## Old Company Name in Catalogs and Other Documents

On April 1<sup>st</sup>, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1<sup>st</sup>, 2010 Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

#### Notice

- 1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
- Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
- 3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
- 4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
- 5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
- 6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
- 7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
  - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
  - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anticrime systems; safety equipment; and medical equipment not specifically designed for life support.
  - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
- 8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
- 12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majorityowned subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



## SuperH RISC engine C/C++ Compiler Package

## **APPLICATION NOTE:** [Compiler use guide] C Coding Guide (Using FPU)

This document explains usage and gives precautions for FPUs (SH-2E, SH2A-FPU, SH-4, SH-4A), for the SuperH RISC engine C/C++ Compiler V.9.

	Table	of	contents
--	-------	----	----------

Notice: There are some errata on page 15.

1.	Floating-point Processing Unit (FPU)	2
1.1	Specification for Floating-point Numbers	
1.1.1	Internal Representation of Floating-point Numbers	
1.1.2	Single-precision (float type) data format	
1.1.3	Double-precision (double type) data format	
1.2	Register	
1.2.1	Floating-point status / control registers (FPSCR)	0
1.2.2	Floating-point registers.	
1.2.3	Floating-point communication register (FPUL)	
1.2.4	Status register (SR)	
1.3	Rounding	
	-	
2.	Options for Floating-point Calculations and #pragma	13
2.1	Conversion from double -> float (DOuble=Float) for SH-2E	13
2.2	Floating-point Calculation Mode (FPu={Single Double}) for SH2A-FPU, SH-4, and SH-4A	
2.3	Rounding Methods (Round={Zero Nearest}) for SH2A-FPU, SH-4, and SH-4A	
2.4	Handling Non-normalized Numbers (DENormalize={OFF ON}) for SH-4 and SH-4A	17
2.5	Converting Floating-point Division into Multiplication (APproxdiv)	18
2.6	Converting Floating-point Division (FDIv)	19
2.7	Switching the FPSCR Register Precision Mode (FPScr={Aggressive Safe}) for SH2A-FPU, SH-4,	
	and SH-4A	
2.8	Skipping Range Checking during Floating-point Number-integer Conversion (SIMple_float_conv)	
2.9	Suppressing Save/Restore for Floating-point Registers (IFUnc, #pragma ifunc)	23
3.	Efficient Drogromming Techniques	07
	Efficient Programming Techniques	
3.1	Using Floating-point Instructions	21
4.	Frequently Asked Questions	28
4.1	Floating-point Calculation Results	
4.2	Values for Floating-point Numbers in the Watch Window	
Websit	te and Support <website and="" support,ws=""></website>	31

## 1. Floating-point Processing Unit (FPU)

SH-2E, SH2A-FPU, SH-4, and SH-4A come with a built-in FPU for performing high-speed floating-point calculations. SH2A-FPU, SH-4, and SH-4A can perform calculations using double-precision (double type) or single-precision (float type) in the FPU, while SH-2E can use only single-precision calculations

The FPU offers the following characteristics:

• Two rounding modes: Round to Nearest and Round to Zero

(SH-2E only has Round to Zero)

- Two processing modes for non-normalized numbers (only on SH-4 and SH-4A)
- Six exception causes, allowing occurring exceptions to be masked or enabled for each.
  - FPU error, invalid calculation, division by zero, overflow, underflow, and inaccurate (SH-2E only has invalid calculation and division by zero)

## 1.1 Specification for Floating-point Numbers

### 1.1.1 Internal Representation of Floating-point Numbers

• Internal representation format

float types are represented in the IEEE single-precision format (32-bit), and double types and long double types are represented in the IEEE double-precision format (64-bit).

• Internal representation configuration

Figure 1-1 shows the configuration for the internal representation of the float type, double type, and long double type.



Note: When double=float is specified, the double type and float type have the same internal representation.

When both cpu=sh2afpu|sh4|sh4a and fpu=single are specified, the double type and long double type have the same internal representation as the float type.

When both cpu=sh2afpu|sh4|sh4a and fpu=double are specified, the float type has the same internal representation as the double type.

Figure 1-1

The following explains the meaning of each configuration element in an internal representation.

- (i) Sign portion
- Indicates the sign of the floating-point number. 0 indicates positive sign, and 1 indicates negative sign.
- (ii) Exponent portion

Indicates the exponent of the floating-point number as a power of two.

(iii)Mantissa portion

Data about the significant digits in the floating-point number.



#### • Types of values expressed

In addition to regular real numbers, floating-point numbers can also represent infinity and other values. The following lists the types of values that can be represented by a floating-point number.

(i) Normalized number

When the exponent portion is 0 or not all bits are 1. This represents a regular real number.

(ii) Non-normalized number

When the exponent portion is 0, and the mantissa portion is not 0. This represents a real number with a small absolute value.

(iii)Zero

When the exponent portion and mantissa portion are 0. This represents the value 0.0.

(iv) Infinity

When all bits in the exponent portion are 1 and the mantissa portion is 0. This represents infinity.

(v) Non-number

When all bits in the exponent portion are 1 and the mantissa portion is not 0. This is obtained when the calculation results are non-numerical, such as for 0.0/0.0,  $\infty/\infty$ , and  $\infty-\infty$ .

Table 1-1 lists the conditions used to determine the values representing floating-point numbers.

#### Table 1-1 Types of values that represent floating-point numbers

	Exponent portion				
Mantissa portion	0	Not all bits are either 0 or 1	All bits are 1		
0	0	Normalized number	Infinity		
Anything other than 0	Non-normalized number	Normalized number	Non-number		

Note: A non-normalized number represents a floating-point number with a small absolute value in a range that cannot be expressed by a normalized number, but which has fewer significant digits than a normalized number. As such, when the calculation results or any temporary results are a non-normalized number, the significant digits of the results are not guaranteed.

## 1.1.2 Single-precision (float type) data format

The internal representation of the float type consists of a 1-bit sign portion, an 8-bit exponent portion, and a 23-bit mantissa portion.

• Normalized numbers

The sign portion is 0 (positive) or 1 (negative), indicating the sign of the value.

The exponent portion is a value from 1 to 254  $(2^8 - 2)$ . The actual exponent is a value 127 less than this value, with a range from -126 to 127.

The mantissa portion is a value from 0 to  $2^{23} - 1$ . The actual mantissa is interpreted as a value whose  $2^{23}$  bit is 1, with a decimal point immediately following.

Values expressing normalized numbers are as follows:

 $(-1)^{sign-portion} \times 2^{exponent-portion-127} \times (1 + mantissa-portion \times 2^{-23})$ 



Example:

Sign: –

Exponent:  $1000000_{(2)} - 127 = 1$ Mantissa:  $1.11_{(2)} = 1.75$ Value:  $-1.75 \times 2^{1} = -3.5$ Note: (2) indicates a decimal number.

• Non-normalized numbers

The sign portion is 0 (positive) or 1 (negative), indicating the sign of the value.

The exponent portion is 0, with an actual exponent of -126.

The mantissa portion is from 1 to  $2^{23}$ -1, with the actual mantissa interpreted as a value whose  $2^{23}$  bit is 0, with a decimal point immediately following.

Values expressing non-normalized numbers are as follows:

 $(-1)^{sign-portion} \times 2^{-126} \times (mantissa-portion \times 2^{-23})$ 

Example:

Sign: + Exponent: -126Mantissa:  $0.11_{(2)} = 0.75$ Value:  $0.75 \times 2^{-126}$ Note: (2) indicates a decimal number.

• Zero

The sign portion is 0 (positive) or 1 (negative), indicating either +0.0 or -0.0. Both the exponent portion and mantissa portion are 0. Both +0.0 and -0.0 indicate a value of 0.0.

• Infinity

The sign portion is 0 (positive) or 1 (negative), indicating either  $+\infty$  or  $-\infty$ . The exponent portion is 255 (2<sup>8</sup>-1). The mantissa portion is 0.

• Non-number

The exponent portion is  $255 \ (2^8-1)$ . The mantissa portion is a value other than 0.

Note: When the CPU is SH-2E, SH2A-FPU, SH-4, or SH-4A, a non-number for which the highest order bit of the mantissa portion is 0 is called qNaN, a non-number for which the highest order bit of the mantissa portion is 1 is called sNaN.

The values of other mantissa fields, and sign portions are not defined.

## 1.1.3 Double-precision (double type) data format

The internal representations of the double type and long double type consist of a 1-bit sign portion, an 11-bit exponent portion, and a 52-bit mantissa portion.

### • Normalized numbers

The sign portion is 0 (positive) or 1 (negative), indicating the sign of the value.

The exponent portion is a value from 1 to 2046 ( $2^{11}$ –2). The actual exponent is a value 1023 less than this value, with a range from –1022 to 1023.

The mantissa portion is a value from 0 to  $2^{52}$ -1. The actual mantissa is interpreted as a value whose  $2^{52}$  bit is 1, with a decimal point immediately following.

Values expressing normalized numbers are as follows:

 $(-1)^{sign-portion} \times 2^{exponent-portion-1023} \times (1 + mantissa-portion \times 2^{-52})$ 

Example:

Sign: +

Exponent:  $111111111_{(2)} - 1023 = 0$ Mantissa:  $1.111_{(2)} = 1.875$ Value:  $1.875 \times 2^{0} = 1.875$ Note: (2) indicates a decimal number.

• Non-normalized number

The sign portion is 0 (positive) or 1 (negative), indicating the sign of the value.

The exponent portion is 0, with an actual exponent of -1022.

The mantissa portion is from 1 to  $2^{52}$ -1, with the actual mantissa interpreted as a value whose  $2^{52}$  bit is 0, with a decimal point immediately following.

Values expressing non-normalized numbers are as follows:

 $(-1)^{\text{sign-portion}} \times 2^{-1022} \times (\text{mantissa-portion} \times 2^{-23})$ 

Example:

Sign: – Exponent: –1022 Mantissa:  $0.111_{(2)} = 0.875$ Value:  $0.875 \times 2^{-1022}$ Note: (2) indicates a decimal number.

• Zero

The sign portion is 0 (positive) or 1 (negative), indicating either +0.0 or -0.0. Both the exponent portion and mantissa portion are 0. Both +0.0 and -0.0 indicate a value of 0.0.

• Infinity

The sign portion is 0 (positive) or 1 (negative), indicating either  $+\infty$  or  $-\infty$ . The exponent portion is 2047 (2<sup>11</sup>–1). The mantissa portion is 0.



#### • Non-number

The exponent portion is  $2047 (2^{11}-1)$ .

The mantissa portion is any value other than 0.

Note: When the CPU is SH2A-FPU, SH-4, or SH-4A, a non-number for which the highest order bit of the mantissa portion is 0 is called *qNaN*, a non-number for which the highest order bit of the mantissa portion is 1 is called *sNaN*.

The values of other mantissa fields, and sign portions are not defined.

## 1.2 Register

#### 1.2.1 Floating-point status / control registers (FPSCR)

FPSCR is a 32-bit register that controls storage of detailed information about the rounding mode, asymptotic underflow (non-normalized number), and FPU exceptions.

Bit:	31	22 21 20 19 1817	12 2	11 7	6 2	1 0
		QIS FR SZ PR DN	Cause	Enable	Flag	RM
			•			

#### Figure 1-2

#### • QIS: (SH2A-FPU)

qNaN or  $\pm\infty$  is handled as sNaN. This only takes effect when V=1 is set for Enable in FPSCR. QIS=0: Processing is performed as qNaN or  $\pm\infty$ . QIS=1: An exception occurred (same processing as sNaN).

- FR: floating-point register bank (SH-4 and SH-4A) Allocation is switched between floating-point registers FPR0\_BANK0 to FPR15\_BANK0 and FPR0\_BANK1 to FPR15\_BANK1.
- SZ: transfer size mode (SH2A-FPU, SH-4, and SH-4A) SZ=0: The data size for FMOV instructions is 32 bits. SZ=1: The data size for FMOV instructions is 32 bit pairs (64 bits).
- PR: precision mode (SH2A-FPU, SH-4, and SH-4A)
   PR=0: Floating-point instructions are executed as single-precision calculations.
   PR=1: Floating-point instructions are executed as double-precision calculations. The results of instructions for which double-precision is not supported are not defined.

Note:

Do not set both SZ and PR to 1 at the same time for SH-4. This setting is reserved. [SZ, PR] = 11: Reserved (The FPU calculation instructions are not defined).

DN: non-normalized mode (SH-2E, SH2A-FPU, SH-4, and SH-4A)
 DN=1 is always set for SH-2E and SH2A-FPU.
 DN=0: Non-normalized numbers are handled as non-normalized numbers.
 DN=1: Non-normalized numbers are handled as 0.

- $\bullet$  Cause: FPU exception cause field (SH-2E, SH2A-FPU, SH-4, and SH-4A)
- Enable: FPU exception enable field (SH-2E, SH2A-FPU, SH-4, and SH-4A)
- Flag: FPU exception flag field (SH-2E, SH2A-FPU, SH-4, and SH-4A) When an FPU calculation instruction is executed, the FPU exception cause field is first set to 0. When the next FPU exception occurs, the corresponding bit in the FPU exception cause field and FPU exception flag field is set to 1.

The FPU exception flag field keeps the status of the exceptions that occur after the FPU exception flag field is last cleared.

						9	
		FPU error	Invalid	0 division	Overflow	Underflow	Undetermined
		(E)	calculation (V)	(Z)	(O)	(U)	(I)
Cause	FPU exception cause field	Bit 17	Bit 16	Bit 15	Bit 14	Bit 13	Bit 12
Enable	FPU exception enable field	None	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7
Flag	FPU exception flag field	None	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2

#### Table 1-2 Bit allocation for FPU exception processing

 RM: rounding mode (SH-2E, SH2A-FPU, SH-4, and SH-4A) This is always 01 (Round to Zero) for SH-2E.
 RM=00: Round to Nearest RM=01: Round to Zero

## 1.2.2 Floating-point registers

SH-2E and SH2A-FPU have 16 32-bit floating-point registers, while SH-4 and SH-4A have 32. Details are as follows.

• SH-2E

Figure 1-3 shows the floating-point registers. There are 16 floating-point registers (FR*n*), from FR0 to FR15, each of which is 32 bits long. Floating-point registers are used for floating-point instructions.

31		0
	FR0	
	FR1	
	FR2	
	FR3	
	FR4	
	FR5	
	FR6	
	FR7	
	FR8	
	FR9	
	FR10	
	FR11	
	FR12	
	FR13	
	FR14	
	FR15	



### • SH2A-FPU

Figure 1-4 shows the floating-point registers. There are 16 floating-point registers, from FR0 to FR15, each of which is 32 bits long. These 16 registers are referenced from FR0 to FR15, as DR0/2/4/6/8/10/12/14. The correspondence between FPR*n* and reference name is determined by the FPSCR PR bit and SZ bit.

- (1) Floating-point registers FPR*n* (16 registers) FPR0, FPR1, FPR2, FPR3, FPR4, FPR5, FPR6, FPR7, FPR8, FPR9, FPR10, FPR11, FPR12, FPR13, FPR14, and FPR15
- (2) Single-precision floating-point registers FR*i* (16 registers) FR0 to FR15 are allocated to FPR0 to FPR15.
- (3) Double-precision floating-point registers, or single-precision floating-point register pairs DRi (8 registers)

 $DR0 = \{FPR0, FPR1\}, DR2 = \{FPR2, FPR3\},\$ 

 $DR4 = \{FPR4, FPR5\}, DR6 = \{FPR6, FPR7\},$ 

 $DR8 = \{FPR8, FPR9\}, DR10 = \{FPR10, FPR11\},$ 

DR12 = {FPR12, FPR13}, DR14 = {FPR14, FPR15}

For a transfer	FPSCR.SZ=0	FPSCR.SZ=1	
For a calculation	FPSCR.PR=0	FPSCR.PR=1	310
	FR0		FPR0
	FR1	DR0	FPR1
	FR2	502	FPR2
	FR3	DR2	FPR3
	FR4		∫ FPR4
	FR5	DR4	FPR5
	FR6	DBC	∫ FPR6
	FR7	DR6	FPR7
	FR8		∫ FPR8
	FR9	DR8	FPR9
	FR10	<b>DD</b> 10	∫ FPR10
	FR11	DR10	FPR11
	FR12	5510	∫ FPR12
	FR13	DR12	FPR13
	FR14		fPR14
	FR15	DR14	FPR15

Figure 1-4

```
RENESAS
```

#### • SH-4 and SH-4A

Figure 1-5 shows the decimal point registers, for which there are thirty-two 32-bit floating-point registers. Each is comprised of two banks, FPR0\_BANK0 to FPR15\_BANK0, and FPR0\_BANK1 to FPR15\_BANK1. These 32 registers are referenced as FR0 to FR15, DR0/2/4/6/8/10/12/14, FV0/4/8/12, XF0 to XF15, XD0/2/4/6/8/10/12/14, and XMTRX. The correspondence between FPR*n*\_BANK*i* and the reference name is determined by the FR bit for FPSCR.

- (1) Floating-point registers FPRn\_BANKi (32 registers)
  FPR0\_BANK0, FPR1\_BANK0, FPR2\_BANK0, FPR3\_BANK0,
  FPR4\_BANK0, FPR5\_BANK0, FPR6\_BANK0, FPR7\_BANK0,
  FPR8\_BANK0, FPR9\_BANK0, FPR10\_BANK0, FPR11\_BANK0,
  FPR12\_BANK0, FPR13\_BANK0, FPR14\_BANK0, and FPR15\_BANK0
  FPR0\_BANK1, FPR1\_BANK1, FPR2\_BANK1, FPR3\_BANK1,
  FPR4\_BANK1, FPR5\_BANK1, FPR6\_BANK1, FPR7\_BANK1,
  FPR8\_BANK1, FPR9\_BANK1, FPR10\_BANK1, FPR11\_BANK1,
  FPR12\_BANK1, FPR13\_BANK1, FPR14\_BANK1, and FPR15\_BANK1
- (2) Single-precision floating-point registers FRi (16 registers)
  When FPSCR.FR = 0 is set, FR0 to FR15 are allocated to FPR0\_BANK0 to FPR15\_BANK0.
  When FPSCR.FR = 1 is set, FR0 to FR15 are allocated to FPR0\_BANK1 to FPR15\_BANK1.
- (3) Double-precision floating-point registers, or pairs of single-precision floating-point registers DR*i* (8 registers) A DR register consists of two FR registers.

 $DR0 = \{FR0, FR1\}, DR2 = \{FR2, FR3\},\$ 

 $DR4 = \{FR4, FR5\}, DR6 = \{FR6, FR7\},\$ 

DR8 = {FR8, FR9}, DR10 = {FR10, FR11},

DR12 = {FR12, FR13}, and DR14 = {FR14, FR15}

- (4) Single-precision floating-point vector registers FV*i* (4 registers)An FV register consists of four FR registers.
  - $FV0 = \{FR0, FR1, FR2, FR3\},\$

 $FV4 = \{FR4, FR5, FR6, FR7\},\$ 

FV8 = {FR8, FR9, FR10, FR11}, and

 $FV12 = \{FR12, FR13, FR14, FR15\}$ 

- (5) Single-precision floating-point extended registers XFi (16 registers)
  When FPSCR.FR = 0 is set, XF0 to XF15 are allocated to FPR0\_BANK1 to FPR15\_BANK1.
  When FPSCR.FR = 1 is set, XF0 to XF15 are allocated to FPR0\_BANK0 to FPR15\_BANK0.
- (6) Single-precision floating-point extended register pairs XDi (8 registers)

An XD register consists of two XF registers.

 $XD0 = {XF0, XF1}, XD2 = {XF2, XF3},$ 

- $XD4 = \{XF4, XF5\}, XD6 = \{XF6, XF7\},\$
- XD8 = {XF8, XF9}, XD10 = {XF10, XF11},

XD12 = {XF12, XF13}, and XD14 = {XF14, XF15}

(7) Single-precision floating-point extended register matrix XMTRX XMTRX consists of 16 XF registers.

	$\sim$			$\sim$
	XF0	XF4	XF8	XF12
	XF1	XF5	XF9	XF13
XMTRX =	XF2	XF6	XF10	XF14
	XF3	XF7	XF11	XF15
	$\overline{\ }$			



FI	PSCR.FR=	0	31	0	F	PSCR.FR=1	l
FV0	DR0	FR0	FPR0_BANK0		XR0	XR0	XMTRX
		FR1	FPR1_BANK0		XR1		
	DR2	FR3	FPR2_BANK0		XR3	XR2	
		FR3	FPR3_BANK0		XR3		
FV4	DR4	FR4	FPR4_BANK0		XR4	XR4	
		FR5	FPR5_BANK0		XR5		
	DR6	FR6	FPR6_BANK0		XR6	XR6	
		FR7	FPR7_BANK0		XR7		
FV8	DR8	FR8	FPR8_BANK0		XR8	XR8	
		FR9	FPR9_BANK0		XR9		
	DR10	FR10	FPR10_BANK0		XR10	XR10	
		FR11	FPR11_BANK0		XR11		
FV12	DR12	FR12	FPR12_BANK0		XR12	XR12	
		FR13	FPR13_BANK0		XR13		
	DR14	FR14	FPR14_BANK0		XR14	XR14	
		FR15	FPR15_BANK0		XR15		
			31	0			
XMTRX	XR0	XR0	FPR0_BANK1		FR0	DR0	FV0
		XR1	FPR1_BANK1		FR1		
	XR2	XR3	FPR2_BANK1		FR3	DR2	
		XR3	FPR3_BANK1		FR3		
	XR4	XR4	FPR4_BANK1		FR4	DR4	FV4
		XR5	FPR5_BANK1		FR5		
	XR6	XR6	FPR6_BANK1		FR6	DR6	
		XR7	FPR7_BANK1		FR7		
	XR8	XR8	FPR8_BANK1		FR8	DR8	FV8
		XR9	FPR9_BANK1		FR9		
	XR10	XR10	FPR10_BANK1		FR10	DR10	
		XR11	FPR11_BANK1		FR11		
	XR12	XR12	FPR12_BANK1		FR12	DR12	FV12
		XR13	FPR13_BANK1		FR13		
	XR14	XR14	FPR14_BANK1		FR14	DR14	
		XR15	FPR15_BANK1		FR15		

Figure 1-5

## 1.2.3 Floating-point communication register (FPUL)

The FPUL register is used to relay information between the FPU and CPU. The 32-bit FPUL register is the system register, and can be accessed from the CPU through LDS and STS instructions. For example, the processing flow for converting an integer stored in general register R1 to a single-precision floating-point is as follows:

R1 -> (LDS instruction) -> FPUL -> (single-precision FLOAT instruction) -> FR1

#### 1.2.4 Status register (SR)

When the FD bit of a status register (SR) is 1, FPU instructions throw general FPU suppression exceptions, and if the FPU instruction is in a delay slot, a slot FPU suppression exception occurs (FPU instruction:  $H'F^{***}$  instruction, LDS (.L) / STS (.L) instruction for FPUL/FPSCR).

Status re	gister (SR)			
Bit:	31 30 29 28 27		16 15 14	10 9 8 7 4 3 2 1 0
	- MDRB BL	-	FD	FDFD IMASK - ST



## 1.3 Rounding

The following rounding is performed during arithmetic calculation for floating-point numbers or constant substitution when the mantissa of the internal representation of a proper value exceeds the significant digits:

- When the CPU is SH-2E, Round to Zero is performed (the portion beyond the significant digits is truncated).
- When the CPU is SH2A-FPU, SH-4, or SH-4A, either Round to Zero or Round to Nearest can be selected for the FPSCR RM.
- For CPUs with no FPU, floating-point calculations are processed by a run-time routine, and Round to Nearest is performed.

Round to Nearest

With Round to Nearest, of two approximating floating-point numbers, the value is rounded to that with the closer internal representation.

The direction in which the value is approximated is determined by the value after the final digit of the mantissa. Note that when the value before approximation is exactly between the two approximated floating-point numbers, rounding is performed to the value with 0 as the final digit of the mantissa.



#### Figure 1-7

For example, since 0.1 cannot be expressed properly within the significant digits for the single-precision floating-point number format, it is rounded.

It is represented as 0x3DCCCCCC for Round to Zero, or 0x3DCCCCCD for Round to Nearest.



This value will become 0.0999..., the approximated value closest to 0.1.



## 2. Options for Floating-point Calculations and #pragma

## 2.1 Conversion from double -> float (DOuble=Float) for SH-2E

This option handles floating-point numbers of the float type and double type (anything other than that declared as a long double type) as used within the program, as a single-precision number. Since the FPU for SH-2E only supports single-precision, double-precision calculation are processed by calling a run-time routine. This option can be specified to enable the FPU to handle floating-point calculations for types other than the long double type, thereby increasing floating-point calculation speed.

#### Note:

This option can also be specified on CPUs with no built-in FPU (SH-1, SH-2, SH-2A, SH2-DSP, SH-3, SH3-DSP, and SH4AL-DSP).

#### Example:

ampie.		*	
Source code		:	
	double a, float b)		
{			
return a ·	+ b;	:	
}			
	nbly code when double=float is		mbly code when double=float is
unspecified (o	default)	specified	
_func:		_func:	
STS.L	PR,@-R15	FADD	FR5,FR4
MOV	R15,R2	; RTS	
ADD	#8,R2	FMOV.S	FR4,FR0
MOV.L	@ (4,R2),R1 ; (part of) a		
MOV.L	@R2,R4 ; (part of) a	:	
MOV.L	R1,@-R15	:	
MOV.L	R4,@-R15		
ADD	#-8,R15	:	
MOV	R15,R4	:	
MOV.L	R4,@-R15		
MOV.L	L11,R5 ;ftod_a		
JSR	@R5		
FMOV.S	FR4,FR0		
ADD	#4,R15	:	
MOV.L	@ (20,R15),R6		
MOV.L	L11+4,R7 ;addd_a		
JSR	@R7	:	
MOV.L	R6,@-R15	:	
ADD	#20,R15		
LDS.L	@R15+,PR	:	
RTS			
NOP			
L11:			
.DATA.L	ftod_a	:	
.DATA.L	addd_a		
	—		

Setting this option in High-Performance Embedded-Workshop (herein as HEW)

SuperH RISC engine Standard Toolchain	? ×
Configuration :	C/C++ Assembly Link/Library Standard Library CPU Deb
Debug	QPU:       SH-2E         Division:       CPU         Endian:       Big         EPU:       Mix         Round to:       Zero         Openormalized number allower as a result         Position independent code (PIC)         Treat double as float         Bit field's members are allocated from the lower bit         Pack struct, union and class         Use try, throw and catch of C++         Enable/disable runtime information
	OK Cancel

Figure 2-1

## 2.2 Floating-point Calculation Mode (FPu={Single|Double}) for SH2A-FPU, SH-4, and SH-4A

This option handles the floating-point number types used within the program by unifying them. Even though the SH2A-FPU, SH-4, and SH-4A FPUs support both single-precision and double-precision calculation, when calculations with different precisions are performed, the FPSCR PR bit needs to be switched. As such, performance may degrade when floating-point calculations of differing precisions are performed. This can be mitigated by using the FPU option to unify the types of floating-point numbers within a program.

• Mix [default]

Calculation is performed as specified in the C/C++ source. The compiler generates code to switch the PR bit in the FPSCR register.

• Single (fpu=single)

All floating-point calculations are performed using single-precision floating-point numbers (float type). The compiler does not use the PR bit in the FPSCR register.

• Double (fpu=double)

All floating-point calculations are performed using double-precision floating-point numbers (double type). The compiler does not use the PR bit in the FPSCR register.

Note:

Since the compiler does not use the PR bit in the FPSCR register when Single or Double is selected, the initial value needs to be set by the user program.

Note that since the value of the PR bit for the initial status of the CPU is 0 (single-precision), the calculation is invalid when Double is selected and no initial value setting was performed. FPSCR can be set by using the set\_fpscr embedded function.

Example:

Source code					•		
double func (d	double func (double a, float b)		:				
{			-				
return a -	- b;						
}			:				
Expanded asser	bly code whe	en fpu is not	Expanded asse	Expanded assembly code when		Expanded assembly code when	
specified (def	ault)		fpu=single is	s specified	fpu=double is	specified	
			func:		func:		
_func:			FADD	FR5,FR4	FMOV.S	FR4,FR0	
FLDS	FR6,FPUL		RTS		FMOV.S	FR5,FR1	
STS	FPSCR,R2		FMOV.S	FR4,FR0	: RTS		
MOV	#8,R6	; H'0000008			FADD	DR6,DR0	
SHLL16	R6		:		:		
OR	R6,R2		:		-		
LDS	R2,FPSCR						
FCNVSD	FPUL,DR0		:				
RTS			:		:		
FADD	DR4,DR0						

Setting this option in HEW

SuperH RISC engine Standard Toolchain	?×
Configuration :	C/C++ Assembly Link/Library Standard Library CPU Deb
Debug	QPU: SH-4         Division: CPU         Endian: Big         EPU: Mix         Nix         Round to: Double Mix         Openormalized number allower as a result         Position independent code (PIC)         Treat double as float         Bit field's members are allocated from the lower bit         Pack struct, union and class         Use try, throw and catch of C++         Enable/disable runtime information
	OK Cancel

## Figure 2-2

## 2.3 Rounding Methods (Round={Zero|Nearest}) for SH2A-FPU, SH-4, and SH-4A

The method used for rounding can be selected for SH2A-FPU, SH-4, and SH-4A. When the rounding method is set, the setting for the RM bit for FPSCR and the setting for the compiler round option must be the same. The compiler round option is performed in **Round to**, in Figure 2-3.

- Zero (round=zero) [default] Rounding is performed using Round to Zero.
- Nearest (round=nearest) Rounding is performed using Round to Nearest.

In the initial CPU status, the value of the RM bit is 01 (Round to Zero). When Nearest is specified, set the RM bit to 00 (Round to Nearest).

Since the compiler does not generate code to change the value of the RM bit in FPSCR, the RM bit must be set explicitly by the user program. In the initial CPU status, the value of the RM bit is 00 (Round to Zero). When Nearest is specified, set the RM bit to 01 (Round to Nearest). FPSCR can be set by using the set\_fpscr embedded function.



Example:

Source code	
float $ff = 0.1f;$	
Expanded assembly code when round=zero is	Expanded assembly code when round=nearest is
specified (default)	specified
_ff:	_ff:
.DATA.L H'3DCCCCCC	.DATA.L H'3DCCCCCD

#### Setting this option in HEW

Configuration : Debug C/C++ Assembly Link/Library Standard Library CPU Deb QPU: SH-4 QPU: SH-4 QPU: SH-4 QPU: SH-4 Division: OPU Endian : Big PU: Single PU: Single PU: Single Position independent code (PIC) Treat double as float Bit field's members are allocated from the lower bit Pack struct, union and class Use try, throw and catch of C++ Enable/disable runtime information	perH RISC engine Standard Toolchain	?
	Configuration : Debug All Loaded Projects All Loaded Projects C source file C source file Assembly source file Assembly source file	C/C+++       Assembly       Link/Library       Standard Library       CPU       Deb ▲ ▲         QPU:       SH-4       ▼         Division:       CPU       ▼         Endian:       Big       ▼         Endian:       Big       ▼         EPU:       Single       ▼         Round to:       Zero       ▼         Denormalized       Nearest       ■         Position independent code (PIC)       Treat double as float       ■         Bit field's members are allocated from the lower bit       ■       Pack struct, union and class         Use try, throw and catch of C++       ■       ■       ■

Figure 2-3



## 2.4 Handling Non-normalized Numbers (DENormalize={OFF|ON}) for SH-4 and SH-4A

SH-4 and SH-4A can handle non-normalized numbers either as non-normalized numbers, or as 0. When handling for non-normalized number is set, the setting for the FPSCR DN bit and the setting for the compiler denormalize option need to be the same. The compiler denormalize option can be set by selecting the **Denormalized number allower** as a result check box in Figure 2-4.

- When the check box is not selected (denormalize=off) [default] Non-normalized numbers are handled as 0.
- When the check box is selected (denormalize=on) Non-normalized constants are handled as non-normalized numbers.

Since the compiler does not generate code to change the value of the FPSCR DN bit, the DN bit must be set explicitly by the user program. The value of the FPSCR DN in the initial CPU status is 1 (handle non-normalized numbers as 0). To handle non-normalized numbers as non-normalized numbers, explicitly set the DN bit to 0 (handle non-normalized numbers). FPSCR can be set by using the set\_fpscr embedded function.

#### Example:

Source code float ff = 1.0e-38f;	
Expanded assembly code when denormalize=off is specified (default) ff:	Expanded assembly code when denormalize=on is specified ff:
DATA.L H'0000000	.DATA.L H'006CE3EE

Setting this option in HEW

SuperH RISC engine Standard Toolchain	?×
Configuration :	C/C++ Assembly Link/Library Standard Library CPU Deb
Debug	QPU: SH-4         Division: OPU         Endian: Big         EPU: Single         Round to: Nearest         ✓         Denormalized number allower as a result         Position independent code (PIC)         Treat double as float         Bit field's members are allocated from the lower bit         Pack struct, union and class         Use try, throw and catch of C++         Enable/disable runtime information
	OK Cancel

Figure 2-4





## 2.5 Converting Floating-point Division into Multiplication (APproxdiv)

This option replaces division by a floating-point constant to multiplication by the inverse of the constant, allowing improved calculation speed. The **Approximate a float-point constant division** check box in Figure 2-5 can be selected to specify optionapproxdiv.

This option can be specified regardless of the FPU used.

#### Note:

Keep in mind that this optimization may change the margin of error for floating-point calculations.

Example: For cpu=sh4 Source code float x; void f (float y) { x=y/3.0f; Expanded assembly code when approxdiv is not Expanded assembly code when approxdiv is specified specified (default) \_f: f: MOVA L11,R0 MOVA L11,R0 L11+4,R2 L11+4,R2 MOV.L MOV.L ; \_x ; \_x FMOV.S @R0,FR8 FMOV.S @R0,FR8 ;FR8=H'4040000 ;FR8=H'3EAAAAAA FDIV FR8, FR4 FMUL FR8, FR4 (3.0) (0.3333333...); ; RTS RTS FR4,@R2 FR4,@R2 FMOV.S FMOV.S ; x ; x L11: L11: н'4040000 .DATA.L н'ЗЕАААААА .DATA.L .DATA.L .DATA.L \_x \_x

In this example, division by 3.0 is converted into multiplication by its inverse.

Setting	this	ontion	in	HFW
Setting	uns	opuon	ш	

SuperH RISC engine Standard Toolchain	? ×
Configuration : Debug All Loaded Projects test_fpu C source file C++ source file C-Assembly source file Linkage symbol file	C/C++ Assembly Link/Library Standard Library CPU Deb
	OK Cancel

#### Figure 2-5



## 2.6 Converting Floating-point Division (FDIv)

This option converts integer division to floating-point division. By converting integer division processed by a run-time routine into an FPU division instruction, calculation speed can be improved. The **Change integer division into floating-point** check box in Figure 2-6 can be selected to specify the optionfdiv option.

Note:

When cpu=sh2afpu|sh4|sh4a and fpu=double are specified, conversion is performed when both the divisor and dividend are within 4 bytes. Otherwise, conversion is performed when both the divisor and dividend are within 2 bytes.

Example: For cpu=sh4 and fpu=double

Source code					
int x;			:		
func (int a, :	int b)		:		
{					
x = a/b;					
}			:		
	-	en fdiv is not	Expanded assem	bly code wher	n fdiv is specified
specified (det	tault)		:		
_func:			:_func:		
STS.L	PR,@-R15		: LDS	R4,FPUL	
MOV.L	L11+2,R2	;divls	MOV.L	L11,R6	<i>i</i> _x
MOV	R4,R1		: FLOAT	FPUL,DR6	
MOV.L	L11+6,R6	; _x	: LDS	R5,FPUL	
JSR	@R2		FLOAT	FPUL,DR8	
MOV	R5,R0		: FDIV	DR8,DR6	
LDS.L	@R15+,PR		: FTRC	DR6,FPUL	
RTS			STS	FPUL,R2	
MOV.L	R0,@R6	; x	: RTS		
L11:			MOV.L	R2,@R6	; x
.RES.W	1		L11:		
.DATA.L	divls		.DATA.L	_x	
.DATA.L	_x		:		

Setting this option in HEW

SuperH RISC engine Standard Toolchain	? ×
Configuration : Debug All Loaded Projects test_fpu C source file C++ source file C++ source file Linkage symbol file	C/C++       Assembly Link/Library Standard Library CPU Deb         Category:       Other         Miscellaneous options:          Obey ansi specifications more strictly       ▲         Phange integer division into floating-point          Not check the range in conversion between floating point nur       ▲         User defined options:
	Options C/C++ : -cpu=sh4 -object="\$(CONFIGDIR)¥\$(FILELEAF).obj" -debug - show=tab=4 -gbr=auto -fdiv -chgincpath -errorpath - global_volatile=0 -opt_range=all -infinite_loop=0 -
	OK Cancel



## 2.7 Switching the FPSCR Register Precision Mode (FPScr={Aggressive|Safe}) for SH2A-FPU, SH-4, and SH-4A

This option specifies whether to guarantee the FPSCR precision mode (PR bit) before and after function calls. When fpscr=aggressivee (default) is specified, the value of the PR bit before and after function calls is not guaranteed.

When fpscr=safe is specified, the value of the PR bit of a called function is guaranteed so that single-precision is always used after the function call.

When fpscr=aggressivee (default) is specified, the value of the PR bit when a function call returns is unknown. As such, when a floating-point calculation occurs after a function call, code is always generated to re-set the FPSCR value. However, when fpscr=safe is specified, FPSCR is only set when needed, since the PR bit when a function call is returned is guaranteed to be single-precision. As such, fpscr=safe generates more efficient code.

When specifying fpscr=safe, select the Change FPSCR register if double data used check box in Figure 2-7.

Note:

Since this option changes the function interface, it needs to be changed for all files at the same time. Extra precaution is required when libraries created using a compiler of a previous version are linked.



Example: For cpu=sh4

Source code			-			
extern void sub	. ,		-			
extern float fl,	f2;					
func ()						
{			:			
sub () ;						
f1 =1.0f;			-			
}						
	y code when fp	scr=aggressive is	: Expan	ded assembl	ly code when	fpscr=safe is
specified(defaul	t)		: speci	fied		
_func:						
STS.L	PR,@-R15		func	:		
MOV.L	L11,R1	; _sub	-	STS.L	PR,@-R15	
JSR	@R1			MOV.L	L11,R1	; _sub
NOP			-	JSR	@R1	
STS	FPSCR,R4			NOP		
MOV.L	L11+4,R6	; H'FFE7FFFF		MOVA	L11+4,R0	
MOVA	L11+8,R0			MOV.L	L11+8,R4	; _f1
MOV.L	L11+12,R5	; _f1		FMOV.S	@R0,FR8	
AND	R6,R4			LDS.L	@R15+,PR	
LDS	R4, FPSCR		:	RTS		
FMOV.S	@R0,FR8			FMOV.S	FR8,@R4	; f1
LDS.L	@R15+,PR		L11:			
RTS			-	.DATA.L	_sub	
FMOV.S	FR8,@R5	; fl	-	.DATA.L	H'3F800000	
L11:			-	.DATA.L	_f1	
.DATA.L	_sub		:			
.DATA.L	H'FFE7FFFF		-			
.DATA.L	H'3F800000					
.DATA.L	_f1		:			
			-			

## Setting this option in HEW

SuperH RISC engine Standard Toolchain	? ×
Configuration :	C/C++ Assembly Link/Library Standard Library CPU Deb
Debug	Category : Other Miscellaneous options : Avoid illegal SH7055 instructions Change FPSCR register if double data used Treats loop condition as volatile qualified User defined options :
	Options C/C++ : -cpu=sh4 -object="\$(CONFIGDIR)¥\$(FILELEAF).obj" -debug - show=tab=4 -gbr=auto -fpscr=safe -chgincpath -errorpath - global_volatile=0 -opt_range=all -infinite_loop=0 -

Figure 2-7



## 2.8 Skipping Range Checking during Floating-point Number-integer Conversion (SIMple\_float\_conv)

This option generates code for skipping range checking for converted values, when type conversion is performed between an unsigned integer type and floating-point number type. By skipping value range checking, calculation speed can be improved. However, when the value before type conversion is neither an integer from 0 to 2147483647 nor a floating-point number from 0.0 to 2147483647.0, the calculation results are invalid. Do not use this option when values outside these ranges may be input.

Example: For cpu=sh4

• Conversion from the float type to the unsigned int type

Source cod	2		
unsigned 1	ong func (float f)	:	
{		:	
return	( (unsigned int) f) ;	:	
}		-	
Expanded a	ssembly code when simple_float_conv	: Expanded as	ssembly code when simple_float_conv
	cified (default)	is specifie	
func:	cified (default)	func:	
_		. —	
MOV	#79,R2 ; 0x000004F	: FTRC	FR4,FPUL ;float -> int
SHLL8	R2 ;	:	; conversion
SHLL16	R2 ; 0x4F000000	: RTS	
LDS	R2, FPUL	: STS	FPUL, RO
FSTS	FPUL, FR8		
FCMP/G	Г FR4,FR8	-	
BT	L12	:	
FADD	FR8,FR8 ; when f>=0x4F000000,	:	
FSUB	FR8,FR4 ; the value before		
FOOD	; conversion	-	
		-	
	; is (f-0x4F800000)	:	
L12:		:	
FTRC	FR4,FPUL ;float -> int conversion	:	
RTS		:	
STS	FPUL, RO	:	

• Conversion from the unsigned int type to the float type

Sou	rce code				
flo	at func (un	signed int ui)			
{					
	return (	(float) ui) ;			
}					
Exp	anded assem	bly code when simple_float_conv_	Expanded asse	mbly code wh	ien
is	not specifi	ed (default)	simple_float_	conv is spec	cified
_fu	nc:		func:		
	LDS	R4,FPUL	LDS	R4,FPUL	
	CMP/PZ	R4	RTS		
	BT/S	L12	FLOAT	FPUL,FR0	;int ->float
	FLOAT	FPUL,FR0 ;int->float			;conversion
		;conversion			
	MOVA	L13+2,R			
	FMOV.S	<pre>@R0,FR9 ; When u &gt;= 0x8000000u,</pre>	•		
	FADD	FR9,FR0 ; 0x4F800000 is added to	-		
		; the converted value.			
L12	:		•		
	RTS		-		
	NOP				
L13	:				
	RES.W	1			
1	DATA.L	H'4F800000			



Setting this option in HEW

SuperH RISC engine Standard Toolchain	?×
Configuration : Debug All Loaded Projects C Source file C Source file C C ++ source file Linkage symbol file	C/C++       Assembly       Link/Library       Standard Library       CPU       Deb ▲ ▲         Category:       Other
	show=tab=4 -gbr=auto -simple_float_conv -chgincpath - errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -
	UN Cancel

Figure 2-8

## 2.9 Suppressing Save/Restore for Floating-point Registers (IFUnc, #pragma ifunc)

This option and #pragma prevent floating-point registers from being saved or restored. This function is used with the interrupt function specification (#pragma interrupt).

If a function call exists within the interrupt function, since which register to use cannot be determined from within the called function, all registers including the floating-point register are saved and restored. However, when floating-point calculation are not used within the called function, save and restore can be skipped for the floating-point register. By specifying optionifunc or #pragma ifunc, save and restore can be suppressed for a floating-point register.

When optionifunc is used, the specification takes effect for all functions in the source file, whereas a #pragma ifunc specification takes effect for a particular function. ifunc and #pragma ifunc need to be specified in both the interrupt function and the function called from the interrupt function. When the function called from an interrupt function requires a floating-point instruction, the following compile error is reported:

C2843 (E) Illegal floating type used in function

Likewise, when a function for which if unc or #pragma if unc is not specified is called from a interrupt function, the following warning is reported:

C1029 (W) Function with ifunc calls function-name without ifunc

Note that even when this warning is reported, suppression of floating-point register save/restore remains suppressed.



Example: For cpu=sh4

Source code (#pragma	ifund not specified)	· Source ac	de (#pragma	ifunc specified)		
	func)					
"Pragma Incerrupt ()		: #pragma interrupt (func) : #pragma ifunc (sub,func)				
void sub ()		void sub				
			()			
{		- {				
}		: }				
void func ()		: void func	: ()			
{		{				
sub () ;		: sub ()	;			
}		: }				
Expanded assembly co	le	Expanded	assembly cod	le		
_sub:		_sub:				
RTS		:	RTS			
NOP		-	NOP			
_func:		_func:				
MOV.L	R0,@-R15	:	MOV.L	R0,@-R15		
MOV.L	R1,@-R15	:	MOV.L	R1,@-R15		
MOV.L	R2,@-R15	:	MOV.L	R2,@-R15		
MOV.L	R3,@-R15		MOV.L	R3,@-R15		
MOV.L	R4,@-R15	:	MOV.L	R4,@-R15		
MOV.L	R5,@-R15	:	MOV.L	R5,@-R15		
MOV.L	R6,@-R15	-	MOV.L	R6,@-R15		
MOV.L	R7,@-R15	-	MOV.L	R7,@-R15		
STS.L	PR,@-R15	:	STS.L	PR,@-R15		
FMOV.S	FR0,@-R15	:	STC	SSR,@-R15		
FMOV.S	FR1,@-R15	-	STC	SPC,@-R15		
	FR1,@-R15 FR2,@-R15	-		_sub		
FMOV.S			BSR	_sub		
FMOV.S	FR3,@-R15	:	NOP	ep15, apa		
FMOV.S	FR4,@-R15	:	LDC	@R15+,SPC		
FMOV.S	FR5,@-R15	-	LDC	@R15+,SSR		
FMOV.S	FR6,@-R15	-	LDS.L	@R15+,PR		
FMOV.S	FR7,@-R15	:	MOV.L	@R15+,R7		
FMOV.S	FR8,@-R15		MOV.L	@R15+,R6		
FMOV.S	FR9,@-R15	-	MOV.L	@R15+,R5		
FMOV.S	FR10,@-R15	-	MOV.L	@R15+,R4		
FMOV.S	FR11,@-R15	:	MOV.L	@R15+,R3		
STS.L	FPUL,@-R15	:	MOV.L	@R15+,R2		
STS.L	FPSCR,@-R15		MOV.L	@R15+,R1		
STC	SSR,@-R15	1	MOV.L	@R15+,R0		
STC	SPC,@-R15	1	RTE			
BSR	_sub	1	NOP			
NOP						
LDC	@R15+,SPC	-				
LDC	@R15+,SSR	:				
LDS.L	@R15+,FPSCR	-				
LDS.L	@R15+,FPUL	-				
FMOV.S	@R15+,FR11					
FMOV.S	@R15+,FR10	:				
FMOV.S	@R15+,FR9					
FMOV.S	@R15+,FR8					
FMOV.S	@R15+,FR7					
		:				
FMOV.S	@R15+,FR6	-				
FMOV.S	@R15+,FR5	-				
FMOV.S	@R15+,FR4	:				
FMOV.S	@R15+,FR3					
FMOV.S	@R15+,FR2	-				
FMOV.S	@R15+,FR1	•				
FMOV.S	@R15+,FR0	:				
LDS.L	@R15+,PR	:				
MOV.L	@R15+,R7	-				
MOV.L	@R15+,R6	:				
MOV.L	@R15+,R5	:				
MOV.L	@R15+,R4	:				
MOV.L	@R15+,R3	-				
MOV.L	@R15+,R2	-				
MOV.L	@R15+,R1	:				
MOV.L	@R15+,R0	:				
RTE		-				



(1) Specifying #pragma ifunc

Format: #pragma ifunc [(]function-name[)]

Description: Suppresses floating-point register save/restore for the function specified by function-name.

Notes:

Specify #pragma ifunc before the function declaration.

A compile error will occur when a floating-point number is used within a function for which #pragma ifunc is specified.

(2) Specifying optionifunc

This is specified on a file basis. A compile error will occur when this is specified for a source program for which floating-point instructions are generated.

SuperH RISC engine Standard Toolchain	<u>?</u> ×
Configuration : Debug Configuration : Debug Configuration : Debug Configuration : Debug Configuration : Configuration : Configuratio : Configuration : Con	C/C++ Assembly Link/Library Standard Library CPU Deb • Category: Object Qutput file type : Machine code (*.obj) Generate debug information Outgut directory : \$(CONFIGDIR)¥ Modify
	-cpu=sh4 -round=nearest -denormalize=on - object="\$(CONFIGDIR)¥\$(FILELEAF).obj" -debug -gbr=auto - chgincpath -errorpath -global_volatile=0 -opt_range=all -
	OK Cancel

Figure 2-9



Choose the Code generation tab, and then select the Use no FPU instructions.

Object details		? ×
Code generation Code generation2		
Section : Program section (P) P Division sub-options : Default Use no EPU instructions Align labels after unconditional branches 16/32byte boundaries : None	Template : Auto S <u>tore string data in</u> Const section	
	ОК	Cancel

Figure 2-10



## 3. Efficient Programming Techniques

## 3.1 Using Floating-point Instructions

Overview

To use floating-point instructions FABS (SH-2E, SH2A-FPU, SH-4, and SH-4A) or FSQRT (SH2A-FPU, SH-4, and SH-4A) for single-precision, include the mathf.h include file, and then call the fabsf or sqrtf single-precision floating-point function. For double-precision, include the math.h include file, and then call the fabs or sqrt double-precision floating-point function.

#### Description

Perform the following to use the single-precision floating-point instruction FABS (SH-2E, SH2A-FPU, SH-4, and SH-4A) or FSQRT (SH2A-FPU, SH-4, and SH-4A):

(a) Include mathf.h.

(b) Call the fabsf function (FABS) or sqrtf function (FSQRT).

Perform the following to use the double-precision floating-point instruction FABS (SH2A-FPU, SH-4, and SH-4A) or FSQRT (SH2A-FPU, SH-4, and SH-4A):

(a) Include math.h.

(b) Call the fabs function (FABS) or sqrt function (FSQRT).

#### Usage examples

In the BEFORE example, since mathf.h is not included, the compiler does not recognize it as a standard function, and calls the fabsf function from the library. If mathf.h is included, the compiler can recognize that the function is for the FABS instruction, and generate a FABS instruction directly.

#### Note:

The mathf.h header is not an ANSI-standard C library function.

Source code - BEFORE				: Source code - AFTER				
float fak	osf (float) ;		-	<pre>#include <mathf.h></mathf.h></pre>				
float f (	float x, flo	paty) {	-	float f (float x,	float y) {			
retu	urn fabsf (x)	+fabsf (y) ;	:	return fabsf	(x) +fabsf (y) ;			
}			-	}				
Assembly	code - BEFOR	<u>RE</u>	:	Assembly code - AFTH	ER			
_f:			-	_f:				
	STS.L	PR,@-R15		FABS	FR4			
	FMOV.S	FR14,@-R15	:	FABS	FR5			
	FMOV.S	FR15,@-R15		FADD	FR5,FR4			
	MOV.L	L11+2,R1 ; _fabsf		RTS				
	JSR	@R1		FMOV.S	FR4,FR0			
	FMOV.S	FR5,FR15	-					
	FMOV.S	FR0,FR14	:					
	MOV.L	L11+2,R4 ; _fabsf						
	JSR	@R4						
	FMOV.S	FR15,FR4	:					
	FADD	FR0,FR14	-					
	FMOV.S	FR14,FR0						
	FMOV.S	@R15+,FR15	:					
	FMOV.S	@R15+,FR14	-					
	LDS.L	@R15+,PR						
	RTS		:					
	NOP		-					
L11:			-					
	.RES.W	1	:					
	.DATA.L	_fabsf						



## 4. Frequently Asked Questions

## 4.1 Floating-point Calculation Results

#### Q:

Why does a floating-point calculation not return the expected value?

### A1:

The expected value may not be returned due to a rounding error.

```
(Sample program)
double a = 0.1 * 100;
int b;
void func(void)
{
    b = (int) a;
}
```

#### List 4-1

This phenomenon occurs due to rounding errors for the floating-point number representation. As such, there is no fundamental solution. Write code as follows, as a workaround.

(Code without error in mind)	(Code with error in mind)
float f;	<pre>const float s = 1.0e-10f;</pre>
	float f;
if ( f == 0.1f ) {	if ((0.1f-s) <= f && f <= (0.1f+s)) {
•	· · · · · · · · · · · · · · · · · · ·
}	•
	}

List 4-2



A2:

For SH2A-FPU, SH-4, and SH-4A, since the option and value for FPCSR may not correspond, check the following:

- 1. The correspondence between the compiler fpu option and FPCSR PR bit value.
- 2. The correspondence between the compiler denormalize option and FPCSR DN bit value (SH-4 and SH-4A only).

3. The correspondence between the compiler round option and FPCSR RM bit value (SH-4 and SH-4A only). Floating-point calculation mode

There are two possible fpu options: fpu=single and fpu=double.

When fpu=single is selected, all floating-point data is handled as single-precision floating-point data. When fpu=double is selected, all floating-point data is handled as double-precision floating-point data. When no fpu option is used, floating-point data is used according to the declaration type of the source code.

When no fpu option is used, the compiler generates code that changes the PR bit according to the corresponding precision during floating-point calculations, but when fpu=single or fpu=double is selected, the compiler does not generate any code that accesses the PR bit.

Meanwhile, the PR bit of the FPSCR register is initialized to 0 during reset.

As such, when the fpu option is not used or fpu=single is selected, operation is performed correctly without concern for the PR bit, but when fpu=double is selected, the PR bit is changed to 1 before the FPU calculation.

#### Handling non-normalized numbers

When selecting the denormalize=on option, make sure that the FPSCR DN bit is set to 0.

When selecting the denormalize=off option, make sure that the FPSCR DN bit is set to 1.

### Rounding methods

When selecting the round=zero option, make sure that the FPSCR RM bit is set to 01. When selecting the round=nearest option, make sure that the FPSCR RM bit is set to 00.

## 4.2 Values for Floating-point Numbers in the Watch Window

#### Q:

Why do the values of floating-point numbers differ between that displayed in the watch window and the actual value?

A:

× •	R Ŗ 🗖 🖊 🗙	i 🥩 📌 🖓		
	Name	Value		Туре
	R a	10	{ 700005c0 }	(double
Ш				
Ш				
	•			Þ
Ш	▲ ▶ \ Watch1 \ Watch	:h2 À Watch3 À Watch4 /	·	

Figure 4-1



The internal representation of the actual value can be checked in the memory window (Figure 4-2) or register window (Figure 4-3).

×		<b>16</b>	<u>10</u> ± <u>10</u>	8	2	dbc	<b>3</b> 8 2	ある	<b>50</b> d		£.	.d	.16	.32
	Address	Label	Regis	ter	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
11	700005C0	_a			40	23	FF	FF	FF	FF	FF	FF		
LU														
LU														
LL														



Ň	Name	Value		<u> </u>
Ē	DR4	000000000000000000000000000000000000000	+0.0000000000000e+000	
	DR6	000000000000000000000000000000000000000	+0.00000000000000e+000	
	DR8	4023FFFFFFFFFFFFF	+1.00000000000000e+001	
	DR10	000000000000000000000000000000000000000	+0.00000000000000e+000	
	np12		+0 0000000p00000-+000	

Figure 4-3



## Website and Support <website and support,ws>

Renesas Technology Website <u>http://japan.renesas.com/</u>

Inquiries

http://japan.renesas.com/inquiry csc@renesas.com

## Revision Record <revision history,rh>

		Description	
Rev.	Date	Page	Summary
1.00	Jun.1.07		First edition issued

## Notes regarding these materials

- 1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
- Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
- 3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
- 4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
- 5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
- 6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
- 7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
  - Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below: (1) artificial life support devices or systems
    - (2) surgical implantations
    - (3) healthcare intervention (e.g., excision, administration of medication, etc.)
    - (4) any other purposes that pose a direct threat to human life

Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.

- 9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
- 10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
- 11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
- 12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
- 13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

© 2007. Renesas Technology Corp., All rights reserved.