Calculates the non-negative inverse hyperbolic tangent of x.

3.4 Rounding and remainder functions

Function	Description
ceil()	Compute smallest integer not less than, double.
ceilf()	Compute smallest integer not less than, float.
floor()	Compute largest integer not greater than, double.
floorf()	Compute largest integer not greater than, float.
fmod()	Compute remainder after division, double.
fmodf()	Compute remainder after division, float.
modf()	Separate integer and fractional parts, double.
modff()	Separate integer and fractional parts, float.

3.4.1 ceil()

Description

Compute smallest integer not less than, double.

Prototype

```
double ceil(double x);
```

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

3.4.2 ceilf()

Description

Compute smallest integer not less than, float.

Prototype

```
float ceilf(float x);
```

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

3.4.3 floor()

Description

Compute largest integer not greater than, double.

Prototype

```
double floor(double x);
```

Parameters

Parameter	Description
x	Value to floor.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

3.4.4 floorf()

Description

Compute largest integer not greater than, float.

Prototype

```
float floorf(float x);
```

Parameters

Parameter	Description
x	Value to floor.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

3.4.5 fmod()

Description

Compute remainder after division, double.

Prototype

```
double fmod(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

3.4.6 fmodf()

Description

Compute remainder after division, float.

Prototype

```
float fmodf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

Additional information

Computes the floating-point remainder of x divided by y, i.e. the value x - i*y for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

3.4.7 modf()

Description

Separate integer and fractional parts, double.

Prototype

```
double modf(double   x,
            double * iptr);
```

Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

Return value

The signed fractional part of x.

Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and modf() returns the signed fractional part of x.

3.4.8 modff()

Description

Separate integer and fractional parts, float.

Prototype

```
float modff(float   x,
            float * iptr);
```

Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

Return value

The signed fractional part of x.

Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and modff() returns the signed fractional part of x.

3.5 Absolute value functions

Function	Description
fabs()	Compute absolute value, double.
fabsf()	Compute absolute value, float.

3.5.1 fabs()

Description

Compute absolute value, double.

Prototype

```
double fabs(double x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

3.5.2 fabsf()

Description

Compute absolute value, float.

Prototype

```
float fabsf(float x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

3.6 Fused multiply functions

Function	Description
fma()	Compute fused multiply-add, double.
fmaf()	Compute fused multiply-add, float.

3.6.1 fma()

Description

Compute fused multiply-add, double.

Prototype

```
double fma(double x,
           double y,
           double z);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

Return value

Return (x * y) + z.

3.6.2 fmaf()

Description

Compute fused multiply-add, float.

Prototype

```
float fmaf(float x,
           float y,
           float z);
```

Parameters

Parameter	Description
x	Multiplier.
y	Multiplicand.
z	Summand.

Return value

Return (x * y) + z.

3.7 Maximum, minimum, and positive difference functions

Function	Description
fmin()	Compute minimum, double.
fminf()	Compute minimum, float.
fmax()	Compute maximum, double.
fmaxf()	Compute maximum, float.

3.7.1 fmin()

Description

Compute minimum, double.

Prototype

```
double fmin(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

3.7.2 fminf()

Description

Compute minimum, float.

Prototype

```
float fminf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

3.7.3 fmax()

Description

Compute maximum, double.

Prototype

```
double fmax(double x,
            double y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

3.7.4 fmaxf()

Description

Compute maximum, float.

Prototype

```
float fmaxf(float x,
            float y);
```

Parameters

Parameter	Description
x	Value #1.
y	Value #2.

Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

Chapter 4

GNU libgcc library API

The GNU floating-point runtime ABI can be realized in C and optionally in assembly language.

The assembly language floating-point funnctions are contained in separate files:

- For RISC-V this is found in `floatasmops_rv.s`.

4.1 Floating arithmetic

Function	Description
__addsf3	Add, float.
__adddf3	Add, double.
__subsf3	Subtract, float.
__subdf3	Subtract, double.
__mulsf3	Multiply, float.
__muldf3	Multiply, double.
__divsf3	Divide, float.
__divdf3	Divide, double.

4.1.1 __addsf3()

Description

Add, float.

Prototype

```
float __addsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

4.1.2 __adddf3()

Description

Add, double.

Prototype

```
double __adddf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

4.1.3 __subsf3()

Description

Subtract, float.

Prototype

```
float __subsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

4.1.4 __subdf3()

Description

Subtract, double.

Prototype

```
double __subdf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

4.1.5 __mulsf3()

Description

Multiply, float.

Prototype

```
float __mulsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

4.1.6 __muldf3()

Description

Multiply, double.

Prototype

```
double __muldf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

4.1.7 __divsf3()

Description

Divide, float.

Prototype

```
float __divsf3(float x,
               float y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

4.1.8 __divdf3()

Description

Divide, double.

Prototype

```
double __divdf3(double x,
                double y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

4.2 Floating conversions

Function	Description
__fixsfsi	Convert float to int.
__fixdfsi	Convert double to int.
__fixsfdi	Convert float to long long.
__fixdfdi	Convert double to long long.
__fixunssfsi	Convert float to unsigned.
__fixunsdfsi	Convert double to unsigned.
__fixunssfdi	Convert float to unsigned long long.
__fixunsdfdi	Convert double to unsigned long long.
__floatsisf	Convert int to float.
__floatsidf	Convert int to double.
__floatdisf	Convert long long to float.
__floatdidf	Convert long long to double.
__floatunsisf	Convert unsigned to float.
__floatunsidf	Convert unsigned to double.
__floatundisf	Convert unsigned long long to float.
__floatundidf	Convert unsigned long long to double.
__extendsfdf2	Extend float to double.
__truncdfsf2	Truncate double to float.

4.2.1 __fixsfsi()

Description

Convert float to int.

Prototype

```
__SEGGER_RTL_I32 __fixsfsi(float x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.2 __fixdfsi()

Description

Convert double to int.

Prototype

```
__SEGGER_RTL_I32 __fixdfsi(double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.3 __fixsfdi()

Description

Convert float to long long.

Prototype

```
__SEGGER_RTL_I64 __fixsfdi(float f);
```

Parameters

Parameter	Description
f	Floating value to convert.

Return value

Integerized value.

Notes

The RV32 compiler converts a float to a 64-bit integer by calling runtime support to handle it.

4.2.4 __fixdfdi()

Description

Convert double to long long.

Prototype

```
__SEGGER_RTL_I64 __fixdfdi(double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for double to int64 conversion.

4.2.5 __fixunssfsi()

Description

Convert float to unsigned.

Prototype

```
__SEGGER_RTL_U32 __fixunssfsi(float x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

4.2.6 __fixunsdfsi()

Description

Convert double to unsigned.

Prototype

```
__SEGGER_RTL_U32 __fixunsdfsi(double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

4.2.7 __fixunssfdi()

Description

Convert float to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunssfdi(float f);
```

Parameters

Parameter	Description
f	Float value to convert.

Return value

Integerized value.

4.2.8 __fixunsdfdi()

Description

Convert double to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunsdfdi(double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

4.2.9 __floatsisf()

Description

Convert int to float.

Prototype

```
float __floatsisf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

4.2.10 __floatsidf()

Description

Convert int to double.

Prototype

```
double __floatsidf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

4.2.11 __floatdisf()

Description

Convert long long to float.

Prototype

```
float __floatdisf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

4.2.12 __floatdidf()

Description

Convert long long to double.

Prototype

```
double __floatdidf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

4.2.13 __floatunsisf()

Description

Convert unsigned to float.

Prototype

```
float __floatunsisf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

4.2.14 __floatunsidf()

Description

Convert unsigned to double.

Prototype

```
double __floatunsidf(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Unsigned value to convert.

Return value

Double value.

4.2.15 __floatundisf()

Description

Convert unsigned long long to float.

Prototype

```
float __floatundisf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Float value.

4.2.16 __floatundidf()

Description

Convert unsigned long long to double.

Prototype

```
double __floatundidf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Double value.

4.2.17 __extendsfdf2()

Description

Extend float to double.

Prototype

```
double __extendsfdf2(float x);
```

Parameters

Parameter	Description
x	Float value to extend.

Return value

Double value.

4.2.18 __truncdfsf2()

Description

Truncate double to float.

Prototype

```
float __truncdfsf2(double x);
```

Parameters

Parameter	Description
x	Double value to truncate.

Return value

Float value.

4.3 Floating comparisons

Function	Description
__eqsf2	Equal, float.
__eqdf2	Equal, double.
__nesf2	Not equal, float.
__nedf2	Not equal, double.
__ltsf2	Less than, float.
__ltdf2	Less than, double.
__lesf2	Less than or equal, float.
__ledf2	Less than or equal, double.
__gtsf2	Greater than, float.
__gtdf2	Greater than, double.
__gesf2	Greater than or equal, float.
__gedf2	Greater than or equal, double.

4.3.1 __eqsf2()

Description

Equal, float.

Prototype

```
int __eqsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

4.3.2 __eqdf2()

Description

Equal, double.

Prototype

```
int __eqdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

4.3.3 __nesf2()

Description

Not equal, float.

Prototype

```
int __nesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

4.3.4 __nedf2()

Description

Not equal, double.

Prototype

```
int __nedf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

4.3.5 __ltsf2()

Description

Less than, float.

Prototype

```
int __ltsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and a < b (GNU three-way boolean).

4.3.6 __ltdf2()

Description

Less than, double.

Prototype

```
int __ltdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and a < b (GNU three-way boolean).

4.3.7 __lesf2()

Description

Less than or equal, float.

Prototype

```
int __lesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≤ 0 if both operands are non-NaN and a < b (GNU three-way boolean).

4.3.8 __ledf2()

Description

Less than or equal, double.

Prototype

```
int __ledf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≤ 0 if both operands are non-NaN and a < b (GNU three-way boolean).

4.3.9 __gtsf2()

Description

Greater than, float.

Prototype

```
int __gtsf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and a > b (GNU three-way boolean).

4.3.10 __gtdf2()

Description

Greater than, double.

Prototype

```
int __gtdf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and a > b (GNU three-way boolean).

4.3.11 __gesf2()

Description

Greater than or equal, float.

Prototype

```
int __gesf2(float x,
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≥ 0 if both operands are non-NaN and a ≥ b (GNU three-way boolean).

4.3.12 __gedf2()

Description

Greater than or equal, double.

Prototype

```
int __gedf2(double x,
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return ≥ 0 if both operands are non-NaN and a ≥ b (GNU three-way boolean).

Chapter 5

Indexes

5.1 Index of types

5.2 Index of functions