emFloat

A floating-point library
for microcontrollers

User Guide & Reference Manual

Document: UM12008
Software Version: 2.6.0
Revision: 0
Date: November 21, 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

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: November 21, 2021

Software	Revision	Date	By	Description
2.8.0	0	211028	PC	Updated to latest software version.
2.4.2	0	210225	PC	Chapter "C library API" • Added `trunc()`, `truncf()`. • Added `scalbln()`, `scalblnf()`.
2.20	0	200505	PC	Updated to latest software version.
2.12	0	191220	PC	Chapter "C library API" • Added `expm1f()`.
2.10	0	190307	PC	Release version.
1.00	0	190204	PC	Internal version.

About this document

Assumptions

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

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

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

How to use this manual

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

Typographic conventions for syntax

This manual uses the following typographic conventions:

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

Table of contents

Chapter 1

Introduction

This section presents an overview of emFloat, its structure, and its capabilities.

1.1 What is emFloat?

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

1.2 Features

emFloat 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 emFloat:

- 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 emFloat 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 emFloat by simply replacing the `LIB` directory. Your application files can be stored anywhere.

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

1.4 Package content

emFloat is provided in source code and contains everything needed. The following table shows the content of the emFloat Package:

Directory	Description
`Doc`	emFloat documentation.
`Src`	emFloat source code.

1.4.1 Include directories

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

- `Src`

> **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 emFloat if you have old files included and therefore mix different versions. If you keep emFloat 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 emFloat

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 emFloat

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

emFloat is delivered in a small number of files that must be added to your project before building:

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

2.3 Configuring the library

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

Symbol	Description
__SEGGER_RTL_BYTE_ORDER	Select the target's byte order.
__SEGGER_RTL_OPTIMIZE	Prefer size-optimized or speed-optimized code.
__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
Compiler identification	
__GNUC__	Compiler is GNU C.
__clang__	Compiler is Clang.
Target instruction set	
__thumb__	Target the Thumb instruction set (as opposed to ARM).
__thumb2__	Target the Thumb-2 instruction set.
ACLE definitions	
__ARM_ARCH	Arm target architecture version.
__ARM_ARCH_PROFILE	Arm architecture profile, if applicable.
__ARM_ARCH_ISA_ARM	Processor implements AArch32 instruction set.
__ARM_ARCH_ISA_THUMB	Processor implements Thumb instruction set.
__ARM_BIG_ENDIAN	Byte order is big endian.
__ARM_PCS	Functions use standard Arm PCS calling convention.
__ARM_PCS_VFP	Functions use Arm VFP calling convention.
__ARM_FP	Arm floating-point hardware availability.
__ARM_FEATURE_CLZ	Indicates existence of CLZ instruction.
__ARM_FEATURE_IDIV	Indicates existence of integer division instructions.

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	Target RV32E base instruction set.
__riscv_compressed	Target has C extension.
__riscv_float_abi_soft	Target has neither F nor D extension.

Symbol	Description
__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_div	Target has M extension with divide support.
__riscv_dsp	Target has P (packed SIMD) extension.
__riscv_zba	Target has Zba (shift-add) extension.
__riscv_zbb	Target has Zbb (CLZ, negated logic) extension.
__riscv_zbs	Target has Zbs (bt manipulation) extension.
__riscv_xlen	Register width.
__riscv_flen	Floating-point register width.
__nds_v5	Andes Performance Extension support.

2.3.1 The SEGGER Runtime Library configuration file

The configuration of emFloat 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

Default

There is no default; it must always be set by the user.

Description

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

> **Note**
>
> This library does not support big-endian RV32 targets

2.3.3 __SEGGER_RTL_OPTIMIZE

Default

```
#ifndef   __SEGGER_RTL_OPTIMIZE
  #define __SEGGER_RTL_OPTIMIZE                0
#endif
```

Description

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

Default

```
#ifndef   __SEGGER_RTL_INCLUDE_C_API
  #define __SEGGER_RTL_INCLUDE_C_API           1
#endif
```

Description

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

Default

```
#ifndef   __SEGGER_RTL_INCLUDE_SEGGER_API
  #define __SEGGER_RTL_INCLUDE_SEGGER_API       0
#endif
```

Description

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

Default

```
#ifndef   __SEGGER_RTL_INCLUDE_AEABI_API
  #define __SEGGER_RTL_INCLUDE_AEABI_API        0
#endif
```

Description

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

Default

```
#ifndef   __SEGGER_RTL_INCLUDE_GNU_API
  #define __SEGGER_RTL_INCLUDE_GNU_API          0
#endif
```

Description

This preprocessor symbol can be set as follows:

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

Default

```
#define __SEGGER_RTL_NAN_FORMAT_IEEE            0
#define __SEGGER_RTL_NAN_FORMAT_FAST            1
#define __SEGGER_RTL_NAN_FORMAT_COMPACT         2

#ifndef   __SEGGER_RTL_NAN_FORMAT
  #define __SEGGER_RTL_NAN_FORMAT               __SEGGER_RTL_NAN_FORMAT_IEEE
#endif
```

This preprocessor symbol can be set as follows:

Value	Description
__SEGGER_RTL_NAN_FORMAT_IEEE	Library adopts IEEE standard NaNs.
__SEGGER_RTL_NAN_FORMAT_FAST	Library adopts Arm fast NaNs.
__SEGGER_RTL_NAN_FORMAT_COMPACT	Library adopts 16-bit-significant compact NaNs.

At present, only __SEGGER_RTL_NAN_FORMAT_IEEE is supported.

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

Default

```
#ifndef   __SEGGER_RTL_SCALED_INTEGER
  #define __SEGGER_RTL_SCALED_INTEGER          0
#endif
```

Description

Define the preprocessor symbol __SEGGER_RTL_SCALED_INTEGER to select scaled-intger 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 aritmetic retains only 24 or 53 bits of precision.

Scaled-integer algorithms are faster than standard algorithms using the floating-point emulator, but can be significantly larger depending upon compiler optimization options.

2.3.10 __SEGGER_RTL_FP_HW

Default

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

Description

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 do 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

Default

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

Description

Indicate that the result of a condition is unlikely. emFloat 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.

2.3.12 __SEGGER_RTL_NEVER_INLINE

Default

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

Description

Indicate that a function should never be inlined. emFloat 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
```


2.3.13 __SEGGER_RTL_ALWAYS_INLINE

Default

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

Description

Indicate that a function should always be inlined. emFloat 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
```


2.3.14 __SEGGER_RTL_REQUEST_INLINE

Default

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

Description

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
```


2.3.15 __SEGGER_RTL_PUBLIC_API

Default

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

Description

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
```


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
*
**********************************************************************
*/

// Not all compilers define __ARM_ACLE so this is disabled for now
#if 0 && !defined(__ARM_ACLE)
  #error This configuration file expects and ACLE-conforming compiler for configuration!
#endif
#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(__SEGGER_RTL_NO_BUILTIN)
  #if defined(__clang__)
    #define __SEGGER_RTL_NO_BUILTIN
  #elif defined(__GNUC__)
    #define __SEGGER_RTL_NO_BUILTIN             __attribute__((optimize("-fno-tree-loop-
distribute-patterns")))
  #else
  #endif
#endif

#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) && (__ARM_FP & 0x08)
  #define __SEGGER_RTL_FP_HW                    2
#elif defined(__ARM_FP) && (__ARM_FP & 0x04)
  #define __SEGGER_RTL_FP_HW                    1
#else
  #define __SEGGER_RTL_FP_HW                    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_HW
  #define __SEGGER_RTL_FP_HW                    0
  #undef  __SEGGER_RTL_FP_ABI
  #define __SEGGER_RTL_FP_ABI                   0
#endif

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

//
// Configuration of floating constant selection.
```



```
//
#if defined(__GNUC__) || defined(__clang__)
  #define __SEGGER_RTL_FLT_SELECT(HEX, DEC)     HEX
#endif

//
// Configuration of multiply-add capability.
//
#if __SEGGER_RTL_TARGET_ISA != __SEGGER_RTL_ISA_T16
  #define __SEGGER_RTL_CORE_HAS_MLA             1
#else
  #define __SEGGER_RTL_CORE_HAS_MLA             0
#endif

//
// Configuration of multiply-subtract capability.
//
#if (__ARM_ARCH >= 7) && (__SEGGER_RTL_TARGET_ISA != __SEGGER_RTL_ISA_T16)
  #define __SEGGER_RTL_CORE_HAS_MLS             1
#else
  #define __SEGGER_RTL_CORE_HAS_MLS             0
#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_SMULL_HI(x0, x1)                      \
      ({ long __hi;                                            \
         __asm__(  "smmul %0, %1, %2"                          \
                 : "=r"(__hi)                                  \
                 : "r"((unsigned)(x0)), "r"((unsigned)(x1)) ); \
         __hi;                                                 \
      })
  #else
    #define __SEGGER_RTL_SMULL_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_SMULL(lo, hi, x0, x1)                 \
    do {                                                     \
      __asm__(  "smull %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_UMULL_X(x, y)                         \
    ((__SEGGER_RTL_U64)(__SEGGER_RTL_U32)(x) *               \
                       (__SEGGER_RTL_U32)(y))

  #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
  //
  #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)
  #define __SEGGER_RTL_CORE_HAS_CLZ             1
#else
  #define __SEGGER_RTL_CORE_HAS_CLZ             0
#endif

// Clang gets __ARM_FEATURE_CLZ wrong for v8M.Baseline, indicating
// that CLZ is available in this ISA  -- 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 this ISA -- 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

#if __SEGGER_RTL_CORE_HAS_CLZ
  //
  // For ACLE-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 strchr(x, y)                        __SEGGER_RTL_HIDE(strchr)(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 (__SEGGER_RTL_TARGET_ISA !
= __SEGGER_RTL_ISA_T16) && defined(__ARM_FEATURE_SIMD32) && __ARM_FEATURE_SIMD32
  #define __SEGGER_RTL_CORE_HAS_UQADD_UQSUB     1
#else
  #define __SEGGER_RTL_CORE_HAS_UQADD_UQSUB     0
#endif

#if defined(__ARM_ARCH) && (__ARM_ARCH >= 7)
  #define __SEGGER_RTL_CORE_HAS_REV             1
#else
  #define __SEGGER_RTL_CORE_HAS_REV             0
#endif

#if defined(__ARM_ARCH) && (__ARM_ARCH >= 6) && (__SEGGER_RTL_TARGET_ISA !
= __SEGGER_RTL_ISA_T16) && defined(__ARM_FEATURE_DSP) && (__ARM_FEATURE_DSP == 1)
  #define __SEGGER_RTL_CORE_HAS_PKHTB_PKHBT     1
#else
  #define __SEGGER_RTL_CORE_HAS_PKHTB_PKHBT     0
#endif

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

// Unfortunately the ACLE specifies "__ARM_FEATURE_IDIV is defined to 1 if the target
// has hardware support for 32-bit integer division in all available instruction sets."
```



```
// For v7R, there is typically no divide in the Arm instruction set but there is
// support for divide in the Thumb instruction set, so provide an exception here
// when targeting v7R in Thumb mode.
#if (__ARM_ARCH_PROFILE == 'R') && (__SEGGER_RTL_TARGET_ISA == __SEGGER_RTL_ISA_T32)
  #undef  __SEGGER_RTL_CORE_HAS_IDIV
  #define __SEGGER_RTL_CORE_HAS_IDIV            1
#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

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

#ifndef   __SEGGER_RTL_USE_FPU_FOR_IDIV
  #define __SEGGER_RTL_USE_FPU_FOR_IDIV         0
#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
  #ifndef   __SEGGER_RTL_LDOUBLE_C_COMPLEX
    #define __SEGGER_RTL_LDOUBLE_C_COMPLEX      long double _Complex
  #endif
#endif

#ifndef   __SEGGER_RTL_PREFER_BRANCH_FREE_CODE
  #define __SEGGER_RTL_PREFER_BRANCH_FREE_CODE  1
#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
*
**********************************************************************
*/

#if !defined(__riscv)
  #error This configuration file expects __riscv to be defined!
#elif !defined(__riscv_xlen) || (__riscv_xlen != 32 && __riscv_xlen != 64)
  #error This configuration file expects __riscv_xlen to be defined to 32 or 64!
#elif defined(__riscv_flen) && (__riscv_flen != 32 && __riscv_flen != 64)
  #error This configuration file expects __riscv_flen to be undefined or defined to 32 or 64!
#endif

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

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

/*********************************************************************
*
*       Configuration of compiler features.
*
**********************************************************************
*/

#if !defined(__SEGGER_RTL_NO_BUILTIN)
  #if defined(__clang__)
    #define __SEGGER_RTL_NO_BUILTIN
  #elif defined(__GNUC__)
    #define __SEGGER_RTL_NO_BUILTIN             __attribute__((optimize("-fno-tree-loop-
distribute-patterns")))
  #endif
#endif

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



```
#if defined(__riscv_abi_rve)
  #define __SEGGER_RTL_CORE_HAS_ISA_RVE               1
#else
  #define __SEGGER_RTL_CORE_HAS_ISA_RVE               0
#endif

#if defined(__riscv_dsp)
  #define __SEGGER_RTL_CORE_HAS_ISA_SIMD              1
#else
  #define __SEGGER_RTL_CORE_HAS_ISA_SIMD              0
#endif

#if defined(__nds_v5)
  #define __SEGGER_RTL_CORE_HAS_ISA_ANDES_V5          1
#else
  #define __SEGGER_RTL_CORE_HAS_ISA_ANDES_V5          0
#endif

#if defined(__riscv_mul)
  #define __SEGGER_RTL_CORE_HAS_MUL_MULH              1
#else
  #define __SEGGER_RTL_CORE_HAS_MUL_MULH              0
#endif

#if defined(__riscv_div)
  #define __SEGGER_RTL_CORE_HAS_DIV                   1
#else
  #define __SEGGER_RTL_CORE_HAS_DIV                   0
#endif

#if defined(__riscv_dsp)
  #define __SEGGER_RTL_CORE_HAS_CLZ32                 1
#else
  #define __SEGGER_RTL_CORE_HAS_CLZ32                 0
#endif

#if  defined(__riscv_zbb)
  #define __SEGGER_RTL_CORE_HAS_CLZ                   1
#else
  #define __SEGGER_RTL_CORE_HAS_CLZ                   0
#endif

#if defined(__riscv_zbb)
  #define __SEGGER_RTL_CORE_HAS_ANDN_ORN_XORN         1
#else
  #define __SEGGER_RTL_CORE_HAS_ANDN_ORN_XORN         0
#endif

#if defined(__riscv_zbs)
  #define __SEGGER_RTL_CORE_HAS_BSET_BCLR_BINV_BEXT   1
#else
  #define __SEGGER_RTL_CORE_HAS_BSET_BCLR_BINV_BEXT   0
#endif

#if defined(__riscv_zba)
  #define __SEGGER_RTL_CORE_HAS_SHxADD                1
#else
  #define __SEGGER_RTL_CORE_HAS_SHxADD                0
#endif

#ifndef   __SEGGER_RTL_CORE_HAS_FUSED_DIVREM
  #define __SEGGER_RTL_CORE_HAS_FUSED_DIVREM          0
#endif

#ifndef   __SEGGER_RTL_PREFER_BRANCH_FREE_CODE
  #define __SEGGER_RTL_PREFER_BRANCH_FREE_CODE        0
#endif

//
// Configuration of CLZ support.
//
#if __SEGGER_RTL_CORE_HAS_CLZ || __SEGGER_RTL_CORE_HAS_CLZ32
  #define __SEGGER_RTL_CLZ_U32(X)                     __builtin_clz(X)
#endif
```



```
//
// 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__)
  #if __riscv_xlen == 32
    #define __SEGGER_RTL_INCLUDE_GNU_API  2
  #else
    #define __SEGGER_RTL_INCLUDE_GNU_API  1
  #endif
#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) && (__riscv_flen == 64)
  #define __SEGGER_RTL_FP_HW                    2
#elif defined(__riscv_flen) && (__riscv_flen == 32)
  #define __SEGGER_RTL_FP_HW                    1
#else
  #define __SEGGER_RTL_FP_HW                    0
#endif

//
// Configuration of long double size
//
#ifndef   __SEGGER_RTL_SIZEOF_LDOUBLE
  #define __SEGGER_RTL_SIZEOF_LDOUBLE           16
#endif

//
// Configuration of full or compact NaNs.
//
#if __SEGGER_RTL_FP_ABI == 0
  #define __SEGGER_RTL_NAN_FORMAT               __SEGGER_RTL_NAN_FORMAT_IEEE
#endif

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



```
#endif

//
// Configuration of stack alignment.
//
#ifndef   __SEGGER_RTL_STACK_ALIGN
  #if __SEGGER_RTL_CORE_HAS_ISA_RVE
    #define __SEGGER_RTL_STACK_ALIGN            4    // 4-byte alignment for RV32E
  #else
    #define __SEGGER_RTL_STACK_ALIGN            16   // 16-byte alignment for RV32E
  #endif
#endif

//
// Configuration of multiplication capability.
//
#if (__riscv_xlen == 32) && __SEGGER_RTL_CORE_HAS_MUL_MULH
  //
  // RV32M
  //
  #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_SMULL_X(x0, x1)
       \
    ({ unsigned __thi, __tlo;
       \
      __SEGGER_RTL_SMULL(__tlo, __thi, x0, x1);
       \
      (__SEGGER_RTL_I64)(((__SEGGER_RTL_U64)__thi << 32) + __tlo);
       \
    })
  #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_NEG_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)                                                   \
    ({ float __out;                                                                     \
      __asm__ ("fabs.s %0, %1" : "=f"(__out) : "f"((float)(X)));                        \
      __out;                                                                            \
    })

  #define __SEGGER_RTL_FLOAT32_FMA(X, Y, Z)                                             \
    ({ float __out;                                                                     \
      __asm__ ("fmadd.s %0, %1, %2, %3" : "=f"(__out) : "f"((float)(X)),                \
                                                        "f"((float)(Y)),                \
                                                        "f"((float)(Z)));               \
      __out;                                                                            \
    })

  #define __SEGGER_RTL_FLOAT32_SQRT_FAST(X)                                             \
    ({ float __result;                                                                  \
      __asm__("fsqrt.s %0, %1" : "=f"(__result) : "f"((float)(X)));                     \
      __result;                                                                         \
    })
  #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)
  \
      ({ double __out;
  \
         __asm__ ("fabs.d %0, %1" : "=f"(__out) : "f"((double)(X)));
  \
         __out;
  \
      })

    #define __SEGGER_RTL_FLOAT64_FMA(X, Y, Z)
  \
      ({ double __out;
  \
         __asm__ ("fmadd.d %0, %1, %2, %3" : "=f"(__out) : "f"((double)(X)),
  \
                                                           "f"((double)(Y)),
  \
                                                           "f"((double)(Z)));
  \
         __out;
  \
      })

    #define __SEGGER_RTL_FLOAT64_SQRT_FAST(X)
  \
      ({ double __result;
  \
        __asm__("fsqrt.d %0, %1" : "=f"(__result) : "f"((double)(X)));
  \
        __result;
  \
      })
  #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
  #ifndef   __SEGGER_RTL_LDOUBLE_C_COMPLEX
    #define __SEGGER_RTL_LDOUBLE_C_COMPLEX      long 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 strlen(x)                           __SEGGER_RTL_HIDE(strlen)(x)
    #define strcpy(x, y)                        __SEGGER_RTL_HIDE(strcpy)(x, y)
    #define strcmp(x, y)                        __SEGGER_RTL_HIDE(strcmp)(x, y)
    #define strchr(x, y)                        __SEGGER_RTL_HIDE(strchr)(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)
    //
    // 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

#if (__SEGGER_RTL_STACK_ALIGN % 4) != 0
  #error Stack alignment must be a multiple of 4
```



```
#endif

#endif

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


2.5 Example command lines for compilation

The following GCC command lines are sufficient to build emFloat 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
sqrt()	Compute square root, double.
sqrtf()	Compute square root, float.
sqrtl()	Compute square root, long double.
cbrt()	Compute cube root, double.
cbrtf()	Compute cube root, float.
cbrtl()	Compute cube root, long double.
exp()	Compute base-e exponential, double.
expf()	Compute base-e exponential, float.
expl()	Compute base-e exponential, long double.
exp2()	Compute base-2 exponential, double.
exp2f()	Compute base-2 exponential, float.
exp2l()	Compute base-2 exponential, long double.
exp10()	Compute base-10 exponential, double.
exp10f()	Compute base-10 exponential, float.
exp10l()	Compute base-10 exponential, long double.
expm1f()	Compute base-e exponential, modified, float.
frexp()	Split to significand and exponent, double.
frexpf()	Split to significand and exponent, float.
frexpl()	Split to significand and exponent, long double.
hypot()	Compute magnitude of complex, double.
hypotf()	Compute magnitude of complex, float.
hypotl()	Compute magnitude of complex, long double.
log()	Compute natural logarithm, double.
logf()	Compute natural logarithm, float.
logl()	Compute natural logarithm, long double.
log2()	Compute base-2 logarithm, double.
log2f()	Compute base-2 logarithm, float.
log2l()	Compute base-2 logarithm, long double.
log10()	Compute common logarithm, double.
log10f()	Compute common logarithm, float.
log10l()	Compute common logarithm, long double.
logb()	Radix-indpendent exponent, double.
logbf()	Radix-indpendent exponent, float.
logbl()	Radix-indpendent exponent, long double.
ilogb()	Radix-independent exponent, double.
ilogbf()	Radix-independent exponent, float.
ilogbl()	Radix-independent exponent, long double.
ldexp()	Scale by power of two, double.
ldexpf()	Scale by power of two, float.
ldexpl()	Scale by power of two, long double.
pow()	Raise to power, double.

Function	Description
powf()	Raise to power, float.
powl()	Raise to power, long double.
scalbn()	Scale, double.
scalbnf()	Scale, float.
scalbnl()	Scale, long double.
scalbln()	Scale, double.
scalblnf()	Scale, float.
scalblnl()	Scale, long double.

3.1.1 sqrt()

Description

Compute square root, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

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

Additional information

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

3.1.2 sqrtf()

Description

Compute square root, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

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

Additional information

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

3.1.3 sqrtl()

Description

Compute square root, long double.

Prototype

```
long double sqrtl(long double x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

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

Additional information

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

3.1.4 cbrt()

Description

Compute cube root, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

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

3.1.5 cbrtf()

Description

Compute cube root, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

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

3.1.6 cbrtl()

Description

Compute cube root, long double.

Prototype

```
long double cbrtl(long double x);
```

Parameters

Parameter	Description
x	Value to compute cube root of.

Return value

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

3.1.7 exp()

Description

Compute base-e exponential, double.

Prototype

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

Parameters

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

Return value

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

3.1.8 expf()

Description

Compute base-e exponential, float.

Prototype

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

Parameters

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

Return value

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

3.1.9 expl()

Description

Compute base-e exponential, long double.

Prototype

```
long double expl(long double x);
```

Parameters

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

Return value

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

3.1.10 exp2()

Description

Compute base-2 exponential, double.

Prototype

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

Parameters

Parameter	Description
x	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.

3.1.11 exp2f()

Description

Compute base-2 exponential, float.

Prototype

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

Parameters

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

Return value

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

3.1.12 exp2l()

Description

Compute base-2 exponential, long double.

Prototype

```
long double exp2l(long double x);
```

Parameters

Parameter	Description
x	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.

3.1.13 exp10()

Description

Compute base-10 exponential, double.

Prototype

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

Parameters

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

Return value

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

3.1.14 exp10f()

Description

Compute base-10 exponential, float.

Prototype

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

Parameters

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

Return value

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

3.1.15 exp10l()

Description

Compute base-10 exponential, long double.

Prototype

```
long double exp10l(long double x);
```

Parameters

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

Return value

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

3.1.16 expm1()

Description

Compute base-e exponential, modified, double.

Prototype

```
double expm1(double x);
```

Parameters

Parameter	Description
x	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).

3.1.17 expm1f()

Description

Compute base-e exponential, modified, float.

Prototype

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

Parameters

Parameter	Description
x	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).

3.1.18 expm1l()

Description

Compute base-e exponential, modified, long double.

Prototype

```
long double expm1l(long double x);
```

Parameters

Parameter	Description
x	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).

3.1.19 frexp()

Description

Split to significand and exponent, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.1.20 frexpf()

Description

Split to significand and exponent, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.1.21 frexpl()

Description

Split to significand and exponent, long double.

Prototype

```
long double frexpl(long double   x,
                   int         * exp);
```

Parameters

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

Return value

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

Additional information

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

3.1.22 hypot()

Description

Compute magnitude of complex, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.1.23 hypotf()

Description

Compute magnitude of complex, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.1.24 hypotl()

Description

Compute magnitude of complex, long double.

Prototype

```
long double hypotl(long double x,
                   long double y);
```

Parameters

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

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrtl(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.

3.1.25 log()

Description

Compute natural logarithm, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.26 logf()

Description

Compute natural logarithm, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.27 logl()

Description

Compute natural logarithm, long double.

Prototype

```
long double logl(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.28 log2()

Description

Compute base-2 logarithm, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.29 log2f()

Description

Compute base-2 logarithm, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.30 log2l()

Description

Compute base-2 logarithm, long double.

Prototype

```
long double log2l(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.31 log10()

Description

Compute common logarithm, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.32 log10f()

Description

Compute common logarithm, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.33 log10l()

Description

Compute common logarithm, long double.

Prototype

```
long double log10l(long double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

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

3.1.34 logb()

Description

Radix-indpendent exponent, double.

Prototype

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

Parameters

Parameter	Description
x	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[FLTRADIX](x).

Additional information

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

3.1.35 logbf()

Description

Radix-indpendent exponent, float.

Prototype

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

Parameters

Parameter	Description
x	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[FLTRADIX](x).

Additional information

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

3.1.36 logbl()

Description

Radix-indpendent exponent, long double.

Prototype

```
long double logbl(long double x);
```

Parameters

Parameter	Description
x	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[FLTRADIX](x).

Additional information

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

3.1.37 ilogb()

Description

Radix-independent exponent, double.

Prototype

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

Parameters

Parameter	Description
x	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[FLTRADIX](x).

3.1.38 ilogbf()

Description

Radix-independent exponent, float.

Prototype

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

Parameters

Parameter	Description
x	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[FLTRADIX](x).

3.1.39 ilogbl()

Description

Radix-independent exponent, long double.

Prototype

```
int ilogbl(long double x);
```

Parameters

Parameter	Description
x	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[FLTRADIX](x).

3.1.40 ldexp()

Description

Scale by power of two, double.

Prototype

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

Parameters

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

Return value

- If x is +/-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()

3.1.41 ldexpf()

Description

Scale by power of two, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

See also

`scalbnf()`

3.1.42 ldexpl()

Description

Scale by power of two, long double.

Prototype

```
long double ldexpl(long double x,
                   int         n);
```

Parameters

Parameter	Description
x	Value to scale.
n	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()

3.1.43 ldexp()

Description

Scale by power of two, double.

Prototype

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

Parameters

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

Return value

- If x is +/-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()`

3.1.44 ldexpf()

Description

Scale by power of two, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

See also

`scalbnf()`

3.1.45 ldexpl()

Description

Scale by power of two, long double.

Prototype

```
long double ldexpl(long double x,
                   int         n);
```

Parameters

Parameter	Description
x	Value to scale.
n	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()`

3.1.46 pow()

Description

Raise to power, double.

Prototype

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

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

3.1.47 powf()

Description

Raise to power, float.

Prototype

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

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

3.1.48 powl()

Description

Raise to power, long double.

Prototype

```
long double powl(long double x,
                 long double y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

3.1.49 scalbn()

Description

Scale, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

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

See also

ldexp()

3.1.50 scalbnf()

Description

Scale, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

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

See also

ldexpf()

3.1.51 scalbnl()

Description

Scale, long double.

Prototype

```
long double scalbnl(long double x,
                    int         n);
```

Parameters

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

Return value

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

Additional information

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

As floating-point arithmetic conforms to IEC 60559, LDBL_RADIX is 2 and scalbnl() is (in this implementation) identical to ldexpl().

See also

ldexpl()

3.1.52 scalbln()

Description

Scale, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

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

See also

ldexp()

3.1.53 scalblnf()

Description

Scale, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

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

3.1.54 scalblnl()

Description

Scale, long double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

As floating-point arithmetic conforms to IEC 60559, LDBL_RADIX is 2 and scalblnl() is (in this implementation) identical to ldexpl().

See also

ldexpl()

3.2 Trigonometric functions

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

3.2.1 sin()

Description

Calculate sine, double.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

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

3.2.2 sinf()

Description

Calculate sine, float.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute sine of, radians.

Return value

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

3.2.3 cos()

Description

Calculate cosine, double.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

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

3.2.4 cosf()

Description

Calculate cosine, float.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute cosine of, radians.

Return value

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

3.2.5 tan()

Description

Compute tangent, double.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

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

3.2.6 tanf()

Description

Compute tangent, float.

Prototype

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

Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

Return value

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

3.2.7 sinh()

Description

Compute hyperbolic sine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

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

3.2.8 sinhf()

Description

Compute hyperbolic sine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

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

3.2.9 cosh()

Description

Compute hyperbolic cosine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

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

3.2.10 coshf()

Description

Compute hyperbolic cosine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

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

3.2.11 tanh()

Description

Compute hyperbolic tangent, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

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

3.2.12 tanhf()

Description

Compute hyperbolic tangent, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

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

3.3 Inverse trigonometric functions

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

3.3.1 asin()

Description

Compute inverse sine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

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

Additional information

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

3.3.2 asinf()

Description

Compute inverse sine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse sine of.

Return value

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

Additional information

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

3.3.3 acos()

Description

Compute inverse cosine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

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

Additional information

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

3.3.4 acosf()

Description

Compute inverse cosine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse cosine of.

Return value

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

Additional information

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

3.3.5 atan()

Description

Compute inverse tangent, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

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

Additional information

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

3.3.6 atanf()

Description

Compute inverse tangent, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse tangent of.

Return value

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

Additional information

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

3.3.7 atan2()

Description

Compute inverse tangent, with quadrant, double.

Prototype

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

Parameters

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

Return value

Inverse tangent of y/x.

Additional information

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

3.3.8 atan2f()

Description

Compute inverse tangent, with quadrant, float.

Prototype

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

Parameters

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

Return value

Inverse tangent of y/x.

Additional information

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

3.3.9 asinh()

Description

Compute inverse hyperbolic sine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

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

3.3.10 asinhf()

Description

Compute inverse hyperbolic sine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

Return value

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

Additional information

Calculates the inverse hyperbolic sine of x.

3.3.11 acosh()

Description

Compute inverse hyperbolic cosine, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

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

3.3.12 acoshf()

Description

Compute inverse hyperbolic cosine, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

Return value

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

3.3.13 atanh()

Description

Compute inverse hyperbolic tangent, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

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

3.3.14 atanhf()

Description

Compute inverse hyperbolic tangent, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

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

3.4 Rounding and remainder functions

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

Return value

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

3.4.2 ceilf()

Description

Compute smallest integer not less than, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute ceiling of.

Return value

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

3.4.3 floor()

Description

Compute largest integer not greater than, double.

Prototype

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

Parameters

Parameter	Description
x	Value to floor.

Return value

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

3.4.4 floorf()

Description

Compute largest integer not greater than, float.

Prototype

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

Parameters

Parameter	Description
x	Value to floor.

Return value

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

3.4.5 trunc()

Description

Truncate to integer, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.4.14 fmodf()

Description

Compute remainder after division, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.4.15 modf()

Description

Separate integer and fractional parts, double.

Prototype

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

Parameters

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

Return value

The signed fractional part of x.

Additional information

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

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

3.4.16 modff()

Description

Separate integer and fractional parts, float.

Prototype

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

Parameters

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

Return value

The signed fractional part of x.

Additional information

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

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

3.4.17 remainder()

Description

Compute remainder after division, double.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.4.18 remainderf()

Description

Compute remainder after division, float.

Prototype

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

Parameters

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

Return value

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

Additional information

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

3.4.19 remquo()

Description

Compute remainder after division, double.

Prototype

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

Parameters

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

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
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

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

3.5.1 fabs()

Description

Compute absolute value, double.

Prototype

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

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

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

3.5.2 fabsf()

Description

Compute absolute value, float.

Prototype

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

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

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

3.6 Fused multiply functions

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

3.6.1 fma()

Description

Compute fused multiply-add, double.

Prototype

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

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

Return value

Return (x * y) + z.

3.6.2 fmaf()

Description

Compute fused multiply-add, float.

Prototype

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

Parameters

Parameter	Description
x	Multiplier.
y	Multiplicand.
z	Summand.

Return value

Return (x * y) + z.

3.7 Maximum, minimum, and positive difference functions

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

3.7.1 fmin()

Description

Compute minimum, double.

Prototype

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

Parameters

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

Return value

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

3.7.2 fminf()

Description

Compute minimum, float.

Prototype

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

Parameters

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

Return value

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

3.7.3 fmax()

Description

Compute maximum, double.

Prototype

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

Parameters

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

Return value

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

3.7.4 fmaxf()

Description

Compute maximum, float.

Prototype

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

Parameters

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

Return value

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

3.7.5 fdim()

Description

Positive difference, double.

Prototype

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

Parameters

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

Return value

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

Chapter 4

Arm AEABI library API

The emFloat provides an implementation of the Arm AEABI functions.

The assembly language floating-point funnctions 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 floatting-point AEABI functions receive their parameters in integer registers and return their results in integer regsisters.

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
x	Augend.
y	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
x	Augend.
y	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
x	Minuend.
y	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
x	Minuend.
y	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
x	Minuend.
y	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
x	Minuend.
y	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
x	Multiplicand.
y	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
x	Multiplicand.
y	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
x	Dividend.
y	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
x	Dividend.
y	Divisor.

Return value

Quotient.

4.2 Floating conversions

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

4.2.1 __aeabi_f2iz()

Description

Convert float to int.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.2 __aeabi_d2iz()

Description

Convert double to int.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.3 __aeabi_f2uiz()

Description

Convert float to unsigned int.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.4 __aeabi_d2uiz()

Description

Convert double to unsigned.

Prototype

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

Parameters

Parameter	Description
x	Double value to convert.

Return value

Integerized value.

4.2.5 __aeabi_f2lz()

Description

Convert float to long long.

Prototype

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

Parameters

Parameter	Description
x	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 double to long long.

Prototype

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

Parameters

Parameter	Description
x	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 float to unsigned long long.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

4.2.8 __aeabi_d2ulz()

Description

Convert double to unsigned long long.

Prototype

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

Parameters

Parameter	Description
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	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
x	Float value to truncate.

Return value

Float value.

4.3 Floating comparisons

Function	Description
__aeabi_fcmpeq	Equal, float.
__aeabi_dcmpeq	Equal, double.
__aeabi_fcmplt	Less than, float.
__aeabi_dcmplt	Less than, double.
__aeabi_fcmple	Less than or equal, float.
__aeabi_dcmple	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
x	Left-hand operand.
y	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
x	Left-hand operand.
y	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
x	Left-hand operand.
y	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
x	Left-hand operand.
y	Right-hand operand.

Return value

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

4.3.5 __aeabi_fcmple()

Description

Less than or equal, float.

Prototype

```
int __aeabi_fcmple(__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
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.

4.3.7 __aeabi_fcmpgt()

Description

Less than, float.

Prototype

```
int __aeabi_fcmpgt(__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 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
x	Left-hand operand.
y	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
x	Left-hand operand.
y	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
x	Left-hand operand.
y	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 funnctions are contained in separate files:

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

5.1 Floating arithmetic

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

5.1.1 __addsf3()

Description

Add, float.

Prototype

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

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

5.1.2 __adddf3()

Description

Add, double.

Prototype

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

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

5.1.3 __subsf3()

Description

Subtract, float.

Prototype

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

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

5.1.4 __subdf3()

Description

Subtract, double.

Prototype

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

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

5.1.5 __mulsf3()

Description

Multiply, float.

Prototype

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

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

5.1.6 __muldf3()

Description

Multiply, double.

Prototype

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

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

5.1.7 __divsf3()

Description

Divide, float.

Prototype

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

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

5.1.8 __divdf3()

Description

Divide, double.

Prototype

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

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

5.2 Floating conversions

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

5.2.1 __fixsfsi()

Description

Convert float to int.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

5.2.2 __fixdfsi()

Description

Convert double to int.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

5.2.3 __fixsfdi()

Description

Convert float to long long.

Prototype

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

Parameters

Parameter	Description
f	Floating value to convert.

Return value

Integerized value.

Notes

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

5.2.4 __fixdfdi()

Description

Convert double to long long.

Prototype

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

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for double to int64 conversion.

5.2.5 __fixunssfsi()

Description

Convert float to unsigned.

Prototype

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

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

5.2.6 __fixunsdfsi()

Description

Convert double to unsigned.

Prototype

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

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

5.2.7 __fixunssfdi()

Description

Convert float to unsigned long long.

Prototype

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

Parameters

Parameter	Description
f	Float value to convert.

Return value

Integerized value.

5.2.8 __fixunsdfdi()

Description

Convert double to unsigned long long.

Prototype

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

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

5.2.9 __floatsisf()

Description

Convert int to float.

Prototype

```
float __floatsisf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

5.2.10 __floatsidf()

Description

Convert int to double.

Prototype

```
double __floatsidf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

5.2.11 __floatdisf()

Description

Convert long long to float.

Prototype

```
float __floatdisf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

5.2.12 __floatdidf()

Description

Convert long long to double.

Prototype

```
double __floatdidf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	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
x	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
x	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
x	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
x	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
x	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
x	Double value to truncate.

Return value

Float value.

5.3 Floating comparisons

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

5.3.1 __eqsf2()

Description

Equal, float.

Prototype

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

Parameters

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

Return value

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

5.3.2 __eqdf2()

Description

Equal, double.

Prototype

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

Parameters

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

Return value

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

5.3.3 __nesf2()

Description

Not equal, float.

Prototype

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

Parameters

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

Return value

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

5.3.4 __nedf2()

Description

Not equal, double.

Prototype

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

Parameters

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

Return value

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

5.3.5 __ltsf2()

Description

Less than, float.

Prototype

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

Parameters

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

Return value

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

5.3.6 __ltdf2()

Description

Less than, double.

Prototype

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

Parameters

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

Return value

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

5.3.7 __lesf2()

Description

Less than or equal, float.

Prototype

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

Parameters

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

Return value

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

5.3.8 __ledf2()

Description

Less than or equal, double.

Prototype

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

Parameters

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

Return value

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

5.3.9 __gtsf2()

Description

Greater than, float.

Prototype

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

Parameters

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

Return value

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

5.3.10 __gtdf2()

Description

Greater than, double.

Prototype

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

Parameters

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

Return value

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

5.3.11 __gesf2()

Description

Greater than or equal, float.

Prototype

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

Parameters

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

Return value

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

5.3.12 __gedf2()

Description

Greater than or equal, double.

Prototype

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

Parameters

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

Return value

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

Chapter 6

Indexes

6.1 Index of types

6.2 Index of functions

