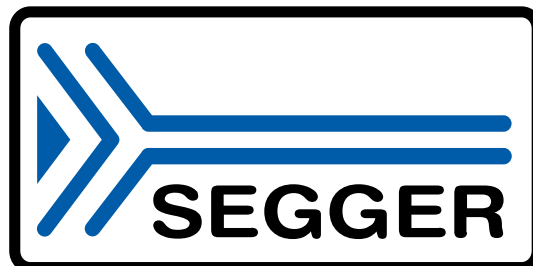


SEGGER Floating-point Library

A floating-point library
for microcontrollers

User Guide & Reference Manual

Document: UM12008
Software Version: 2.4.2
Revision: 0
Date: February 25, 2021



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

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2003-2021 SEGGER Microcontroller GmbH, Monheim am Rhein / Germany

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: February 25, 2021

Software	Revision	Date	By	Description
2.4.2	0	210225	PC	Chapter "C library API" <ul style="list-style-type: none"> • Added <code>trunc()</code>, <code>truncf()</code>. • Added <code>scalbln()</code>, <code>scalblnf()</code>.
2.20	0	200505	PC	Updated to latest software version.
2.12	0	191220	PC	Chapter "C library API" <ul style="list-style-type: none"> • Added <code>expm1f()</code>.
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.
<i>SEGGER home page</i>	A hyperlink to an external document or web site.

Table of contents

1	Introduction	12
1.1	What is the SEGGER Floating-point Library?	13
1.2	Features	13
1.3	Recommended project structure	14
1.4	Package content	15
1.4.1	Include directories	15
2	Compiling the SEGGER Floating-point Library	16
2.1	User-facing source files	17
2.2	Implementation source files	18
2.3	Configuring the library	19
2.3.1	The SEGGER Runtime Library configuration file	20
2.3.2	__SEGGER_RTL_BYTE_ORDER	21
2.3.3	__SEGGER_RTL_OPTIMIZE	22
2.3.4	__SEGGER_RTL_INCLUDE_C_API	23
2.3.5	__SEGGER_RTL_INCLUDE_SEGGER_API	24
2.3.6	__SEGGER_RTL_INCLUDE_AEABI_API	25
2.3.7	__SEGGER_RTL_INCLUDE_GNU_API	26
2.3.8	__SEGGER_RTL_NAN_FORMAT	27
2.3.9	__SEGGER_RTL_SCALED_INTEGER	28
2.3.10	__SEGGER_RTL_FP_HW	29
2.3.11	__SEGGER_RTL_UNLIKELY	30
2.3.12	__SEGGER_RTL_NEVER_INLINE	31
2.3.13	__SEGGER_RTL_ALWAYS_INLINE	32
2.3.14	__SEGGER_RTL_REQUEST_INLINE	33
2.3.15	__SEGGER_RTL_PUBLIC_API	34
2.4	Example configuration files	35
2.4.1	Arm configuraiton file	35
2.4.2	RISC-V configuraiton file	42
2.5	Example command lines for compilation	49
3	Standard C library API	50
3.1	Exponential and logarithm functions	51
3.1.1	sqrt()	52
3.1.2	sqrtf()	53
3.1.3	cbrt()	54
3.1.4	cbrtf()	55
3.1.5	exp()	56
3.1.6	expf()	57

3.1.7	exp2()	58
3.1.8	exp2f()	59
3.1.9	exp10()	60
3.1.10	exp10f()	61
3.1.11	expm1f()	62
3.1.12	frexp()	63
3.1.13	frexpf()	64
3.1.14	hypot()	65
3.1.15	hypotf()	66
3.1.16	log()	67
3.1.17	logf()	68
3.1.18	log2()	69
3.1.19	log2f()	70
3.1.20	log10()	71
3.1.21	log10f()	72
3.1.22	logb()	73
3.1.23	logbf()	74
3.1.24	ilogb()	75
3.1.25	ilogbf()	76
3.1.26	ldexp()	77
3.1.27	ldexpf()	78
3.1.28	ldexp()	79
3.1.29	ldexpf()	80
3.1.30	pow()	81
3.1.31	powf()	82
3.1.32	scalbn()	83
3.1.33	scalbnf()	84
3.1.34	scalbln()	85
3.1.35	scalblnf()	86
3.2	Trigonometric functions	87
3.2.1	sin()	88
3.2.2	sinf()	89
3.2.3	cos()	90
3.2.4	cosf()	91
3.2.5	tan()	92
3.2.6	tanf()	93
3.2.7	sinh()	94
3.2.8	sinhf()	95
3.2.9	cosh()	96
3.2.10	coshf()	97
3.2.11	tanh()	98
3.2.12	tanhf()	99
3.3	Inverse trigonometric functions	100
3.3.1	asin()	101
3.3.2	asinf()	102
3.3.3	acos()	103
3.3.4	acosf()	104
3.3.5	atan()	105
3.3.6	atanf()	106
3.3.7	atan2()	107
3.3.8	atan2f()	108
3.3.9	asinh()	109
3.3.10	asinhf()	110
3.3.11	acosh()	111
3.3.12	acoshf()	112
3.3.13	atanh()	113
3.3.14	atanhf()	114
3.4	Rounding and remainder functions	115
3.4.1	ceil()	116
3.4.2	ceilf()	117

3.4.3	floor()	118
3.4.4	floorf()	119
3.4.5	trunc()	120
3.4.6	truncf()	121
3.4.7	rint()	122
3.4.8	rintf()	123
3.4.9	round()	124
3.4.10	roundf()	125
3.4.11	nearbyint()	126
3.4.12	nearbyintf()	127
3.4.13	fmod()	128
3.4.14	fmodf()	129
3.4.15	modf()	130
3.4.16	modff()	131
3.4.17	remainder()	132
3.4.18	remainderf()	133
3.4.19	remquo()	134
3.4.20	remquof()	135
3.5	Absolute value functions	136
3.5.1	fabs()	137
3.5.2	fabsf()	138
3.6	Fused multiply functions	139
3.6.1	fma()	140
3.6.2	fmaf()	141
3.7	Maximum, minimum, and positive difference functions	142
3.7.1	fmin()	143
3.7.2	fminf()	144
3.7.3	fmax()	145
3.7.4	fmaxf()	146
3.7.5	fdim()	147
3.7.6	fdimf()	148
4	Arm AEABI library API	149
4.1	Floating arithmetic	149
4.1.1	__aeabi_fadd()	150
4.1.2	__aeabi_dadd()	151
4.1.3	__aeabi_fsub()	152
4.1.4	__aeabi_dsub()	153
4.1.5	__aeabi_frsub()	154
4.1.6	__aeabi_drsub()	155
4.1.7	__aeabi_fmul()	156
4.1.8	__aeabi_dmul()	157
4.1.9	__aeabi_fdiv()	158
4.1.10	__aeabi_ddiv()	159
4.2	Floating conversions	160
4.2.1	__aeabi_f2iz()	161
4.2.2	__aeabi_d2iz()	162
4.2.3	__aeabi_f2uiz()	163
4.2.4	__aeabi_d2uiz()	164
4.2.5	__aeabi_f2lz()	165
4.2.6	__aeabi_d2lz()	166
4.2.7	__aeabi_f2ulz()	167
4.2.8	__aeabi_d2ulz()	168
4.2.9	__aeabi_i2f()	169
4.2.10	__aeabi_i2d()	170
4.2.11	__aeabi_ui2f()	171
4.2.12	__aeabi_ui2d()	172
4.2.13	__aeabi_l2f()	173
4.2.14	__aeabi_l2d()	174

4.2.15	__aeabi_ul2f()	175
4.2.16	__aeabi_ul2d()	176
4.2.17	__aeabi_f2d()	177
4.2.18	__aeabi_d2f()	178
4.2.19	__aeabi_f2h()	179
4.2.20	__aeabi_d2h()	180
4.2.21	__aeabi_h2f()	181
4.2.22	__aeabi_f2h()	182
4.3	Floating comparisons	183
4.3.1	__aeabi_fcmpeq()	184
4.3.2	__aeabi_dcmpeq()	185
4.3.3	__aeabi_fcmlt()	186
4.3.4	__aeabi_dcmplt()	187
4.3.5	__aeabi_fcuple()	188
4.3.6	__aeabi_dcuple()	189
4.3.7	__aeabi_fcmpgt()	190
4.3.8	__aeabi_dcmpgt()	191
4.3.9	__aeabi_fcmpge()	192
4.3.10	__aeabi_dcmpge()	193
5	GNU libgcc library API	194
5.1	Floating arithmetic	194
5.1.1	__addsf3()	195
5.1.2	__adddf3()	196
5.1.3	__subsf3()	197
5.1.4	__subdf3()	198
5.1.5	__mulsf3()	199
5.1.6	__muldf3()	200
5.1.7	__divsf3()	201
5.1.8	__divdf3()	202
5.2	Floating conversions	203
5.2.1	__fixsfsi()	204
5.2.2	__fixdfsi()	205
5.2.3	__fixsfdi()	206
5.2.4	__fixdfdi()	207
5.2.5	__fixunssfsi()	208
5.2.6	__fixunsdfsi()	209
5.2.7	__fixunssfdi()	210
5.2.8	__fixunsdfdi()	211
5.2.9	__floatsisf()	212
5.2.10	__floatsidf()	213
5.2.11	__floatdisf()	214
5.2.12	__floatdidf()	215
5.2.13	__floatunsisf()	216
5.2.14	__floatunsidf()	217
5.2.15	__floatundisf()	218
5.2.16	__floatundidf()	219
5.2.17	__extendsfdf2()	220
5.2.18	__truncdfsf2()	221
5.3	Floating comparisons	222
5.3.1	__eqsf2()	223
5.3.2	__eqdf2()	224
5.3.3	__nesf2()	225
5.3.4	__nedf2()	226
5.3.5	__ltsf2()	227
5.3.6	__ltdf2()	228
5.3.7	__lesf2()	229
5.3.8	__ledf2()	230
5.3.9	__gtsf2()	231

5.3.10	__gtdf2()	232
5.3.11	__gesf2()	233
5.3.12	__gedf2()	234
6	Indexes	235
6.1	Index of types	236
6.2	Index of functions	237

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_FP.h	Interface to the SEGGER floating-point library.

In addition some private header files are required:

File	Description
__SEGGER_RTL_FP_Conf.h	Configuration of the library.
__SEGGER_RTL_FP_ConfDefault.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
<code>floatops.c</code>	Support for high-level floating point functions.
<code>floatasmops_arm.s</code>	Support for low-level floating point functions (ARM).
<code>floatasmops_rv.s</code>	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.
__SEGGER_RTL_INCLUDE_C_API	Control whether C standard entry points are included.
__SEGGER_RTL_INCLUDE_SEGGER_API	Control whether SEGGER entry points are included.
__SEGGER_RTL_INCLUDE_AEABI_API	Control whether Arm AEABI entry points are included.
__SEGGER_RTL_INCLUDE_GNU_API	Control whether GNU C entry points are included.
__SEGGER_RTL_NAN_FORMAT	Set the expected NaN format for the library.

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.
__ARM_PCS_VFP	Arm procedure calling standard for floating point.
__ARM_FP	Arm floating-point capability.
__ARM_ARCH	Arm architecture version.
__ARM_BIG_ENDIAN	Target byte order.
__GNUC__	Configuration for GNU C compiler.
__clang__	Configuration for Clang compiler.

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 `__SEGGER_RTL_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 RV32I 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.

2.3.4 `__SEGGER_RTL_INCLUDE_C_API`

This preprocessor symbol can be set as follows:

Value	Description
0	Exclude the standard C library API.
1	Include the standard C library API.

2.3.5 `__SEGGER_RTL_INCLUDE_SEGGER_API`

This preprocessor symbol can be set as follows:

Value	Description
0	Exclude the SEGGER floating-point API.
1	Include the SEGGER floating-point API.

The SEGGER floating-point API prefixes all standard C library symbols with `"SEGGER_"`.

2.3.6 `__SEGGER_RTL_INCLUDE_AEABI_API`

This preprocessor symbol can be set as follows:

Value	Description
0	Arm AEABI entry points are excluded.
1	Those Arm AEABI entry points that are capable of being coded in C are implemented using standard C functions. Functions that preclude C implementation (such as compares returning flags in the status register) are excluded.
2	All AEABI entry points are coded in assembly language.

This setting is intended for Arm architectures only but, as a C implementation does exist, it is possible to compile for the Arm AEABI on any architecture.

2.3.7 `__SEGGER_RTL_INCLUDE_GNU_API`

This preprocessor symbol can be set as follows:

Value	Description
0	GNU API entry points are excluded.
1	All GNU API entry points are implemented using standard C functions.
2	All GNU API entry points are coded in assembly language.

RV32 compilers universally adopt the GNU API for floating-point support and as such SEGGER provides RV32I and RV32E implementations of this API.

Arm compilers have migrated to the Arm AEABI and therefore no assembly language implementation exists for the deprecated GNU ABI on any Arm architecture.

2.3.8 `__SEGGER_RTL_NAN_FORMAT`

This preprocessor symbol can be set as follows:

Value	Description
<code>__SEGGER_RTL_NAN_FORMAT_AEABI_FAST</code>	Library adopts Arm fast NaNs.
<code>__SEGGER_RTL_NAN_FORMAT_IEEE</code>	Library adopts IEEE standard NaNs.

Arm fast NaNs for double-precision require that one of the significand bits in the high 32 bits of the NaN is set. This enables a software implementation to quickly distinguish double-precision Inf from NaN by using only the high-order 32 bits of the NaN.

2.3.9 `__SEGGER_RTL_SCALED_INTEGER`

Define the preprocessor symbol `__SEGGER_RTL_SCALED_INTEGER` to select scaled-integer algorithms over standard floating-point algorithms.

Value	Description
0	Algorithms use C-language floating-point arithmetic.
1	IEEE single-precision functions use scaled integer arithmetic if there is a scaled-integer implementation of the function.
+2	IEEE single-precision and double-precision functions use scaled integer arithmetic if there is a scaled-integer implementation of the function.

Note that selecting scaled-integer arithmetic does not reduce the range or accuracy of the function as seen by a user. Scaled-integer arithmetic runs quickly on integer-only processors and delivers results that are correctly rounded in more cases as 31 bits or 63 bits of precision are retained internally whereas using IEEE arithmetic retains only 24 or 53 bits of precision.

2.3.10 `__SEGGER_RTL_FP_HW`

This preprocessor symbol can be set as follows:

Value	Description
0	The target has no floating-point unit.
1	The target has a single-precision floating-point unit only.
2	The target has a floating-point unit supporting both single-precision and double-precision arithmetic.

This acts as an indication on how to optimize certain common arithmetic operations in the library. For instance, multiplying and dividing by a power of two can be tailored to whether it's faster to use inline arithmetic or to call `ldexp()`.

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_FP_HW
#define __SEGGER_RTL_FP_HW 0
#endif
```

2.3.11 `__SEGGER_RTL_UNLIKELY`

Indicate that the result of a condition is unlikely. The SEGGER floating-point library uses this when dealing with rare, special conditions, for example:

```
if (__SEGGER_RTL_UNLIKELY(__SEGGER_RTL_float32_isspecial(x))) {
```

For GNU C and Clang, the following definition suffices to direct the compiler:

```
//  
// Configuration of static branch probability.  
//  
#if defined(__GNUC__) || defined(__clang__)  
#define __SEGGER_RTL_UNLIKELY(X)      __builtin_expect((X), 0)  
#endif
```

The above definition will move special code off the hot path, and in doing so may increase code size slightly.

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_UNLIKELY  
#define __SEGGER_RTL_UNLIKELY(X)      (X)  
#endif
```

2.3.12 `__SEGGER_RTL_NEVER_INLINE`

Indicate that a function should never be inlined. The SEGGER floating-point library uses this when a function should never be inlined because code size will be adversely affected, or that the function use is sufficiently rare that it does not deserve inlining.

The definition is used like this:

```
static double __SEGGER_RTL_NEVER_INLINE
__SEGGER_RTL_float64_sqrt_outline(double x) {
    return __SEGGER_RTL_float64_sqrt_inline(x);
}
```

For GNU C and Clang, the following definition suffices to direct the compiler:

```
#if defined(__clang__)
    #define __SEGGER_RTL_NEVER_INLINE    __attribute__((__noinline__))
#else
    #define __SEGGER_RTL_NEVER_INLINE    __attribute__((__noinline__, __noclone__))
#endif
```

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_NEVER_INLINE
    #define __SEGGER_RTL_NEVER_INLINE
#endif
```

2.3.13 `__SEGGER_RTL_ALWAYS_INLINE`

Indicate that a function should always be inlined. The SEGGER floating-point library uses this when a function should always be inlined because it is trivial or code size will be adversely affected if the function is always called.

The definition is used like this:

```
static __SEGGER_RTL_ALWAYS_INLINE int
__SEGGER_RTL_float64_isfinite_inline(double x) {
```

For GNU C and Clang, the following definition suffices to direct the compiler:

```
#ifndef __SEGGER_RTL_ALWAYS_INLINE
#define __SEGGER_RTL_ALWAYS_INLINE
    __inline__ __attribute__((__always_inline__))
#endif
```

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_ALWAYS_INLINE
#define __SEGGER_RTL_ALWAYS_INLINE
#endif
```


2.3.14 `__SEGGER_RTL_REQUEST_INLINE`

Indicate that a function should be inlined using the compiler's default inlining strategy. The inlining strategy may well be altered when directing the compiler to optimize for size or for speed.

The definition is used like this:

```
static int __SEGGER_RTL_REQUEST_INLINE
__SEGGER_RTL_float64_isfinite_soft(__SEGGER_RTL_U64 x) {
```

For GNU C and Clang, the following definition suffices to direct the compiler:

```
#ifndef __SEGGER_RTL_REQUEST_INLINE
#define __SEGGER_RTL_REQUEST_INLINE __inline__
#endif
```

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_REQUEST_INLINE
#define __SEGGER_RTL_REQUEST_INLINE
#endif
```

2.3.15 `__SEGGER_RTL_PUBLIC_API`

Indicate that a function is part of the public API. This enables the compiler to set object file attributes for the link symbol.

The definition is used like this:

```
double __SEGGER_RTL_PUBLIC_API sin(double x) {  
    return __SEGGER_RTL_float64_sin_inline(x);  
}
```

For GNU C and Clang, the following definition suffices to make all API symbols weak:

```
#ifndef __SEGGER_RTL_PUBLIC_API  
#define __SEGGER_RTL_PUBLIC_API __attribute__((__weak__))  
#endif
```

The default definition, if not configured, is:

```
#ifndef __SEGGER_RTL_PUBLIC_API  
#define __SEGGER_RTL_PUBLIC_API  
#endif
```

2.4 Example configuration files

2.4.1 Arm configuraiton file

The following file is an example library configuration file for GNU C and Clang compilers targeting Arm processors:

```

/*****
 *
 *          (c) SEGGER Microcontroller GmbH
 *          The Embedded Experts
 *          www.segger.com
 *****/

----- END-OF-HEADER -----
*/

#ifndef __SEGGER_RTL_ARM_CONF_H
#define __SEGGER_RTL_ARM_CONF_H

/*****
 *
 *          Applicability checks
 *****/

*/

#if !defined(__ARM_ARCH_ISA_ARM) && !defined(__ARM_ARCH_ISA_THUMB)
#error This configuration file expects __ARM_ARCH_ISA_ARM or __ARM_ARCH_ISA_THUMB to be defined!
#endif

/*****
 *
 *          Defines, fixed
 *****/

*/

#define __SEGGER_RTL_ISA_T16      0
#define __SEGGER_RTL_ISA_T32     1
#define __SEGGER_RTL_ISA_ARM     2

/*****
 *
 *          Defines, configurable
 *****/

*/

#if defined(__thumb__) && !defined(__thumb2__)
#define __SEGGER_RTL_TARGET_ISA  __SEGGER_RTL_ISA_T16
#elif defined(__thumb2__)
#define __SEGGER_RTL_TARGET_ISA  __SEGGER_RTL_ISA_T32
#else
#define __SEGGER_RTL_TARGET_ISA  __SEGGER_RTL_ISA_ARM
#endif

//
// GCC and clang on ARM default to include the Arm AEABI
// with assembly speedups (2). Define this to 1 to use the
// C implementation.
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_INCLUDE_AEABI_API  2
#endif

//
// Configuration of byte order.
//
#if defined(__ARM_BIG_ENDIAN) && (__ARM_BIG_ENDIAN == 1)
#define __SEGGER_RTL_BYTE_ORDER      (+1)
#else
#define __SEGGER_RTL_BYTE_ORDER      (-1)
#endif

```

```

//
// Configuration of typeset.
//
#define __SEGGER_RTL_TYPESET          32

//
// Configuration of maximal type alignment
//
#define __SEGGER_RTL_MAX_ALIGN       8

//
// Configuration of floating-point ABI.
//
#if defined(__ARM_PCS_VFP) && (__ARM_PCS_VFP == 1)
    //
    // PCS uses hardware registers for passing parameters. For VFP
    // with only single-precision operations, parameters are still
    // passed in floating registers.
    //
    #define __SEGGER_RTL_FP_ABI        2
    //
#elif defined(__ARM_PCS) && (__ARM_PCS == 1)
    //
    // PCS is standard integer PCS.
    //
    #define __SEGGER_RTL_FP_ABI        0
    //
#else
    #error Unable to determine floating-point ABI used
#endif

//
// Configuration of floating-point hardware.
//
#if defined(__ARM_FP)
    //
    // Classification of Arm floating-point hardware.
    //
    #if (__ARM_FP & 0x08)
        #define __SEGGER_RTL_FP_HW     2
    #elif (__ARM_FP & 0x04)
        #define __SEGGER_RTL_FP_HW     1
    #else
        #define __SEGGER_RTL_FP_HW     0
    #endif
#else
    #define __SEGGER_RTL_FP_HW         0
#endif

//
// Configuration of full or Arm AEABI NaNs.
//
#if __SEGGER_RTL_FP_ABI == 0
    #define __SEGGER_RTL_NAN_FORMAT    __SEGGER_RTL_NAN_FORMAT_AEABI_FAST
#endif

//
// Configuration of floating constant selection.
//
#if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_FLT_SELECT(HEX, DEC)  HEX
#endif

//
// Configuration of multiplication capability.
//
//
// In the ARM Architecture Reference Manual, DDI 01001, Arm states
// the following for SMULL and UMULL:
//
// "Specifying the same register for either <RdHi> and <Rm>,
// or <RdLo> and <Rm>, was previously described as producing
// UNPREDICTABLE results. There is no restriction in ARMv6, and
// it is believed all relevant ARMv4 and ARMv5 implementations
// do not require this restriction either, because high
// performance multipliers read all their operands prior to

```

```

//   writing back any results."
//
// Unfortunately, the GNU assembler enforces this restriction which
// means that assembly-level inserts will not work for ARMv4 and
// ARMv5 even though there is no indication that they fail in
// practice.
//
#if __SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T16
//
// T16 ISA has no extended multiplication at all.
//
#define __SEGGER_RTL_CORE_HAS_EXT_MUL 0
//
#elif __ARM_ARCH >= 6
//
// ARMv6 and above have no restrictions on their input
// and output registers, so assembly-level inserts with
// constraints to guide the compiler are acceptable.
//
#define __SEGGER_RTL_CORE_HAS_EXT_MUL 1
//
#elif (__ARM_ARCH == 5) && defined(__clang__)
//
// Take Arm at its word and disable restrictions on input
// and output registers.
//
#define __SEGGER_RTL_CORE_HAS_EXT_MUL 1
//
#else
//
// ARMv5TE and lower have restrictions on their input
// and output registers, therefore do not enable extended
// multiply inserts.
//
#define __SEGGER_RTL_CORE_HAS_EXT_MUL 0
//
#endif

#if __SEGGER_RTL_CORE_HAS_EXT_MUL && (defined(__GNUC__) || defined(__clang__))
//
// v6+DSP and v7E-M has SMMUL instruction, others do not.
//
// Benchmarking using GCC and Clang/LLVM shows that using SMULL results in faster
// code than SMMUL. Using SMMUL results in marginally smaller code because there
// is less register pressure (no requirement to allocate a bit-bucket register).
//
// Given this, we disable the detection of SMMUL and use SMULL always.
//
#if 0 && (__ARM_ARCH >= 6) && defined(__ARM_FEATURE_DSP) && (__ARM_FEATURE_DSP == 1)
#define __SEGGER_RTL_SMMUL_HI(x0, x1) \
({ long __hi; \
__asm__( "smmul %0, %1, %2" \
: "=r"(__hi) \
: "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
__hi; \
})
#else
#define __SEGGER_RTL_SMMUL_HI(x0, x1) \
({ long __trash, __hi; \
__asm__( "smull %0, %1, %2, %3" \
: "=r"(__trash), "=r"(__hi) \
: "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
__hi; \
})
#endif

#define __SEGGER_RTL_SMMUL(lo, hi, x0, x1) \
do { \
__asm__( "smmul %0, %1, %2, %3" \
: "=r"(lo), "=r"(hi) \
: "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
} while (0)

#define __SEGGER_RTL_SMLAL(lo, hi, x0, x1) \
do { \
__asm__( "smlal %0, %1, %2, %3" \

```

```

        : "+r"(lo), "+r"(hi) \
        : "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
    } while (0)

#define __SEGGER_RTL_UMULL_HI(x0, x1) \
({ unsigned long __trash, __hi; \
  __asm__( "umull %0, %1, %2, %3" \
    : "=r"(__trash), "=r"(__hi) \
    : "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
  __hi; \
})

#define __SEGGER_RTL_UMULL(lo, hi, x0, x1) \
do { \
  __asm__( "umull %0, %1, %2, %3" \
    : "=r"(lo), "=r"(hi) \
    : "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
} while (0)

#define __SEGGER_RTL_UMLAL(lo, hi, x0, x1) \
do { \
  __asm__("umlal %0, %1, %2, %3" \
    : "+r"(lo), "+r"(hi) \
    : "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
} while (0)

#endif

//
// Configuration of static branch probability.
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_UNLIKELY(X) __builtin_expect((X), 0)
#endif

//
// Configuration of thread-local storage
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_THREAD __thread
#endif

//
// Configuration of inlining.
//
#if (defined(__GNUC__) || defined(__clang__)) && (__SEGGER_RTL_CONFIG_CODE_COVERAGE == 0)
#ifndef __SEGGER_RTL_NEVER_INLINE
//
// Clang doesn't know noclone...
//
#if defined(__clang__)
#define __SEGGER_RTL_NEVER_INLINE __attribute__((__noinline__))
#else
#define __SEGGER_RTL_NEVER_INLINE __attribute__((__noinline__, __noclone__))
#endif
#endif
#endif
//
#ifndef __SEGGER_RTL_ALWAYS_INLINE
#define __SEGGER_RTL_ALWAYS_INLINE __inline__ __attribute__((__always_inline__))
#endif
//
#ifndef __SEGGER_RTL_REQUEST_INLINE
#define __SEGGER_RTL_REQUEST_INLINE __inline__
#endif
//
#endif

//
// Configuration of public APIs.
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_PUBLIC_API __attribute__((__weak__))
#endif

//
// Using these builtins requires that the library is compiled

```

```

// with -fno-math-errno.
//
#if (__SEGGER_RTL_FP_HW >= 1) && (defined(__GNUC__) || defined(__clang__))
    #define __SEGGER_RTL_FLOAT32_ABS(X)    __builtin_fabsf(X)
#endif
#if (__SEGGER_RTL_FP_HW >= 2) && (defined(__GNUC__) || defined(__clang__))
    #define __SEGGER_RTL_FLOAT64_ABS(X)    __builtin_fabs(X)
#endif

#if (__SEGGER_RTL_FP_HW >= 1) && (defined(__GNUC__) || defined(__clang__))
    #define __SEGGER_RTL_FLOAT32_SQRT(X)    __builtin_sqrtf(X)
#endif
#if (__SEGGER_RTL_FP_HW >= 2) && (defined(__GNUC__) || defined(__clang__))
    #define __SEGGER_RTL_FLOAT64_SQRT(X)    __builtin_sqrt(X)
#endif

//
// Configuration of CLZ support.
//
#if defined(__ARM_FEATURE_CLZ) && (__ARM_FEATURE_CLZ == 1)
    //
    // ForACLE-conforming C compilers that declare the architecture or
    // profile has a CLZ instruction, use that CLZ instruction.
    //
    #define __SEGGER_RTL_CLZ_U32(X)          __builtin_clz(X)
#endif

//
// For Arm architectures using GNU C or clang, the SEGGER RTL offers
// optimized versions written in GNU-compatible assembly language.
// Selection of them is made here.
//
#if defined(__SEGGER_RTL_COMPILING_LIBRARY) && __SEGGER_RTL_COMPILING_LIBRARY
    #if defined(__GNUC__) || defined(__clang__)
        #define strcpy(x, y)                __SEGGER_RTL_HIDE strcpy(x, y)
        #define strcmp(x, y)                __SEGGER_RTL_HIDE strcmp(x, y)
        #define strlen(x)                   __SEGGER_RTL_HIDE strlen(x)
        #define memcpy(x, y, z)             __SEGGER_RTL_HIDE memcpy(x, y, z)
        #define memset(x, y, z)             __SEGGER_RTL_HIDE memset(x, y, z)
    #endif
#endif

/*****
 *
 *      Configuration of core features.
 *
 *****/

#if defined(__ARM_FEATURE_CLZ) && (__ARM_FEATURE_CLZ == 1)
    #define __SEGGER_RTL_CORE_HAS_CLZ        1
#else
    #define __SEGGER_RTL_CORE_HAS_CLZ        0
#endif

// ARMv8A/R only has ADDW in Thumb mode
#if (__ARM_ARCH >= 7) && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T32)
    #define __SEGGER_RTL_CORE_HAS_ADDW_SUBW  1
#else
    #define __SEGGER_RTL_CORE_HAS_ADDW_SUBW  0
#endif

#if __ARM_ARCH >= 7
    #define __SEGGER_RTL_CORE_HAS_MOVW_MOVT  1
#else
    #define __SEGGER_RTL_CORE_HAS_MOVW_MOVT  0
#endif

#if (defined(__ARM_FEATURE_IDIV) && __ARM_FEATURE_IDIV) || (__ARM_ARCH_PROFILE == 'R')
    #define __SEGGER_RTL_CORE_HAS_IDIV        1
#else
    #define __SEGGER_RTL_CORE_HAS_IDIV        0
#endif

#if (__ARM_ARCH >= 7) && (__SEGGER_RTL_TARGET_ISA != __SEGGER_RTL_ISA_ARM)
    #define __SEGGER_RTL_CORE_HAS_CBZ_CBNZ    1

```

```

#else
  #define __SEGGER_RTL_CORE_HAS_CBZ_CBNZ    0
#endif

#if (__ARM_ARCH >= 7) && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T32)
  #define __SEGGER_RTL_CORE_HAS_TBB_TBH    1
#else
  #define __SEGGER_RTL_CORE_HAS_TBB_TBH    0
#endif

#if __ARM_ARCH >= 6
  #define __SEGGER_RTL_CORE_HAS_UXT_SXT    1
#else
  #define __SEGGER_RTL_CORE_HAS_UXT_SXT    0
#endif

#if (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T32) || (__ARM_ARCH >= 7)
  #define __SEGGER_RTL_CORE_HAS_BFC_BFI_BFX 1
#else
  #define __SEGGER_RTL_CORE_HAS_BFC_BFI_BFX 0
#endif

#if __ARM_ARCH >= 5
  #define __SEGGER_RTL_CORE_HAS_BLX_REG     1
#else
  #define __SEGGER_RTL_CORE_HAS_BLX_REG     0
#endif

#if (__ARM_ARCH >= 6) && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T32)
  #define __SEGGER_RTL_CORE_HAS_LONG_SHIFT 1
#else
  #define __SEGGER_RTL_CORE_HAS_LONG_SHIFT 0
#endif

// Clang gets __ARM_FEATURE_CLZ wrong for v8M.Baseline, indicating
// that CLZ is available in hardware -- which it isn't. Patch that
// definition up here.
#if (__ARM_ARCH == 8) && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T16)
  #undef __SEGGER_RTL_CORE_HAS_CLZ
  #define __SEGGER_RTL_CORE_HAS_CLZ        0
#endif

// GCC gets __ARM_FEATURE_CLZ wrong for v5TE compiling for Thumb,
// indicating that CLZ is available in hardware -- which it isn't.
// Patch that definition up here.
#if (__ARM_ARCH == 5) && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T16)
  #undef __SEGGER_RTL_CORE_HAS_CLZ
  #define __SEGGER_RTL_CORE_HAS_CLZ        0
#endif

// Clang gets __ARM_FP wrong for the T16 target ISA indicating
// that floating-point instructions exist in this ISA -- which
// they don't. Patch that definition up here.
#if __ARM_ARCH_ISA_THUMB == 1
  #undef __SEGGER_RTL_FP_ABI
  #define __SEGGER_RTL_FP_ABI              0
  #undef __SEGGER_RTL_FP_HW
  #define __SEGGER_RTL_FP_HW              0
#endif

#ifndef __SEGGER_RTL_FAST_CODE_SECTION
  #if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_FAST_CODE_SECTION(X) __attribute__((__section__(".fast." X)))
  #endif
#endif

//
// GCC and clang provide a built-in support for _Complex.
//
#if defined(__GNUC__) || defined(__clang__)
  #ifndef __SEGGER_RTL_FLOAT32_C_COMPLEX
    #define __SEGGER_RTL_FLOAT32_C_COMPLEX float _Complex
  #endif
  #ifndef __SEGGER_RTL_FLOAT64_C_COMPLEX
    #define __SEGGER_RTL_FLOAT64_C_COMPLEX double _Complex
  #endif
#endif

```



```
#endif

//
// GCC and clang provide a built-in va_list.
//
#if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_VA_LIST      __builtin_va_list
#endif

//
// ARM C library ABI requires low-level assert function to be __aeabi_assert
//
#define __SEGGER_RTL_X_assert      __aeabi_assert

//
// ARM C library ABI defines how to interrogate errno
//
#define __SEGGER_RTL_X_errno_addr  __aeabi_errno_addr

#endif

/***** End of file *****/
```

2.4.2 RISC-V configuraiton file

The following file is an example library configuration file for GNU C targeting RISC-V RV32I and RV32E microcontrollers:

```

/*****
 *
 *          (c) SEGGER Microcontroller GmbH
 *          The Embedded Experts
 *          www.segger.com
 *****/

----- END-OF-HEADER -----
*/

#ifndef __SEGGER_RTL_RISCV_CONF_H
#define __SEGGER_RTL_RISCV_CONF_H

/*****
 *
 *          Applicability checks
 *****/

/*
 *
 *          Defines, fixed
 *****/

//
// Values returned when classifying floating values for RISC-V using fclass instruction
//
#define __SEGGER_RTL_RV_NEG_INF      (1u<<0)
#define __SEGGER_RTL_RV_NEG_NORMAL  (1u<<1)
#define __SEGGER_RTL_RV_NEG_SUBNORMAL (1u<<2)
#define __SEGGER_RTL_RV_NEG_ZERO    (1u<<3)
#define __SEGGER_RTL_RV_POS_ZERO    (1u<<4)
#define __SEGGER_RTL_RV_POS_SUBNORMAL (1u<<5)
#define __SEGGER_RTL_RV_POS_NORMAL  (1u<<6)
#define __SEGGER_RTL_RV_POS_INF     (1u<<7)
#define __SEGGER_RTL_RV_SNAN        (1u<<8)
#define __SEGGER_RTL_RV_QNAN        (1u<<9)

/*****
 *
 *          Defines, configurable
 *****/

//
// GCC and clang provide a built-in va_list.
//
#if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_VA_LIST      __builtin_va_list
#endif

//
// GCC and clang on RISC-V default to include the GNU libgcc API with assembly speedups.
//
#if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_INCLUDE_GNU_API 2
#endif

//

```

```

// Configuration of byte order.
//
#define __SEGGER_RTL_BYTE_ORDER          (-1)

//
// Configuration of typeset.
//
#define __SEGGER_RTL_TYPESET            __riscv_xlen

//
// Configuration of maximal type alignment
//
#define __SEGGER_RTL_MAX_ALIGN          16

//
// Configuration of floating-point ABI.
//
#if defined(__riscv_float_abi_soft)
#define __SEGGER_RTL_FP_ABI              0
#elif defined(__riscv_float_abi_single)
#define __SEGGER_RTL_FP_ABI              1
#elif defined(__riscv_float_abi_double)
#define __SEGGER_RTL_FP_ABI              2
#else
#error Cannot determine RISC-V floating-point ABI
#endif

//
// Configuration of floating-point hardware.
//
#if defined(__riscv_flen)
//
// Classification of RISC-V floating-point hardware.
//
#if __riscv_flen == 64
#define __SEGGER_RTL_FP_HW              2
#elif __riscv_flen == 32
#define __SEGGER_RTL_FP_HW              1
#else
#error Cannot determine RISC-V floating-point hardware
#endif
#endif

//
// Configuration of floating constant selection.
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_FLT_SELECT(HEX, DEC)  HEX
#endif

//
// Configuration of square root capability.
//
#if defined(__riscv_flen) && (__riscv_flen >= 32)
#define __SEGGER_RTL_FLOAT32_SQRT_FAST(X) \
({ float __result; \
__asm__("fsqrt.s %0, %1" : "=f"(__result) : "f"((float)(X))); \
__result; \
})
#endif

#if defined(__riscv_flen) && (__riscv_flen >= 64)
#define __SEGGER_RTL_FLOAT64_SQRT_FAST(X) \
({ double __result; \
__asm__("fsqrt.d %0, %1" : "=f"(__result) : "f"((double)(X))); \
__result; \
})
#endif

//
// Configuration of multiplication capability.
//
#if (__riscv_xlen == 32) && defined(__riscv_mul)
//
// RV32M (excludes F/D extensions)
//

```

```

#define __SEGGER_RTL_SMULL_HI(x0, x1) \
({ int __p; \
  __asm__("mulh %0, %1, %2" : "=r"(__p) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __p; \
})
#define __SEGGER_RTL_SMULL(lo, hi, x0, x1) \
do { unsigned __tmphi, __tmplo; \
  __asm__("mulh %0, %1, %2" : "=r"(__tmphi) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("mul %0, %1, %2" : "=r"(__tmplo) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  hi = __tmphi; lo = __tmplo; \
} while (0)
#define __SEGGER_RTL_SMLAL(lo, hi, x0, x1) \
do { \
  unsigned __tmp; \
  __asm__("mul %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("add %0, %0, %1" : "+r"(lo) : "r"(__tmp) ); \
  __asm__("sltu %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)lo), "r"((unsigned)__tmp) ); \
  __asm__("add %0, %0, %1" : "+r"(hi) : "r"(__tmp) ); \
  __asm__("mulh %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("add %0, %0, %1" : "+r"(hi) : "r"(__tmp) ); \
} while (0)
#define __SEGGER_RTL_UMULL_HI(x0, x1) \
({ unsigned __product; \
  __asm__("mulhu %0, %1, %2" : "=r"(__product) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __product; \
})
#define __SEGGER_RTL_UMULL(lo, hi, x0, x1) \
do { \
  __asm__("mulhu %0, %1, %2" : "=r"(hi) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("mul %0, %1, %2" : "=r"(lo) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
} while (0)
#define __SEGGER_RTL_UMLAL(lo, hi, x0, x1) \
do { \
  unsigned __tmp; \
  __asm__("mul %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("add %0, %0, %1" : "+r"(lo) : "r"(__tmp) ); \
  __asm__("sltu %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)lo), "r"((unsigned)__tmp) ); \
  __asm__("add %0, %0, %1" : "+r"(hi) : "r"(__tmp) ); \
  __asm__("mulhu %0, %1, %2" : "=r"(__tmp) : "r"((unsigned)(x0)), "r"((unsigned)(x1))); \
  __asm__("add %0, %0, %1" : "+r"(hi) : "r"((unsigned)__tmp) ); \
} while (0)
//
#endif

//
// Configuration of thread-local storage
//
#if defined(__GNUC__) || defined(__clang__)
#define __SEGGER_RTL_THREAD __thread
#endif

//
// Configuration of inlining.
//

```

```

#if defined(__GNUC__) || defined(__clang__) && (__SEGGER_RTL_CONFIG_CODE_COVERAGE == 0)
  #if defined(__clang__)
    #define __SEGGER_RTL_NEVER_INLINE    __attribute__((__noinline__))
    // clang does not support noclone.
  #else
    #define __SEGGER_RTL_NEVER_INLINE    __attribute__((__noinline__, __noclone__))
  #endif
  //
  #define __SEGGER_RTL_ALWAYS_INLINE    __inline__ __attribute__((__always_inline__))
  #define __SEGGER_RTL_REQUEST_INLINE  __inline__
#endif

//
// Configuration of static branch probability.
//
#if defined(__GNUC__) || defined(__clang__)
  #define __SEGGER_RTL_UNLIKELY(X)      __builtin_expect((X), 0)
#endif

//
// Configuration of division-remainder.
//
#if defined(__GNUC__)
  #define __SEGGER_RTL_DIVMOD_U32(Q, R, N, D) \
  do { \
    Q = N / D; \
    __asm__("# %0" : "+r"(Q)); /* Poison remainder rewrite by leaving \
                               compiler clueless as to the value \
                               contained in Q. */ \
    R = N - Q*D; \
  } while (0)

  #define __SEGGER_RTL_DIVMOD_U64(Q, R, N, D) \
  do { \
    Q = N / D; \
    __asm__("# %0" : "+r"(Q)); /* Poison remainder rewrite by leaving \
                               compiler clueless as to the value \
                               contained in Q. */ \
    R = N - Q*D; \
  } while (0)
#endif

//
// Configuration of floating-point inquiry functions.
//
#if defined(__riscv_flen) && (__riscv_flen >= 32) && !__SEGGER_RTL_FORCE_SOFT_FLOAT
  #if defined(__GNUC__) || defined(__clang__)

    #define __SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) \
    ({ unsigned __cls; \
      __asm__("fclass.s %0,%1" : "=r"(__cls) : "f"((float)(X))); \
      __cls; \
    })

    #define __SEGGER_RTL_FLOAT32_ISSPECIAL(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & ( __SEGGER_RTL_RV_NEG_ZERO \
                                              | __SEGGER_RTL_RV_POS_ZERO \
                                              | __SEGGER_RTL_RV_NEG_SUBNORMAL \
                                              | __SEGGER_RTL_RV_POS_SUBNORMAL \
                                              | __SEGGER_RTL_RV_NEG_INF \
                                              | __SEGGER_RTL_RV_POS_INF \
                                              | __SEGGER_RTL_RV_QNAN \
                                              | __SEGGER_RTL_RV_SNAN))

    #define __SEGGER_RTL_FLOAT32_ISSPECIAL_OR_NEGATIVE(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & ( __SEGGER_RTL_RV_NEG_ZERO \
                                              | __SEGGER_RTL_RV_POS_ZERO \
                                              | __SEGGER_RTL_RV_NEG_SUBNORMAL \
                                              | __SEGGER_RTL_RV_POS_SUBNORMAL \
                                              | __SEGGER_RTL_RV_NEG_NORMAL \
                                              | __SEGGER_RTL_RV_NEG_INF \
                                              | __SEGGER_RTL_RV_POS_INF \
                                              | __SEGGER_RTL_RV_QNAN \
                                              | __SEGGER_RTL_RV_SNAN))

    #define __SEGGER_RTL_FLOAT32_ISINF(X) \

```

```

(__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_INF | \
    __SEGGER_RTL_RV_POS_INF)) \

#define __SEGGER_RTL_FLOAT32_ISPOSINF(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & __SEGGER_RTL_RV_POS_INF)

#define __SEGGER_RTL_FLOAT32_ISNEGINF(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & __SEGGER_RTL_RV_POS_INF)

#define __SEGGER_RTL_FLOAT32_ISNAN(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & (__SEGGER_RTL_RV_QNAN | \
    __SEGGER_RTL_RV_SNAN))

#define __SEGGER_RTL_FLOAT32_ISFINITE(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_ZERO | \
    __SEGGER_RTL_RV_POS_ZERO | \
    __SEGGER_RTL_RV_NEG_SUBNORMAL | \
    __SEGGER_RTL_RV_POS_SUBNORMAL | \
    __SEGGER_RTL_RV_NEG_NORMAL | \
    __SEGGER_RTL_RV_POS_NORMAL))

#define __SEGGER_RTL_FLOAT32_ISNORMAL(X) \
    (__SEGGER_RTL_RV_FLOAT32_CLASSIFY(X) & (__SEGGER_RTL_RV_POS_NORMAL | \
    __SEGGER_RTL_RV_NEG_NORMAL))

#define __SEGGER_RTL_FLOAT32_SIGNBIT_COPY(X, Y) \
    ({ float __out; \
    __asm__ ("fsgnj.s %0, %1, %2" : "=f"(__out) : "f"((float)(X)), "f"((float)(Y))); \
    __out; \
    })

#define __SEGGER_RTL_FLOAT32_SIGNBIT_XOR(X, Y) \
    ({ float __out; \
    __asm__ ("fsgnjx.s %0, %1, %2" : "=f"(__out) : "f"((float)(X)), "f"((float)(Y))); \
    __out; \
    })

#define __SEGGER_RTL_FLOAT32_ABS(X) __builtin_fabsf(X)
#define __SEGGER_RTL_FLOAT32_FMA(X, Y, Z) __builtin_fmaf(X, Y, Z)

#endif
#endif

#if defined(__riscv_flen) && (__riscv_flen >= 64) && !__SEGGER_RTL_FORCE_SOFT_FLOAT
    #if defined(__GNUC__) || defined(__clang__)

#define __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) \
    ({ unsigned __cls; \
    __asm__("fclass.d %0,%1" : "=r"(__cls) : "f"((double)(X))); \
    __cls; \
    })

#define __SEGGER_RTL_FLOAT64_ISSPECIAL(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_ZERO | \
    __SEGGER_RTL_RV_POS_ZERO | \
    __SEGGER_RTL_RV_NEG_SUBNORMAL | \
    __SEGGER_RTL_RV_POS_SUBNORMAL | \
    __SEGGER_RTL_RV_NEG_INF | \
    __SEGGER_RTL_RV_POS_INF | \
    __SEGGER_RTL_RV_QNAN | \
    __SEGGER_RTL_RV_SNAN)

#define __SEGGER_RTL_FLOAT64_ISSPECIAL_OR_NEGATIVE(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_ZERO | \
    __SEGGER_RTL_RV_POS_ZERO | \
    __SEGGER_RTL_RV_NEG_SUBNORMAL | \
    __SEGGER_RTL_RV_POS_SUBNORMAL | \
    __SEGGER_RTL_RV_NEG_NORMAL | \
    __SEGGER_RTL_RV_NEG_INF | \
    __SEGGER_RTL_RV_POS_INF | \
    __SEGGER_RTL_RV_QNAN | \
    __SEGGER_RTL_RV_SNAN)

#define __SEGGER_RTL_FLOAT64_ISINF(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_INF | \
    __SEGGER_RTL_RV_POS_INF)

```

```

#define __SEGGER_RTL_FLOAT64_ISNAN(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_QNAN | \
    __SEGGER_RTL_RV_SNAN)

#define __SEGGER_RTL_FLOAT64_ISFINITE(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_NEG_ZERO | \
    __SEGGER_RTL_RV_POS_ZERO | \
    __SEGGER_RTL_RV_NEG_SUBNORMAL | \
    __SEGGER_RTL_RV_POS_SUBNORMAL | \
    __SEGGER_RTL_RV_NEG_NORMAL | \
    __SEGGER_RTL_RV_POS_NORMAL)

#define __SEGGER_RTL_FLOAT64_ISNORMAL(X) \
    __SEGGER_RTL_RV_FLOAT64_CLASSIFY(X) & (__SEGGER_RTL_RV_POS_NORMAL | \
    __SEGGER_RTL_RV_NEG_NORMAL)

#define __SEGGER_RTL_FLOAT64_SIGNBIT_COPY(X, Y)
\
({ double __out;
\
    __asm__ ("fsgnj.d %0, %1, %2" : "=f"(__out) : "f"((double)(X)), "f"((double)(Y)));
\
    __out;
\
})

#define __SEGGER_RTL_FLOAT64_SIGNBIT_XOR(X, Y)
\
({ double __out;
\
    __asm__ ("fsgnjx.d %0, %1, %2" : "=f"(__out) : "f"((double)(X)), "f"((double)(Y)));
\
    __out;
\
})

#define __SEGGER_RTL_FLOAT64_ABS(X)    __builtin_fabs(X)
#define __SEGGER_RTL_FLOAT64_FMA(X, Y, Z) __builtin_fma(X, Y, Z)
#endif
#endif

//
// GCC and clang provide a built-in support for _Complex.
//
#if defined(__GNUC__) || defined(__clang__)
    #ifndef __SEGGER_RTL_FLOAT32_C_COMPLEX
        #define __SEGGER_RTL_FLOAT32_C_COMPLEX float _Complex
    #endif
    #ifndef __SEGGER_RTL_FLOAT64_C_COMPLEX
        #define __SEGGER_RTL_FLOAT64_C_COMPLEX double _Complex
    #endif
#endif

//
// Configuration of public APIs.
//
#if defined(__GNUC__) || defined(__clang__)
    #define __SEGGER_RTL_PUBLIC_API    __attribute__((__weak__))
#endif

//
// For RISC-V architectures using GNU C or clang, the SEGGER RTL offers
// optimized versions written in GNU-compatible assembly language.
// Selection of them is made here.
//
#if defined(__SEGGER_RTL_COMPILING_LIBRARY) && __SEGGER_RTL_COMPILING_LIBRARY && (__SEGGER_RTL_INCLUDE_GNU_A
= 1)
    #if defined(__GNUC__) || defined(__clang__)
        #define strcpy(x, y)    __SEGGER_RTL_HIDE(strcpy)(x, y)
        #define strcmp(x, y)    __SEGGER_RTL_HIDE(strcmp)(x, y)
        #define __mulsi3(x, y)  __SEGGER_RTL_HIDE(__mulsi3)(x, y)
        #define __divsi3(x, y)  __SEGGER_RTL_HIDE(__divsi3)(x, y)
        #define __modsi3(x, y)  __SEGGER_RTL_HIDE(__modsi3)(x, y)
        #define __udivsi3(x, y) __SEGGER_RTL_HIDE(__udivsi3)(x, y)
        #define __umodsi3(x, y) __SEGGER_RTL_HIDE(__umodsi3)(x, y)
    #endif

```

```

//
// RV32{E,I} defaults to assembly language implementation.
// RV64I defaults to C implementation.
//
#if __riscv_xlen == 32
#define __eqsf2(x, y)      __SEGGER_RTL_HIDE(__eqsf2)(x, y)
#define __nesf2(x, y)      __SEGGER_RTL_HIDE(__nesf2)(x, y)
#define __ltsf2(x, y)      __SEGGER_RTL_HIDE(__ltsf2)(x, y)
#define __lesf2(x, y)      __SEGGER_RTL_HIDE(__lesf2)(x, y)
#define __gtsf2(x, y)      __SEGGER_RTL_HIDE(__gtsf2)(x, y)
#define __gesf2(x, y)      __SEGGER_RTL_HIDE(__gesf2)(x, y)
#define __unordsf2(x, y)   __SEGGER_RTL_HIDE(__unordsf2)(x, y)
//
#define __eqdf2(x, y)      __SEGGER_RTL_HIDE(__eqdf2)(x, y)
#define __nedf2(x, y)      __SEGGER_RTL_HIDE(__nedf2)(x, y)
#define __ltdf2(x, y)      __SEGGER_RTL_HIDE(__ltdf2)(x, y)
#define __ledf2(x, y)      __SEGGER_RTL_HIDE(__ledf2)(x, y)
#define __gtdf2(x, y)      __SEGGER_RTL_HIDE(__gtdf2)(x, y)
#define __gedf2(x, y)      __SEGGER_RTL_HIDE(__gedf2)(x, y)
#define __unorddf2(x, y)   __SEGGER_RTL_HIDE(__unorddf2)(x, y)
//
#define __addsf3(x, y)     __SEGGER_RTL_HIDE(__addsf3)(x, y)
#define __subsf3(x, y)     __SEGGER_RTL_HIDE(__subsf3)(x, y)
#define __mulsf3(x, y)     __SEGGER_RTL_HIDE(__mulsf3)(x, y)
#define __divsf3(x, y)     __SEGGER_RTL_HIDE(__divsf3)(x, y)
//
#define __adddf3(x, y)     __SEGGER_RTL_HIDE(__adddf3)(x, y)
#define __subdf3(x, y)     __SEGGER_RTL_HIDE(__subdf3)(x, y)
#define __muldf3(x, y)     __SEGGER_RTL_HIDE(__muldf3)(x, y)
#define __divdf3(x, y)     __SEGGER_RTL_HIDE(__divdf3)(x, y)
//
#define __fixsfsi(x)       __SEGGER_RTL_HIDE(__fixsfsi)(x)
#define __fixsfdi(x)       __SEGGER_RTL_HIDE(__fixsfdi)(x)
#define __fixdfsi(x)       __SEGGER_RTL_HIDE(__fixdfsi)(x)
#define __fixdfdi(x)       __SEGGER_RTL_HIDE(__fixdfdi)(x)
#define __fixunssfsi(x)    __SEGGER_RTL_HIDE(__fixunssfsi)(x)
#define __fixunssfdi(x)    __SEGGER_RTL_HIDE(__fixunssfdi)(x)
#define __fixunsdfsi(x)    __SEGGER_RTL_HIDE(__fixunsdfsi)(x)
#define __fixunsdfdi(x)    __SEGGER_RTL_HIDE(__fixunsdfdi)(x)
//
#define __floatsisf(x)     __SEGGER_RTL_HIDE(__floatsisf)(x)
#define __floatsidf(x)     __SEGGER_RTL_HIDE(__floatsidf)(x)
#define __floatdisf(x)     __SEGGER_RTL_HIDE(__floatdisf)(x)
#define __floatdidf(x)     __SEGGER_RTL_HIDE(__floatdidf)(x)
#define __floatunsisf(x)   __SEGGER_RTL_HIDE(__floatunsisf)(x)
#define __floatunsidf(x)   __SEGGER_RTL_HIDE(__floatunsidf)(x)
#define __floatundisf(x)   __SEGGER_RTL_HIDE(__floatundisf)(x)
#define __floatundidf(x)   __SEGGER_RTL_HIDE(__floatundidf)(x)
//
#define __extendsfdf2(x)   __SEGGER_RTL_HIDE(__extendsfdf2)(x)
#define __truncdfsf2(x)    __SEGGER_RTL_HIDE(__truncdfsf2)(x)
#endif
#endif
#endif

/***** End of file *****/

```


2.5 Example command lines for compilation

The following GCC command lines are sufficient to build the floating-point library for RV32IMAC 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
<code>sqrt()</code>	Compute square root, double.
<code>sqrtf()</code>	Compute square root, float.
<code>cbrt()</code>	Compute cube root, double.
<code>cbrtf()</code>	Compute cube root, float.
<code>exp()</code>	Compute base-e exponential, double.
<code>expf()</code>	Compute base-e exponential, float.
<code>exp2()</code>	Compute base-2 exponential, double.
<code>exp2f()</code>	Compute base-2 exponential, float.
<code>exp10()</code>	Compute base-10 exponential, double.
<code>exp10f()</code>	Compute base-10 exponential, float.
<code>expm1f()</code>	Compute base-e exponential, modified, double.
<code>frexp()</code>	Split to significand and exponent, double.
<code>frexpf()</code>	Split to significand and exponent, float.
<code>hypot()</code>	Compute magnitude of complex, double.
<code>hypotf()</code>	Compute magnitude of complex, float.
<code>log()</code>	Compute natural logarithm, double.
<code>logf()</code>	Compute natural logarithm, float.
<code>log2()</code>	Compute base-2 logarithm, double.
<code>log2f()</code>	Compute base-2 logarithm, float.
<code>log10()</code>	Compute common logarithm, double.
<code>log10f()</code>	Compute common logarithm, float.
<code>logb()</code>	Radix-independent exponent, double.
<code>logbf()</code>	Radix-independent exponent, float.
<code>ilogb()</code>	Radix-independent exponent, double.
<code>ilogbf()</code>	Radix-independent exponent, float.
<code>ldexp()</code>	Scale by power of two, double.
<code>ldexpf()</code>	Scale by power of two, float.
<code>pow()</code>	Raise to power, double.
<code>powf()</code>	Raise to power, float.
<code>scalbn()</code>	Scale, double.
<code>scalbnf()</code>	Scale, float.
<code>scalbln()</code>	Scale, double.
<code>scalblnf()</code>	Scale, float.

3.1.1 `sqrt()`

Description

Compute square root, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	Value to compute base-2 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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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 expm1f()

Description

Compute base-e exponential, modified, double.

Prototype

```
float expm1f(float x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute exponential of.

Return value

- If `x` is NaN, return `x`.
- Else, return base-e exponential of `x` minus 1 ($e^{**x} - 1$).

3.1.12 frexp()

Description

Split to significand and exponent, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.
<code>exp</code>	Pointer to integer receiving the power-of-two exponent of <code>x</code> .

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 frexpf()

Description

Split to significand and exponent, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.
<code>exp</code>	Pointer to integer receiving the power-of-two exponent of <code>x</code> .

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.

3.1.14 hypot()

Description

Compute magnitude of complex, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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 hypotf()

Description

Compute magnitude of complex, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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.

3.1.16 log()

Description

Compute natural logarithm, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to compute logarithm of.

Return value

- If `x` = NaN, return `x`.
- If `x` < 0, return NaN.
- If `x` = 0, return -Inf.
- If `x` is +Inf, return +Inf.
- ELse, return base-e logarithm of `x`.

3.1.17 logf()

Description

Compute natural logarithm, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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`.

3.1.18 log2()

Description

Compute base-2 logarithm, double.

Prototype

```
double log2(double x);
```

Parameters

Parameter	Description
<code>x</code>	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 log2f()

Description

Compute base-2 logarithm, float.

Prototype

```
float log2f(float x);
```

Parameters

Parameter	Description
<code>x</code>	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`.

3.1.20 log10()

Description

Compute common logarithm, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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`.

3.1.21 log10f()

Description

Compute common logarithm, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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`.

3.1.22 logb()

Description

Radix-independent exponent, double.

Prototype

```
double logb(double x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.

Return value

- If `x` is zero, return `-Inf`.
- If `x` is infinite, return `+Inf`.
- If `x` is NaN, return NaN.
- Else, return integer part of $\log[\text{FLTRADIX}](x)$.

Additional information

Calculates the exponent of `x`, which is the integral part of the FLTRADIX-logarithm of `x`.

3.1.23 logbf()

Description

Radix-independent exponent, float.

Prototype

```
float logbf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.

Return value

- If `x` is zero, return -Inf.
- If `x` is infinite, return +Inf.
- If `x` is NaN, return NaN.
- Else, return integer part of $\log[\text{FLTRADIX}](x)$.

Additional information

Calculates the exponent of `x`, which is the integral part of the FLTRADIX-logarithm of `x`.

3.1.24 ilogb()

Description

Radix-independent exponent, double.

Prototype

```
int ilogb(double x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.

Return value

- If `x` is zero, return `FP_ILOGB0`.
- If `x` is NaN, return `FP_ILOGBNAN`.
- If `x` is infinite, return `MAX_INT`.
- Else, return integer part of $\log[\text{FLTRADIX}](x)$.

3.1.25 ilogbf()

Description

Radix-independent exponent, float.

Prototype

```
int ilogbf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to operate on.

Return value

- If `x` is zero, return `FP_ILOGB0`.
- If `x` is NaN, return `FP_ILOGBNAN`.
- If `x` is infinite, return `MAX_INT`.
- Else, return integer part of $\log[\text{FLTRADIX}](x)$.

3.1.26 ldexp()

Description

Scale by power of two, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of two to scale by.

Return value

- If `x` is +/-0, return `x`;
- If `x` is +/-Inf, 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()`

3.1.27 ldexpf()

Description

Scale by power of two, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	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()`

3.1.28 ldexp()

Description

Scale by power of two, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of two to scale by.

Return value

- If `x` is +/-0, return `x`;
- If `x` is +/-Inf, 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()`

3.1.29 ldexpf()

Description

Scale by power of two, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	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 * 2n`.

Additional information

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

See also

`scalbnf()`

3.1.30 pow()

Description

Raise to power, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Base.
<code>y</code>	Power.

Return value

Return `x` raised to the power `y`.

3.1.31 powf()

Description

Raise to power, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Base.
<code>y</code>	Power.

Return value

Return `x` raised to the power `y`.

3.1.32 scalbn()

Description

Scale, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of <code>DBL_RADIX</code> 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()`

3.1.33 scalbnf()

Description

Scale, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of <code>FLT_RADIX</code> 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()`

3.1.34 scalbn()

Description

Scale, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of <code>DBL_RADIX</code> 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()`

3.1.35 scalbnf()

Description

Scale, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value to scale.
<code>n</code>	Power of <code>FLT_RADIX</code> 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()`.

3.2 Trigonometric functions

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

3.2.1 sin()

Description

Calculate sine, double.

Prototype

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

Parameters

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

3.3.1 asin()

Description

Compute inverse sine, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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 $[0, \text{Pi}]$ radians.

3.3.4 acosf()

Description

Compute inverse cosine, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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 $[0, \text{Pi}]$ radians.

3.3.5 atan()

Description

Compute inverse tangent, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>y</code>	Rise value of angle.
<code>x</code>	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 $[-\text{Pi}, +\text{Pi}]$ radians.

3.3.8 atan2f()

Description

Compute inverse tangent, with quadrant, float.

Prototype

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

Parameters

Parameter	Description
<code>y</code>	Rise value of angle.
<code>x</code>	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 $[-\text{Pi}, +\text{Pi}]$ radians.

3.3.9 asinh()

Description

Compute inverse hyperbolic sine, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>ceil()</code>	Compute smallest integer not less than, double.
<code>ceilf()</code>	Compute smallest integer not less than, float.
<code>floor()</code>	Compute largest integer not greater than, double.
<code>floorf()</code>	Compute largest integer not greater than, float.
<code>trunc()</code>	Truncate to integer, double.
<code>truncf()</code>	Truncate to integer, float.
<code>rint()</code>	Round to nearest integer, double.
<code>rintf()</code>	Round to nearest integer, float.
<code>round()</code>	Round to nearest integer, double.
<code>roundf()</code>	Round to nearest integer, float.
<code>nearbyint()</code>	Round to nearest integer, double.
<code>nearbyintf()</code>	Round to nearest integer, float.
<code>fmod()</code>	Compute remainder after division, double.
<code>fmodf()</code>	Compute remainder after division, float.
<code>modf()</code>	Separate integer and fractional parts, double.
<code>modff()</code>	Separate integer and fractional parts, float.
<code>remainder()</code>	Compute remainder after division, double.
<code>remainderf()</code>	Compute remainder after division, float.
<code>remquo()</code>	Compute remainder after division, double.
<code>remquof()</code>	Compute remainder after division, float.

3.4.1 ceil()

Description

Compute smallest integer not less than, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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
<code>x</code>	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 trunc()

Description

Truncate to integer, double.

Prototype

```
double trunc(double x);
```

Parameters

Parameter	Description
<code>x</code>	Value to truncate.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return `x` with fractional part removed.

3.4.6 truncf()

Description

Truncate to integer, float.

Prototype

```
float truncf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Value to truncate.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return `x` with fractional part removed.

3.4.7 rint()

Description

Round to nearest integer, double.

Prototype

```
double rint(double x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`.

3.4.8 rintf()

Description

Round to nearest integer, float.

Prototype

```
float rintf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`.

3.4.9 round()

Description

Round to nearest integer, double.

Prototype

```
double round(double x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`, ties away from zero.

3.4.10 roundf()

Description

Round to nearest integer, float.

Prototype

```
float roundf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`, ties away from zero.

3.4.11 nearbyint()

Description

Round to nearest integer, double.

Prototype

```
double nearbyint(double x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`.

3.4.12 nearbyintf()

Description

Round to nearest integer, float.

Prototype

```
float nearbyintf(float x);
```

Parameters

Parameter	Description
<code>x</code>	Value to compute nearest integer of.

Return value

- If `x` is infinite, return `x`.
- If `x` is NaN, return `x`.
- Else, return the nearest integer value to `x`.

3.4.13 fmod()

Description

Compute remainder after division, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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`.

3.4.14 fmodf()

Description

Compute remainder after division, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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`.

3.4.15 modf()

Description

Separate integer and fractional parts, double.

Prototype

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

Parameters

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

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`.

3.4.16 modff()

Description

Separate integer and fractional parts, float.

Prototype

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

Parameters

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

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`.

3.4.17 remainder()

Description

Compute remainder after division, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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`.

3.4.18 remainderf()

Description

Compute remainder after division, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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`.

3.4.19 remquo()

Description

Compute remainder after division, double.

Prototype

```
double remquo(double x,
              double y,
              int * quo);
```

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	Value #2.
<code>quo</code>	Pointer to object that receives the integer part of <code>x</code> divided by <code>y</code> .

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`.

3.4.20 remquof()

Description

Compute remainder after division, float.

Prototype

```
float remquof(float x,
              float y,
              int * quo);
```

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	Value #2.
<code>quo</code>	Pointer to object that receives the integer part of <code>x</code> divided by <code>y</code> .

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`.

3.5 Absolute value functions

Function	Description
<code>fabs()</code>	Compute absolute value, double.
<code>fabsf()</code>	Compute absolute value, float.

3.5.1 fabs()

Description

Compute absolute value, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	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
<code>x</code>	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
<code>fma()</code>	Compute fused multiply-add, double.
<code>fmaf()</code>	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
<code>x</code>	Multiplicand.
<code>y</code>	Multiplier.
<code>z</code>	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
<code>x</code>	Multiplier.
<code>y</code>	Multiplicand.
<code>z</code>	Summand.

Return value

Return $(x * y) + z$.

3.7 Maximum, minimum, and positive difference functions

Function	Description
<code>fmin()</code>	Compute minimum, double.
<code>fminf()</code>	Compute minimum, float.
<code>fmax()</code>	Compute maximum, double.
<code>fmaxf()</code>	Compute maximum, float.
<code>fdim()</code>	Positive difference, double.
<code>fdimf()</code>	Positive difference, float.

3.7.1 fmin()

Description

Compute minimum, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	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
<code>x</code>	Value #1.
<code>y</code>	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
<code>x</code>	Value #1.
<code>y</code>	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
<code>x</code>	Value #1.
<code>y</code>	Value #2.

Return value

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

3.7.5 fdim()

Description

Positive difference, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	Value #2.

Return value

- If `x > y`, `x-y`.
- Else, `+0`.

3.7.6 fdimf()

Description

Positive difference, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Value #1.
<code>y</code>	Value #2.

Return value

- If `x > y`, `x-y`.
- Else, `+0`.

Chapter 4

Arm AEABI library API

The SEGGER Floating-point Library provides an implementation of the Arm AEABI functions.

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

- For Arm this is found in `floatasmops_arm.s`.

The interface to the AEABI functions differs from the standard calling convention when the hard-floating ABI is used: all floating-point AEABI functions receive their parameters in integer registers and return their results in integer registers.

4.1 Floating arithmetic

Function	Description
__aeabi_fadd	Add, float.
__aeabi_dadd	Add, double.
__aeabi_fsub	Subtract, float.
__aeabi_dsub	Subtract, double.
__aeabi_frsub	Reverse subtract, float.
__aeabi_drsub	Reverse subtract, double.
__aeabi_fmul	Multiply, float.
__aeabi_dmul	Multiply, double.
__aeabi_fdiv	Divide, float.
__aeabi_ddiv	Divide, double.

4.1.1 `__aeabi_fadd()`

Description

Add, float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_fadd(__SEGGER_RTL_U32 x,  
                              __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Augend.
<code>y</code>	Addend.

Return value

Sum.

4.1.2 `__aeabi_dadd()`

Description

Add, double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_dadd(__SEGGER_RTL_U64 x,  
                              __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Augend.
<code>y</code>	Addend.

Return value

Sum.

4.1.3 `__aeabi_fsub()`

Description

Subtract, float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_fsub(__SEGGER_RTL_U32 x,  
                              __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

4.1.4 `__aeabi_dsub()`

Description

Subtract, double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_dsub(__SEGGER_RTL_U64 x,  
                              __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

4.1.5 `__aeabi_frsub()`

Description

Reverse subtract, float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_frsub(__SEGGER_RTL_U32 x,  
                               __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

4.1.6 `__aeabi_drsub()`

Description

Reverse subtract, double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_drsub(__SEGGER_RTL_U64 x,  
                               __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

4.1.7 `__aeabi_fmul()`

Description

Multiply, float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_fmul(__SEGGER_RTL_U32 x,  
                              __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Multiplicand.
<code>y</code>	Multiplier.

Return value

Product.

4.1.8 `__aeabi_dmul()`

Description

Multiply, double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_dmul(__SEGGER_RTL_U64 x,  
                              __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Multiplicand.
<code>y</code>	Multiplier.

Return value

Product.

4.1.9 `__aeabi_fdiv()`

Description

Divide, float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_fdiv(__SEGGER_RTL_U32 x,  
                              __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Dividend.
<code>y</code>	Divisor.

Return value

Quotient.

4.1.10 `__aeabi_ddiv()`

Description

Divide, double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_ddiv(__SEGGER_RTL_U64 x,  
                              __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Dividend.
<code>y</code>	Divisor.

Return value

Quotient.

4.2 Floating conversions

Function	Description
<code>__aeabi_f2iz</code>	Convert <code>__SEGGER_RTL_U32</code> to int.
<code>__aeabi_d2iz</code>	Convert <code>__SEGGER_RTL_U64</code> to int.
<code>__aeabi_f2uiz</code>	Convert <code>__SEGGER_RTL_U32</code> to <code>__SEGGER_RTL_U32</code> .
<code>__aeabi_d2uiz</code>	Convert <code>__SEGGER_RTL_U64</code> to unsigned.
<code>__aeabi_f2lz</code>	Convert <code>__SEGGER_RTL_U32</code> to long long.
<code>__aeabi_d2lz</code>	Convert <code>__SEGGER_RTL_U64</code> to long long.
<code>__aeabi_f2ulz</code>	Convert <code>__SEGGER_RTL_U32</code> to <code>__SEGGER_RTL_U64</code> .
<code>__aeabi_d2ulz</code>	Convert <code>__SEGGER_RTL_U64</code> to <code>__SEGGER_RTL_U64</code> .
<code>__aeabi_i2f</code>	Convert int to float.
<code>__aeabi_i2d</code>	Convert int to double.
<code>__aeabi_ui2f</code>	Convert unsigned to float.
<code>__aeabi_ui2d</code>	Convert unsigned to double.
<code>__aeabi_l2f</code>	Convert long long to float.
<code>__aeabi_l2d</code>	Convert long long to double.
<code>__aeabi_ul2f</code>	Convert unsigned long long to float.
<code>__aeabi_ul2d</code>	Convert unsigned long long to double.
<code>__aeabi_f2d</code>	Extend float to double.
<code>__aeabi_d2f</code>	Truncate double to float.
<code>__aeabi_f2h</code>	Truncate float to IEEE half-precision float.
<code>__aeabi_d2h</code>	Truncate double to IEEE half-precision float.
<code>__aeabi_h2f</code>	Convert IEEE half-precision float to float.
<code>__aeabi_h2d</code>	Convert IEEE half-precision float to double.

4.2.1 `__aeabi_f2iz()`

Description

Convert `__SEGGER_RTL_U32` to int.

Prototype

```
__SEGGER_RTL_I32 __aeabi_f2iz(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

4.2.2 `__aeabi_d2iz()`

Description

Convert `__SEGGER_RTL_U64` to int.

Prototype

```
__SEGGER_RTL_I32 __aeabi_d2iz(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

4.2.3 `__aeabi_f2uiz()`

Description

Convert `__SEGGER_RTL_U32` to `__SEGGER_RTL_U32`.

Prototype

```
__SEGGER_RTL_U32 __aeabi_f2uiz(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

4.2.4 `__aeabi_d2uiz()`

Description

Convert `__SEGGER_RTL_U64` to unsigned.

Prototype

```
__SEGGER_RTL_U32 __aeabi_d2uiz(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Double value to convert.

Return value

Integerized value.

4.2.5 `__aeabi_f2lz()`

Description

Convert `__SEGGER_RTL_U32` to long long.

Prototype

```
__SEGGER_RTL_I64 __aeabi_f2lz(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

Notes

The RV32 compiler converts a `__SEGGER_RTL_U32` to a 64-bit integer by calling runtime support to handle it.

4.2.6 `__aeabi_d2lz()`

Description

Convert `__SEGGER_RTL_U64` to long long.

Prototype

```
__SEGGER_RTL_I64 __aeabi_d2lz(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for `__SEGGER_RTL_U64` to int64 conversion.

4.2.7 `__aeabi_f2ulz()`

Description

Convert `__SEGGER_RTL_U32` to `__SEGGER_RTL_U64`.

Prototype

```
__SEGGER_RTL_U64 __aeabi_f2ulz(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

4.2.8 `__aeabi_d2ulz()`

Description

Convert `__SEGGER_RTL_U64` to `__SEGGER_RTL_U64`.

Prototype

```
__SEGGER_RTL_U64 __aeabi_d2ulz(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

4.2.9 `__aeabi_i2f()`

Description

Convert int to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_i2f(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

4.2.10 `__aeabi_i2d()`

Description

Convert int to double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_i2d(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

4.2.11 `__aeabi_ui2f()`

Description

Convert unsigned to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_ui2f(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

4.2.12 `__aeabi_ui2d()`

Description

Convert unsigned to double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_ui2d(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Unsigned value to convert.

Return value

`__SEGGER_RTL_U64` value.

4.2.13 `__aeabi_l2f()`

Description

Convert long long to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_l2f(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

4.2.14 `__aeabi_l2d()`

Description

Convert long long to double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_l2d(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

4.2.15 `__aeabi_ul2f()`

Description

Convert unsigned long long to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_ul2f(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Unsigned long long value to convert.

Return value

`__SEGGER_RTL_U32` value.

4.2.16 `__aeabi_ul2d()`

Description

Convert unsigned long long to double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_ul2d(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Unsigned long long value to convert.

Return value

`__SEGGER_RTL_U64` value.

4.2.17 `__aeabi_f2d()`

Description

Extend float to double.

Prototype

```
__SEGGER_RTL_U64 __aeabi_f2d(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Floating value to extend.

Return value

`__SEGGER_RTL_U64` value.

4.2.18 `__aeabi_d2f()`

Description

Truncate double to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_d2f(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Double value to truncate.

Return value

Float value.

4.2.19 `__aeabi_f2h()`

Description

Truncate float to IEEE half-precision float.

Prototype

```
__SEGGER_RTL_U16 __aeabi_f2h(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Float value to truncate.

Return value

Float value.

4.2.20 `__aeabi_d2h()`

Description

Truncate double to IEEE half-precision float.

Prototype

```
__SEGGER_RTL_U16 __aeabi_d2h(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
<code>x</code>	Double value to truncate.

Return value

Half-precision value.

4.2.21 `__aeabi_h2f()`

Description

Convert IEEE half-precision float to float.

Prototype

```
__SEGGER_RTL_U32 __aeabi_h2f(__SEGGER_RTL_U16 x);
```

Parameters

Parameter	Description
<code>x</code>	Half-precision float.

Return value

Single-precision float.

4.2.22 `__aeabi_f2h()`

Description

Truncate float to IEEE half-precision float.

Prototype

```
__SEGGER_RTL_U16 __aeabi_f2h(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
<code>x</code>	Float value to truncate.

Return value

Float value.

4.3 Floating comparisons

Function	Description
__aeabi_fcmpeq	Equal, float.
__aeabi_dcmpeq	Equal, double.
__aeabi_fcmlt	Less than, float.
__aeabi_dcmlt	Less than, double.
__aeabi_fcmlt	Less than or equal, float.
__aeabi_dcmlt	Less than, double.
__aeabi_fcmpgt	Less than, float.
__aeabi_dcmpgt	Less than, double.
__aeabi_fcmpge	Less than, float.
__aeabi_dcmpge	Less than, double.

4.3.1 `__aeabi_fcmpeq()`

Description

Equal, float.

Prototype

```
int __aeabi_fcmpeq(__SEGGER_RTL_U32 x,  
                  __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

0 `x` is not equal to `y`.
1 `x` is equal to `y`.

4.3.2 `__aeabi_dcmpeq()`

Description

Equal, double.

Prototype

```
int __aeabi_dcmpeq(__SEGGER_RTL_U64 x,  
                  __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

0 `x` is not equal to `y`.
1 `x` is equal to `y`.

4.3.3 `__aeabi_fcmplt()`

Description

Less than, float.

Prototype

```
int __aeabi_fcmplt(__SEGGER_RTL_U32 x,  
                  __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

0 `x` is not less than `y`.
1 `x` is less than `y`.

4.3.4 `__aeabi_dcmplt()`

Description

Less than, double.

Prototype

```
int __aeabi_dcmplt(__SEGGER_RTL_U64 x,  
                  __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

0 `x` is not less than `y`.
1 `x` is less than `y`.

4.3.5 __aeabi_fcmlpe()

Description

Less than or equal, float.

Prototype

```
int __aeabi_fcmlpe(__SEGGER_RTL_U32 x,  
                  __SEGGER_RTL_U32 y);
```

Parameters

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

Return value

- 0 [x](#) is not less than or equal to [y](#).
- 1 [x](#) is less than or equal to [y](#).

4.3.6 __aeabi_dcmple()

Description

Less than, double.

Prototype

```
int __aeabi_dcmple(__SEGGER_RTL_U64 x,  
                  __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

- 0 `x` is not less than or equal to `y`.
- 1 `x` is less than or equal to `y`.

4.3.7 __aeabi_fcmpgt()

Description

Less than, float.

Prototype

```
int __aeabi_fcmpgt(__SEGGER_RTL_U32 x,  
                  __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

- 0 `x` is not greater than `y`.
- 1 `x` is greater than `y`.

4.3.8 `__aeabi_dcmpgt()`

Description

Less than, double.

Prototype

```
int __aeabi_dcmpgt(__SEGGER_RTL_U64 x,  
                  __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

- 0 `x` is not greater than `y`.
- 1 `x` is greater than `y`.

4.3.9 __aeabi_fcmpge()

Description

Less than, float.

Prototype

```
int __aeabi_fcmpge(__SEGGER_RTL_U32 x,  
                  __SEGGER_RTL_U32 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

- 0 `x` is not greater than or equal to `y`.
- 1 `x` is greater than or equal to `y`.

4.3.10 `__aeabi_dcmpge()`

Description

Less than, double.

Prototype

```
int __aeabi_dcmpge(__SEGGER_RTL_U64 x,  
                  __SEGGER_RTL_U64 y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

- 0 `x` is not greater than or equal to `y`.
- 1 `x` is greater than or equal to `y`.

Chapter 5

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 functions are contained in separate files:

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

5.1 Floating arithmetic

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

5.1.1 `__addsf3()`

Description

Add, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Augend.
<code>y</code>	Addend.

Return value

Sum.

5.1.2 __adddf3()

Description

Add, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Augend.
<code>y</code>	Addend.

Return value

Sum.

5.1.3 `__subsf3()`

Description

Subtract, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

5.1.4 `__subdf3()`

Description

Subtract, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Minuend.
<code>y</code>	Subtrahend.

Return value

Difference.

5.1.5 `__mulsf3()`

Description

Multiply, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Multiplicand.
<code>y</code>	Multiplier.

Return value

Product.

5.1.6 `__muldf3()`

Description

Multiply, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Multiplicand.
<code>y</code>	Multiplier.

Return value

Product.

5.1.7 `__divsf3()`

Description

Divide, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Dividend.
<code>y</code>	Divisor.

Return value

Quotient.

5.1.8 `__divdf3()`

Description

Divide, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Dividend.
<code>y</code>	Divisor.

Return value

Quotient.

5.2 Floating conversions

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

5.2.1 `__fixsfsi()`

Description

Convert float to int.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

5.2.2 `__fixdfsi()`

Description

Convert double to int.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

5.2.3 `__fixsfdi()`

Description

Convert float to long long.

Prototype

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

Parameters

Parameter	Description
<code>f</code>	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.

5.2.4 `__fixdfdi()`

Description

Convert double to long long.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for double to int64 conversion.

5.2.5 `__fixunssfsi()`

Description

Convert float to unsigned.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Float value to convert.

Return value

Integerized value.

5.2.6 `__fixunsdfsi()`

Description

Convert double to unsigned.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Float value to convert.

Return value

Integerized value.

5.2.7 `__fixunssfdi()`

Description

Convert float to unsigned long long.

Prototype

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

Parameters

Parameter	Description
<code>f</code>	Float value to convert.

Return value

Integerized value.

5.2.8 `__fixunsdfdi()`

Description

Convert double to unsigned long long.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Float value to convert.

Return value

Integerized value.

5.2.9 __floatsisf()

Description

Convert int to float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

5.2.10 `__floatsidf()`

Description

Convert int to double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

5.2.11 __floatdisf()

Description

Convert long long to float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

5.2.12 __floatdidf()

Description

Convert long long to double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

5.2.13 __floatunsisf()

Description

Convert unsigned to float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Integer value to convert.

Return value

Floating value.

5.2.14 __floatunsidf()

Description

Convert unsigned to double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Unsigned value to convert.

Return value

Double value.

5.2.15 __floatundisf()

Description

Convert unsigned long long to float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Unsigned long long value to convert.

Return value

Float value.

5.2.16 __floatundidf()

Description

Convert unsigned long long to double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Unsigned long long value to convert.

Return value

Double value.

5.2.17 `__extendsfdf2()`

Description

Extend float to double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Float value to extend.

Return value

Double value.

5.2.18 `__truncdfsf2()`

Description

Truncate double to float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Double value to truncate.

Return value

Float value.

5.3 Floating comparisons

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

5.3.1 `__eqsf2()`

Description

Equal, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.2 `__eqdf2()`

Description

Equal, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.3 `__nesf2()`

Description

Not equal, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.4 `__neqf2()`

Description

Not equal, double.

Prototype

```
int __neqf2(double x,  
            double y);
```

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.5 __ltsf2()

Description

Less than, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.6 `__ltdf2()`

Description

Less than, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.7 `__lesf2()`

Description

Less than or equal, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.8 `__ledf2()`

Description

Less than or equal, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.9 `__gtsf2()`

Description

Greater than, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.10 `__gtdf2()`

Description

Greater than, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.11 __gesf2()

Description

Greater than or equal, float.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

5.3.12 `__gedf2()`

Description

Greater than or equal, double.

Prototype

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

Parameters

Parameter	Description
<code>x</code>	Left-hand operand.
<code>y</code>	Right-hand operand.

Return value

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

Chapter 6

Indexes

6.1 Index of types

6.2 Index of functions

__adddf3, **196**
 __addsf3, **195**
 __aeabi_d2f, **178**
 __aeabi_d2h, **180**
 __aeabi_d2iz, **162**
 __aeabi_d2lz, **166**
 __aeabi_d2uiz, **164**
 __aeabi_d2ulz, **168**
 __aeabi_dadd, **151**
 __aeabi_dcmpeq, **185**
 __aeabi_dcmpge, **193**
 __aeabi_dcmpgt, **191**
 __aeabi_dcmples, **189**
 __aeabi_dcmples, **187**
 __aeabi_ddiv, **159**
 __aeabi_dmul, **157**
 __aeabi_drsub, **155**
 __aeabi_dsub, **153**
 __aeabi_f2d, **177**
 __aeabi_f2h, **179, 179**
 __aeabi_f2iz, **161**
 __aeabi_f2lz, **165**
 __aeabi_f2uiz, **163**
 __aeabi_f2ulz, **167**
 __aeabi_fadd, **150**
 __aeabi_fcmpeq, **184**
 __aeabi_fcmpge, **192**
 __aeabi_fcmpgt, **190**
 __aeabi_fcmples, **188**
 __aeabi_fcmples, **186**
 __aeabi_fdiv, **158**
 __aeabi_fmuls, **156**
 __aeabi_frsub, **154**
 __aeabi_fsub, **152**
 __aeabi_h2f, **181**
 __aeabi_i2d, **170**
 __aeabi_i2f, **169**
 __aeabi_l2d, **174**
 __aeabi_l2f, **173**
 __aeabi_ui2d, **172**
 __aeabi_ui2f, **171**
 __aeabi_ul2d, **176**
 __aeabi_ul2f, **175**
 __divdf3, **202**
 __divsf3, **201**
 __eqdf2, **224**
 __eqsf2, **223**
 __extendsfdf2, **220**
 __fixdfdi, **207**
 __fixdfsi, **205**
 __fixsfdi, **206**
 __fixsfsi, **204**
 __fixunsdfdi, **211**
 __fixunsdfsi, **209**
 __fixunssfdi, **210**
 __fixunssfsi, **208**
 __floatdidf, **215**
 __floatdisf, **214**
 __floatsidf, **213**
 __floatsisf, **212**
 __floatundidf, **219**
 __floatundisf, **218**
 __floatunsidf, **217**
 __floatunsisf, **216**
 __gedf2, **234**
 __gesf2, **233**
 __gtdf2, **232**
 __gtsf2, **231**
 __ledf2, **230**
 __lesf2, **229**
 __ltdf2, **228**
 __ltsf2, **227**
 __muldf3, **200**
 __mulsf3, **199**

`__nedf2`, **226**
`__nesf2`, **225**
`__subdf3`, **198**
`__subsf3`, **197**
`__truncdfsf2`, **221**
`acos`, **103**
`acosf`, **104**
`acosh`, **111**
`acoshf`, **112**
`asin`, **101**
`asinf`, **102**
`asinh`, **109**
`asinhf`, **110**
`atan`, **105**
`atan2`, **107**
`atan2f`, **108**
`atanf`, **106**
`atanh`, **113**
`atanhf`, **114**
`cbrt`, **54**
`cbrtf`, **55**
`ceil`, **116**
`ceilf`, **117**
`cos`, **90**
`cosf`, **91**
`cosh`, **96**
`coshf`, **97**
`exp`, **56**
`exp10`, **60**
`exp10f`, **61**
`exp2`, **58**
`exp2f`, **59**
`expf`, **57**
`expm1f`, **62**
`fabs`, **137**
`fabsf`, **138**
`fdim`, **147**
`fdimf`, **148**
`floor`, **118**
`floorf`, **119**
`fma`, **140**
`fmaf`, **141**
`fmax`, **145**
`fmaxf`, **146**
`fmin`, **143**
`fminf`, **144**
`fmod`, **128**
`fmodf`, **129**
`frexp`, **63**
`frexpf`, **64**
`hypot`, **65**
`hypotf`, **66**
`ilogb`, **75**
`ilogbf`, **76**
`ldexp`, **77, 77**
`ldexpf`, **78, 78**
`log`, **67**
`log10`, **71**
`log10f`, **72**
`log2`, **69**
`log2f`, **70**
`logb`, **73**
`logbf`, **74**
`logf`, **68**
`modf`, **130**
`modff`, **131**
`nearbyint`, **126**
`nearbyintf`, **127**
`pow`, **81**
`powf`, **82**
`remainder`, **132**
`remainderf`, **133**
`remquo`, **134**
`remquof`, **135**
`rint`, **122**
`rintf`, **123**

round, **124**
roundf, **125**
scalbln, **85**
scalblnf, **86**
scalbn, **83**
scalbnf, **84**
sin, 34, **88**
sinf, **89**
sinh, **94**
sinhf, **95**
sqrt, **52**
sqrtf, **53**
tan, **92**
tanf, **93**
tanh, **98**
tanhf, **99**
trunc, **120**
truncf, **121**