

RH850G3MH

User's Manual: Software

Renesas microcontroller

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics www.renesas.com

Rev.1.30 Dec, 2016

Notice

- 1. 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.
- 2. 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.
- 3. 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.
- 4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
- 5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"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 etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anticrime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.

- 6. 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.
- 7. 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 systems manufactured by you.
- 8. 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.
- 9. 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. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics 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.
- 10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
- 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.

NOTES FOR CMOS DEVICES

- (1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).
- (2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.
- (3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.
- (4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.
- (5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.
- (6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

How to Use This Manual

Target and Readers	This manual is intended for users who wish to understand the RH850G3M software and design application systems using these products.		
Conventions	Data significance:	Higher digits on the left and lower digits on the right	
	Active low representation:	xxx (overscore over pin or signal name)	
	Memory map address:	Higher addresses on the top and lower addresses on the bottom	
	Note:	Footnote for item marked with Note in the text	
	Caution:	Information requiring particular attention	
	Remark:	Supplementary information	
	Numeric representation:	Binary xxxx or xxxx _B	
		Decimal xxxx	
		Hexadecimal xxxx _H	
	Prefix indicating power of 2 (address space, memory capacity):		
		K (kilo): 2 ¹⁰ = 1,024	
		M (mega): 2 ²⁰ = 1,024 ²	

G (giga): 2³⁰ = 1,024³

All trademarks and registered trademarks are the property of their respective owners.

Table of Contents

Sectio	on 1	OVERVIEW	8
1.1	Fea	tures of the RH850G3MH	
1.2	Cha	nges from the RH850G3M	9
Sectio	n 2	PROCESSOR MODEL	10
2.1		J Operating Modes	
	2.1.1	Definition of CPU Operating Modes	
	2.1.2	CPU Operating Mode Transition	
	2.1.3	CPU Operating Modes and Privileges	
2.2		ruction Execution	
2.3		eptions and Interrupts	
	2.3.1	Exception Level	
2.4	Cop	rocessors	
	2.4.1	Coprocessor Use Permissions	
	2.4.2	Correspondences between Coprocessor Use Permissions and Coprocessors	
	2.4.3	Coprocessor Unusable Exceptions	
	2.4.4	System Registers	
2.5		isters	
	2.5.1	Program Registers	
	2.5.2	System Registers	
	2.5.3 2.5.4	Register Updating Accessing Undefined Registers	
26		a Types	
2.6			
	2.6.1 2.6.2	Data formats Data Representation	
	2.6.2	Data Alignment	
2.7		ress Space	
2.1		•	
	2.7.1 2.7.2	Memory Map Instruction Addressing	27 29
	2.7.2	Data Addressing	
2.8		uiring the CPU Number	
2.0	•	em Protection Identifier	
2.9	Sys		
Sectio	on 3	REGISTER SET	36
3.1	Pro	gram Registers	
	3.1.1	General-Purpose Registers	
	3.1.2	PC — Program Counter	
3.2	Bas	c System Registers	39
3.3	Inte	rupt Function Registers	58
	3.3.1	Interrupt Function System Registers	58
3.4	FPL	Function Registers	
	3.4.1	Floating-Point Registers	
	3.4.2	Floating-Point Function System Registers	

3.5	MPU	J Function Registers	69
	3.5.1	MPU Function System Registers	69
3.6	Cac	he Operation Function Registers	77
	3.6.1	Cache Control Function System Registers	77
3.7	Data	a Buffer Operation Registers	83
	3.7.1	Data Buffer Control System Registers	
	-		
Sectio	on 4	EXCEPTIONS AND INTERRUPTS	
4.1	Out	ine of Exceptions	84
	4.1.1	Exception Cause List	84
	4.1.2	Overview of Exception Causes	87
	4.1.3	Types of Exceptions	88
	4.1.4	Exception Acknowledgment Conditions and Priority Order	89
	4.1.5	Interrupt Exception Priority and Priority Masking	
	4.1.6	Return and Restoration	
	4.1.7	Context Saving	
4.2	Ope	ration When Acknowledging an Exception	93
	4.2.1	Special Operations	95
4.3	Retu	urn from Exception Handling	
4.4	Exc	eption Handler Address	
	4.4.1	Resets, Exceptions, and Interrupts	
	4.4.2	System Calls	103
	4.4.3	Models for Application	104
Sectio	n 5	MEMORY MANAGEMENT	106
	-		
5.1		nory Protection Unit (MPU)	
	5.1.1	Features	
	5.1.2	Protection Area Settings	
	5.1.3	Caution Points for Protection Area Setup	
	5.1.4	Access Control	
	5.1.5 5.1.6	Violations and Exceptions Memory Protection Setting Check Function	
5.0			
5.2			
	5.2.1	Cache Operation Registers	
	5.2.2 5.2.3	Change Cache Use Mode Cache Operations using CACHE Instruction	
	5.2.3	Cache Operation when the PREF Instruction is Executed	
	5.2.4	Cache Index Specification Method	
	5.2.6	Execution Privilege of the CACHE/PREF Instruction	
	5.2.7	Memory Protection for CACHE and PREF Instructions	
5.3		ual Exclusion	
2.0	5.3.1	Shared Data that does not Require Mutual Exclusion Processing	
	5.3.2	Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions	
	5.3.3	Performing Mutual Exclusion by Using the SET1 Instruction	

	5.3.4	Performing Mutual Exclusion by Using the CAXI Instruction	121
5.4	Sy	nchronization Function	122
Sectio	on 6	COPROCESSOR	124
6.1	Flo	pating-Point Operation	124
	6.1.1	Configuration of Floating-Point Operation Function	124
	6.1.2	Data Types	125
	6.1.3	Register Set	128
	6.1.4	Floating-Point Instructions	128
	6.1.5	Floating-Point Operation Exceptions	129
	6.1.6	Exception Details	132
	6.1.7	Saving and Returning Status	136
	6.1.8	Flushing Subnormal Numbers	
	6.1.9	Flush to Nearest	139
Sectio	on 7	INSTRUCTION	140
7.1	Op	codes and Instruction Formats	140
	7.1.1	CPU Instructions	140
	7.1.2	Coprocessor Instructions	145
	7.1.3	Reserved Instructions	145
7.2	Ba	sic Instructions	146
	7.2.1	Overview of Basic Instructions	146
	7.2.2	Basic Instruction Set	151
7.3	Ca	che Instructions	302
	7.3.1	Overview of Cache Instructions	302
	7.3.2	Cache Instruction Set	
7.4	Flo	ating-Point Instructions	307
	7.4.1	Instruction formats	
	7.4.2	Overview of Floating-Point Instructions	
	7.4.3	Conditions for Comparison Instructions	
	7.4.4	Floating-Point Instruction Set	313
Sectio	on 8	RESET	408
8.1	Sta	atus of Registers After Reset	408
APPE		с. С	
APPE			
B.1	Nu	mbers of Clock Cycles for Execution	410
B.2	2 Nu	mber of G3MH Instruction Execution Clocks	411
		C REGISTER INDEX D INSTRUCTION INDEX	

Section 1 OVERVIEW

1.1 Features of the RH850G3MH

The RH850G3MH features compatibility with the instruction set for all 32-bit RISC microcontrollers of the RH850G3M Series, but has even better performance.

Table 1.1 shows the features of the RH850G3MH.

Table 1.1 Features of the RH850G3MH

Item	Features
CPU	High performance 32-bit architecture for embedded control
	32-bit internal data bus
	Thirty-two 32-bit general-purpose registers
	 RISC type instruction set (compatible with RH850G3M) Long/short type load/store instructions Three-operand instructions Instruction set based on C
	CPU operating modes User mode and supervisor mode
	Address space: 4-Gbyte linear space for both data and instructions
Coprocessor	 A floating point operation coprocessor (FPU) can be installed. Supports single precision (32-bit) and double precision (64-bit) Supports IEEE754-compliant data types and exceptions Rounding modes : Nearest, 0 direction, +∞ direction, and -∞ direction Handling on non-normalized numbers: These are truncated to 0, or an exception is reported because such numbers do not comply with IEEE754.
Exceptions/interrupts	16-level interrupt priority that can be specified for each channel
	 Vector selection method that can be selected according to performance requirements and the amount of consumed memory Direct branch method exception vector (direct vector method) Address-table-referencing indirect branch method exception vector (table reference method)
	 Support for high-speed context backup and restoration processing on interrupt by using dedicated instructions (PUSHSP, POPSP)
Memory management	A memory protection unit (MPU) can be installed.
Caches	An instruction cache can be installed.



1.2 Changes from the RH850G3M

Item	Changes		
CPU	 The specifications of system registers listed below were changed. The specifications of MEI and MCTL were changed (see Section 3.2, Basic System Registers). 		
	 The function of the SYNCE instruction was changed (see Section 7.2.2, Basic Instruction Set). 		
	 The hazard resolution procedure does not proceed after updating of SCCFG as it was found to be unnecessary in that case (see APPENDIX A, HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS). 		
Exceptions/interrupts	 The specifications of system registers were changed and a system register was deleted as stated below. FPIPR was deleted, and the specifications of ISPR, PMR, and ICSR were changed (see Section 3.3.1, Interrupt Function System Registers). 		
	 Changes were made to the exceptions for floating-point operations. The FPP exception and FPI exception were abolished. An FPINT exception was added. 		
Coprocessor	 The specification of a system register was changed and a system register was deleted as stated below. The specification of FPSR was changed and FPEC was deleted (see Section 3.4. Floating-Point Function System Registers). 		
	 The hazard resolution procedure does not proceed after updating of any of the FP registers (FPSR, FPEPC, FPST, FPCC, or FPCFG) as it was found to be unnecessary in these cases (see APPENDIX A, HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS). 		
	 In some cases, an approximate result of the following floating-point instructions differs very small from an approximate result of the same instruction in RH850G3N RECIPF.D, RECIPF.S, RSQRTF.D and RSQRTF.S 		
Memory management	 A specification of the MPU was changed. The operation in response to memory access spanning contiguous areas to which access is enabled by the MPU was changed (see Section 5.1.3, Caution Points for Protection Area Setup). 		
Cache	 The specifications of system registers listed below were changed. The specifications of ICCTRL, ICTAGL, ICTAGH, ICCFG and ICERR were change (see Section 3.6.1, Cache Control Function System Registers). 		
	 The wait for the completion of clearing of the instruction cache by the ICCTRL.ICHCLR bit is not necessary (see APPENDIX A, HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS). 		

Table 1.2 Changes from the RH850G3M



Section 2 PROCESSOR MODEL

This CPU defines a processor model that has basic operation functions, registers, and an exception management function.

This section describes the unique features of the processor model of this CPU.

2.1 CPU Operating Modes

This CPU has defines two operating statuses of the supervisor mode (SV) and the user mode (UM). Whether the system is in supervisor mode or user mode is indicated by the UM bit in the PSW register.

- Supervisor mode (PSW.UM = 0) : All hardware functions can be managed or used.
- User mode (PSW.UM = 1) : The usable hardware functions are restricted.

2.1.1 Definition of CPU Operating Modes

(1) Supervisor mode (SV)

All hardware functions can be managed or used in this mode. The system always starts up in supervisor mode after the end of reset processing.

(2) User mode (UM)

This operating mode makes up a pair with the supervisor mode. In user mode, address spaces to which access is permitted by the supervisor and the system registers defined as user resources can be used. Supervisor-privileged instructions cannot be executed and result in exceptions if they are.

Restriction in user mode (PSW.UM = 1)

• Privileged instruction violations due to SV-privileged-instruction operating restrictions (\rightarrow PIE exceptions) For details about privileged-instruction operating restrictions, see Section 2.1.3, CPU Operating Modes and Privileges



2.1.2 CPU Operating Mode Transition

The CPU operating mode changes due to three events.

(1) Change due to acknowledging an exception

When an exception is acknowledged, the CPU operating mode changes to the mode specified for the exception.

(2) Change due to a return instruction

When a return instruction is executed, the PSW value is restored according to the value of the corresponding bit backed up to EIPSW and FEPSW.

(3) Change due to a system register instruction

The CPU operating mode changes when an LDSR instruction is used to directly overwrite the PSW operating mode bits.

CAUTIONS

- 1. In supervisor mode, the LDSR instruction can be used to directly change the value of the PSW.UM bit, but system-register-related hazards are defined in the hardware specifications.For the change of this bit, it is recommended to use a return instruction to avoid PSW-register-related hazards.
- 2. In user mode, the CPU operating mode cannot be changed because the higher 31 to 5 bits of the PSW register cannot be overwritten. The CPU operating mode might be changed in supervisor mode, but system register access-related hazards are defined in the hardware specifications. For the change of this bit, it is recommended to use a return instruction to avoid PSW-register-related hazards.



2.1.3 CPU Operating Modes and Privileges

In this CPU, the usable functions can be restricted according to usage permission settings for specific resources and the CPU operating mode. Specification instructions (including instructions that update specific system registers) can only be executed in the defined operating mode. The permissions necessary to execute these specification instructions are called "privileges" below. In operating modes that do not have privileges, these instructions are not executed and exceptions occur.

This CPU defines the following two types of privileges (and usage permission).

- Supervisor (SV) privilege : Important system resources operation, fatal error processing, privilege necessary for user-mode program execution management
- Coprocessor use permissions : Permissions necessary to use a coprocessor



Figure 2.1 CPU Operating Modes and Privileges



(1) Supervisor privilege (SV privilege)

The privilege necessary to perform the operation for important system resources, fatal error processing, and user-mode program execution management is called the supervisor privilege (SV privilege). This privilege is available in supervisor mode. The SV privilege is generally necessary to execute instructions used to perform the operation for important system resources, and these instructions are sometimes called SV privileged instructions.

(2) Coprocessor use permissions

Regardless of the CPU operating mode, it is possible to separately specify whether coprocessors can be used.

The CU2 to CU0 bits in the PSW register are used in supervisor mode to specify whether coprocessors can be used by each program. If the CU bits are not set to 1, a coprocessor unusable exception occurs when the corresponding coprocessor instruction is executed or the system register is accessed.

If no coprocessor is installed, it is not possible to set the corresponding CU bits to 1. The setting of the CU2 to CU0 bits is valid regardless of the CPU operating mode, and, if the supervisor accesses coprocessor system registers, it is necessary to set the CU2 to CU0 bits to enable coprocessor use.

(3) Operation when there is a privilege violation

When an attempt is made to execute a privileged instruction by someone who does not have the required privilege, a PIE exception or UCPOP exception occurs. **Table 2.1** shows the relationships between the operating mode, usage permission status, and whether instructions can be executed.

	PSW				
	UM	CU2	CU1	CU0	Whether Operation is Possible
SV privileged instruction	0	_	_		Possible
	1	_	_	_	Not possible/PIE exception
Coprocessor instruction 1 ^{*1}		—	_	1	Possible
(PSW.CU0 bit)		_	_	0	Not possible/UCPOP exception
Coprocessor instruction 2*1	_	—	1	—	Possible
(PSW.CU1 bit)		_	0	_	Not possible/UCPOP exception
Coprocessor instruction 3*1	—	1	_	_	Possible
(PSW.CU2 bit)	_	0	_	—	Not possible/UCPOP exception
Instructions other than the above (user instructions)	_	_	_	_	Possible

Table 2.1Operation When There is a Privilege Violation

Note 1. This includes the LDSR/STSR instruction for the coprocessor system register. **Note:** -: 0 or 1



2.2 Instruction Execution

The instruction execution flow of this CPU is shown below.



Figure 2.2 Instruction Execution Flow



If terminating exceptions can be acknowledged or if the execution privilege of the instruction is not satisfied, an exception occurs before the instruction is executed. If a resumable exception occurs during the execution of an instruction, the exception is acknowledged during execution of the instruction. In these cases, the result of instruction execution is not reflected in the registers or memory, and the CPU state before the instruction was executed is retained^{*1}.

For a pending exception such as a software exception, the exception is acknowledged after the result of instruction execution has been reflected.

Note 1. The following instructions might cause intermediate results to be reflected in the memory. PREPARE, DISPOSE, PUSHSP, POPSP



2.3 Exceptions and Interrupts

Exceptions and interrupts are exceptional events that cause the program under execution to branch to another program. Exceptions and interrupts are triggered by various sources, including interrupts from peripherals and program abnormalities.

For details, see Section 4, EXCEPTIONS AND INTERRUPTS.

2.3.1 Exception Level

In this CPU, if an exception with a high degree of urgency occurs while another exception is being processed, the urgent exception will be processed by priority. To make it possible to return to the interrupted exception handling after acknowledging the urgent exception, even if the context had not been saved to the memory, exception causes are managed in the following two hierarchical levels.

- EI level exception
- FE level exception

EI level exceptions are used for processing such as regular user processing, interrupt servicing, and OS processing. FE level exceptions are used to enable interrupts with a high degree of urgency for the system or exceptions from the memory management function that might occur during OS processing to be acknowledged even while an EI level exception is being processed.



2.4 Coprocessors

In this CPU, single-precision and double-precision FPU expansion functions are incorporated.

2.4.1 Coprocessor Use Permissions

To execute a coprocessor instruction or defined opcode processing, permission to use the corresponding coprocessor instruction is necessary. Coprocessor use permissions are specified by the PSW.CU2 to PSW.CU0 bits, and, if an attempt is made to execute an instruction for which the corresponding coprocessor use permission is cleared to 0, a coprocessor unusable exception (UCPOP) occurs.

2.4.2 Correspondences between Coprocessor Use Permissions and Coprocessors

This CPU defines coprocessor use permissions to control the availability of the coprocessor for each program during CPU operation. There are three coprocessor use permissions (CU0 to CU2), and their correspondences with the coprocessors are shown in the following table.

 Table 2.2
 Correspondences Between Coprocessor Use Permissions and Coprocessors

Coprocessor Use Permission	Coprocessor Function	Exception Cause Code
U0 Single-precision FPU expansion function		80 _H
	Double-precision FPU expansion function	_
CU1	Reserved	81 _H
CU2	Reserved	82 _H

2.4.3 Coprocessor Unusable Exceptions

A coprocessor unusable exception occurs if an attempt is made to execute a coprocessor instruction or access a system register of the coprocessor without having the corresponding coprocessor use permission (PSW.CUn = 0).

2.4.4 System Registers

Some coprocessor functions are defined by system registers. The coprocessor use permission is necessary to access the system register of a coprocessor function. For some system registers, the supervisor privilege (SV permission) is necessary in addition to the coprocessor use permission.

For details about the permissions necessary to access system registers, see Section 2.5, Registers.



2.5 Registers

This CPU defines program registers (general-purpose registers and the program counter PC) and system registers for controlling the status and storing exception information.

2.5.1 **Program Registers**

The program registers include general-purpose registers (r0 to r31) and the program counter (PC).

Category	Access Permission	Name	
Program counter	UM	PC	
General-purpose registers	UM	r0 to r31	

Note: UM: User register. This register can always be accessed because no access permission is required.

2.5.2 System Registers

For details about program registers, see Section 3.1, Program Registers.

Group numbers 0 to 3 : Registers related to basic functions Group numbers 4 to 7 : Registers related to the memory management function Group numbers 12 to 15 : Registers defined in the CPU hardware specifications Group numbers 16 and later : Reserved for future expansion

For details about system registers, see the relevant sections in Section 3, REGISTER SET.

2.5.3 Register Updating

There are several methods used to update registers. Normally, no particular restrictions apply when updating register by using an instruction. However, when updating registers by using the following instructions, some restrictions might apply, depending on the operating mode.

- LDSR
- STSR



(1) LDSR and STSR

The LDSR and STSR instructions can access all the system registers. However, If a system register is accessed without the proper permission, a PIE exception or UCPOP exception might occur. For details about the access permission for each register, see the description of system registers in **Section 3**, **REGISTER SET**. For details about behaviors when a privilege violation occurs, see **Section 2.1.3**, **CPU Operating Modes and Privileges**.

Figure 2.3 shows the flow of executing the LDSR and STSR instructions.



Figure 2.3 Flow of Executing the LDSR and STSR Instructions



2.5.4 Accessing Undefined Registers

If a system register number without any register assigned is accessed or if an inaccessible register is accessed, the following results occur.

- Undefined registers are handled as having the SV permission. When they are accessed by an LDSR or STSR instruction in user mode (PSW.UM = 1), a PIE exception occurs.
- For a read operation, the read result is undefined. If the read value is used in a program, unexpected behaviors might occur.
- For a write operation, the write operation is ignored. However, writing to the following system register numbers is prohibited. Writing prohibited: [SR11, 0], [SR1, 1], [SR7, 1], [SR10, 1], [SR13, 1], [SR14, 1], [SR15, 1], [SR16, 1], [SR5, 2], [SR20, 5]



2.6 Data Types

2.6.1 Data formats

This CPU handles data in little endian format. This means that byte 0 of a halfword or a word is always the least significant (rightmost) byte.

The supported data format is as follows.

- Byte (8-bit data)
- Halfword (16-bit data)
- Word (32-bit data)
- Double-word (64-bit data)
- Bit (1-bit data)

(1) Byte

A byte is 8 consecutive bits of data that starts from any byte boundary. Numbers from 0 to 7 are assigned to these bits, with bit 0 as the LSB (least significant bit) and bit 7 as the MSB (most significant bit). The byte address is specified as "A".



(2) Halfword

A halfword is two consecutive bytes (16 bits) of data that starts from any byte boundary. Numbers from 0 to 15 are assigned to these bits, with bit 0 as the LSB and bit 15 as the MSB. The bytes in a halfword are specified using address "A", so that the two addresses comprise byte data of "A" and "A + 1".





(3) Word

A word is four consecutive bytes (32 bits) of data that starts from any byte boundary. Numbers from 0 to 31 are assigned to these bits, with bit 0 as the LSB (least significant bit) and bit 31 as the MSB (most significant bit). A word is specified by address "A" and consists of byte data of four addresses: "A", "A + 1", "A + 2", and "A + 3".



(4) Double-word

A double-word is eight consecutive bytes (64 bits) that start from any 4-byte boundary. Numbers from 0 to 63 are assigned to these bits, with bit 0 as the LSB and bit 63 as the MSB. A double-word is specified by address "A" and consists of byte data of eight addresses: "A", "A + 1", "A + 2", "A + 3", "A + 4", "A + 5", "A + 6", and "A + 7".



(5) Bit

A bit is bit data at the nth bit within 8-bit data that starts from any byte boundary. Each bit is specified using its byte address "A" and its bit number "n" (n = 0 to 7).





2.6.2 Data Representation

(1) Integers

Integers are represented as binary values using 2's complement, and are used in one of four lengths: 64 bits, 32 bits, 16 bits, or 8 bits. Regardless of the length of an integer, its place uses bit 0 as the LSB, and this place gets higher as the bit number increases. Because this is a 2's complement representation, the MSB is used as a signed bit.

The integer ranges for various data lengths are as follows.

- Double-word (64 bits) : -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
- Word (32 bits) : -2,147,483,648 to +2,147,483,647
- Halfword (16 bits) : -32,768 to +32,767
- Byte (8 bits) : -128 to +127

(2) Unsigned integers

In contrast to "integers" which are data that can take either a positive or negative sign, "unsigned integers" are never negative integers. Like integers, unsigned integers are represented as binary values, and are used in one of four lengths: 64 bits, 32 bits, 16 bits, or 8 bits. Also like integers, the place of unsigned integers uses bit 0 as the LSB and gets higher as the bit number increases. However, unsigned integers do not use a sign bit.

The unsigned integer ranges for various data lengths are as follows.

- Double-word (64 bits): 0 to 18,446,744,073,709,551,615
- Word (32 bits): 0 to 4,294,967,295
- Halfword (16 bits): 0 to 65,535
- Byte (8 bits): 0 to 255

(3) Bits

Bit data are handled as single-bit data with either of two values: cleared (0) or set (1). There are four types of bit-related operations (listed below), which target only single-byte data in the memory space.

- Set
- Clear
- Invert
- Test



2.6.3 Data Alignment

When the result of address calculation is a misaligned address, a misaligned access exception (MAE) occurs.

When the data to be processed is in halfword format, misaligned access indicates the access to an address that is not at the halfword boundary (where the address LSB = 0), and when the data to be processed is in word format, misaligned access indicates the access to an address that is not at the word boundary (where the lower two bits of the address = 0). When the data to be processed is in double-word format, misaligned access indicates the access to an address that is not at the double-word format, where the lower 3 bits of the address = 0).

For the double-word format only, a misaligned access exception does not occur when data is placed at the word boundary rather than the double-word boundary.

CAUTIONS

- 1. The following instructions might possibly cause misaligned access. For details, see the relevant descriptions in Section 7, INSTRUCTION.
 - LD.H, LD.HU, LD.W, LD.DW
 - SLD.H, SLD.HU, SLD.W
 - ST.H, ST.W, ST.DW
 - SST.H, SST.W
 - LDL.W, STC.W, CAXI
- 2. The following instructions do not cause misaligned access, because the address is rounded in the instruction specification when the alignment specification is incorrect.
 - PREPARE, DISPOSE
 - PUSHSP, POPSP





RENESAS

2.7 Address Space

This CPU supports a linear address space of up to 4 Gbytes. Both memory and I/O can be mapped to this address space (using the memory mapped I/O method). The CPU outputs a 32-bit address for memory and I/O, in which the highest address number is " $2^{32} - 1$ ".

The byte data placed at various addresses is defined with bit 0 as the LSB and bit 7 as the MSB. When the data is comprised of multiple bytes, it is defined so that the byte data at the lowest address is the LSB and the byte data at the highest address is the MSB (i.e., in little endian format).

This manual stipulates that, when representing data comprised of multiple bytes, the right edge must be represented as the lower address and the left side as the upper address, as shown below.



Figure 2.5 Address Space Byte Format



2.7.1 Memory Map

This CPU is 32-bit architecture and supports a linear address space of up to 4 Gbytes. The whole range of this 4-Gbyte address space can be addressed by instruction addressing (instruction access) and operand addressing (data access).

A memory map is shown in **Figure 2.6**.



Figure 2.6 Memory Map (Address Space)



2.7.2 Instruction Addressing

The instruction address is determined based on the contents of the program counter (PC), and is automatically incremented according to the number of bytes in the executed instruction. When a branch instruction is executed, the addressing shown below is used to set the branch destination address to the PC.

(1) Relative addressing (PC relative)

Signed N-bit data (displacement: disp N) is added to the instruction code in the program counter (PC). In this case, displacement is handled as 2's complement data, and the MSB is a signed bit (S). If the displacement is less than 32 bits, the higher bits are sign-extended (N differs from one instruction to another).





Figure 2.7 Relative Addressing

(2) Register addressing (register indirect)

The contents of the general-purpose register (reg1) or system register (regID) specified by the instruction are transferred to the program counter (PC).

The JMP, CTRET, EIRET, FERET, and DISPOSE instructions are used with this type of addressing.



Figure 2.8 Register Addressing

(3) Based addressing

Contents that are specified by the instruction in the general-purpose register (reg1) and that include the added N-bit displacement (dispN) are transferred to the program counter (PC). At this time, the displacement is handled as a 2's complement data, and the MSB is a signed bit (S). If the displacement is less than 32 bits, the higher bits are sign-extended (N differs from one instruction to another).

The JMP instruction is used with this type of addressing.



Figure 2.9 Based Addressing

(4) Other addressing

A value specified by an instruction is transferred to the program counter (PC). How a value is specified is explained in [Operation] or [Description] of each instruction.

The CALLT, SYSCALL, TRAP, FETRAP, and RIE instructions, and branch in case of an exception are used with this type of addressing.



2.7.3 Data Addressing

The following methods can be used to access the target registers or memory when executing an instruction.

(1) Register addressing

This addressing method accesses the general-purpose register or system register specified in the general-purpose register field as an operand.

Any instruction that includes the operand reg1, reg2, reg3, or regID is used with this type of addressing.

(2) Immediate addressing

This address mode uses arbitrary size data as the operation target in the instruction code.

Any instruction that includes the operand imm5, imm16, vector, or cccc is used with this type of addressing.

NOTE

- vector : This is immediate data that specifies the exception vector (00_H to 1F_H), and is an operand used by the TRAP, FETRAP, and SYSCALL instructions. The data width differs from one instruction to another.
- cccc : This is 4-bit data that specifies a condition code, and is an operand used in the CMOV instruction, SASF instruction, and SETF instruction. One bit (0) is added to the higher position and is then assigned to an opcode as a 5-bit immediate data.

(3) Based addressing

There are two types of based addressing, as described below.

(a) Type 1

The contents of the general-purpose register (reg1) specified at the addressing specification field in the instruction code are added to the N-bit displacement (dispN) data sign-extended to word length to obtain the operand address, and addressing accesses the target memory for the operation. At this time, the displacement is handled as a 2's complement data, and the MSB is a signed bit (S). If the displacement is less than 32 bits, the higher bits are sign-extended (N differs from one instruction to another).







(b) Type 2

This addressing accesses a memory to be manipulated by using as an operand address the sum of the contents of the element pointer (r30) and N-bit displacement data (dispN) that is zeroextended to a word length. If the displacement is less than 32 bits, the higher bits are signextended (N differs from one instruction to another).

The SLD instruction and SST instruction are used with this type of addressing.



Figure 2.11 Based Addressing (Type 2)

(4) Bit addressing

The contents of the general-purpose register (reg1) are added to the N-bit displacement (dispN) data sign-extended to word length to obtain the operand address, and bit addressing accesses one bit (as specified by 3-bit data "bit #3") in one byte of the target memory space. At this time, the displacement is handled as a 2's complement data, and the MSB is a signed bit (S). If the displacement is less than 32 bits, the higher bits are sign-extended (N differs from one instruction to another).

The CLR1, SET1, NOT1, and TST1 instructions are used with this type of addressing.



Figure 2.12 Bit Addressing

(5) Post index increment/decrement addressing

The contents of the general-purpose register (reg1) are used as an operand address to access the target memory, and then the general-purpose register (reg1) is updated. The register is updated by either incrementing or decrementing it, and there are three types (1 to 3).

If the result of incrementing the general-purpose register (reg1) value exceeds the positive maximum value 0xFFFF FFFF, the result wraps around to 0x0000 0000, and, if the result of decrementing the general-purpose register value is less than the positive minimum value 0x0000 0000, the result wraps around to 0xFFFF FFFF.

(a) Type 1

The general-purpose register (reg1) is updated by adding a constant that depends on the type of accessed data (the size of the accessed data) to the contents of the general-purpose register (reg1). If the type of accessed data is a byte, 1 is added, if the type is a halfword, 2 is added, if the type is a word, 4 is added, and if the type is a double-word, 8 is added.



Figure 2.13 Post Index Increment/Decrement Addressing (Type 1)



(b) Type 2

The general-purpose register (reg1) is updated by subtracting a constant that depends on the size of the accessed data from the contents of the general-purpose register (reg1). If the size of accessed data is a byte, 1 is subtracted, if the size is a halfword, 2 is subtracted, if the size is a word, 4 is subtracted, and if the size is a double-word, 8 is subtracted.



Figure 2.14 Post Index Increment/Decrement Addressing (Type 2)

(c) Type 3

The general-purpose register (reg1) is updated by adding the contents of another general-purpose register (reg2) to it. If the MSB of the general-purpose register (reg2) is 1, a negative value is indicated, so a post decrement operation is performed. If this MSB is 0, a positive value is indicated, so a post increment operation is performed. The value of the general-purpose register (reg2) does not change.



Figure 2.15 Post Index Increment/Decrement Addressing (Type 3)

(6) Other addressing

This addressing is to access a memory to be manipulated by using a value specified by an instruction as the operand address. How a value is specified is explained in [Operation] or [Description] of each instruction.

The SWITCH, CALLT, SYSCALL, PREPARE, DISPOSE, PUSHSP, and POPSP instructions are used with this type of addressing.



2.8 Acquiring the CPU Number

This CPU provides a method for identifying CPUs in a multi-processor system.

In the multi-processor configuration, you can identify which CPU core is running a program by referencing HTCFG0.PEID. With HTCFG0.PEID, unique numbers are assigned within multi-processor systems.

2.9 System Protection Identifier

In this CPU, memory resources and peripheral devices are managed by system protection groups. By specifying the group to which the program being executed belongs, you can assign operable memory resources and peripheral devices to each machine.

The program being executed belongs to the group shown by MCFG0.SPID, and whether the memory resources and peripheral devices are operable is decided using this SPID. Any value can be set to MCFG0.SPID by the supervisor.

CAUTION

According to the value of MCFG0.SPID, how operations are assigned to memory resources and peripheral devices is determined by the hardware specifications.



Section 3 REGISTER SET

This chapter describes the program register and system register mounted on this CPU.

3.1 Program Registers

Program registers includes general-purpose registers (r0 to r31) and the program counter (PC). r0 always retains 0, whereas the value after reset is undefined in r1 to r31.

Program Register	Name	Function	Description	
General-purpose	r0	Zero register	Always retains 0	
registers	r1	Assembler reserved register	Used as working register for generating addresses	
	r2	Register for address and data variables (used when the real-time OS used does not use this register)		
	r3	Stack pointer (SP)	Used for generating a stack frame when a function is called	
	r4	Global pointer (GP)	Used for accessing a global variable in the data area	
	r5	Text pointer (TP)	Used as a register that indicates the start of the text area (area where program code is placed)	
	r6 to r29	Register for addresses and data variables		
	r30	Element pointer (EP)	Used as a base pointer for generating addresses when accessing memory	
	r31	Link pointer (LP)	Used when the compiler calls a function	
Program counter	PC	Retains instruction addresses during execution of programs		

Table 3.1 Program Registers

Note: For further descriptions of r1, r3 to r5, and r31 used for an assembler and/or C compiler, see the manual of each software development environment.


3.1.1 General-Purpose Registers

A total of 32 general-purpose registers (r0 to r31) are provided. All of these registers can be used for either data variables or address variables.

Of the general-purpose registers, r0 to r5, r30, and r31 are assumed to be used for special purposes in software development environments, so it is necessary to note the following when using them.

(a) r0, r3, and r30

These registers are implicitly used by instructions.

r0 is a register that always retains 0. It is used for operations that use 0, addressing with base address being 0, etc.

r3 is implicitly used by the PREPARE, DISPOSE, PUSHSP, and POPSP instructions.

r30 is used as a base pointer when the SLD instruction or SST instruction accesses memory.

(b) r1, r4, r5, and r31

These registers are implicitly used by the assembler and C compiler.

When using these registers, register contents must first be saved so they are not lost and can be restored after the registers are used.

(c) r2

This register is used by a real-time OS in some cases. If the real-time OS that is being used is not using r2, r2 can be used as a register for address variables or data variables.



3.1.2 PC — Program Counter

The PC retains the address of the instruction being executed.

 Table 3.2
 PC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	PC31 to PC1	These bits indicate the address of the instruction being executed.	R/W	*1
0	PC0	This bit is fixed to 0. Branching to an odd number address is disabled.	R/W	0

Note 1. For details, see the hardware manual of the product used.



3.2 Basic System Registers

The basic system registers are used to control CPU status and to retain exception information.

Basic system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID.

Table 3.3 Basic System Registers

Register No. (regID, seIID)	Symbol	Function	Access Permission
SR0, 0	EIPC	Status save registers when acknowledging EI level exception	SV
SR1, 0	EIPSW	Status save registers when acknowledging EI level exception	SV
SR2, 0	FEPC	Status save registers when acknowledging FE level exception	SV
SR3, 0	FEPSW	Status save registers when acknowledging FE level exception	SV
SR5, 0	PSW	Program status word	*1
SR6, 0	FPSR	(See Section 3.4, FPU Function Registers)	CU0 and SV
SR7, 0	FPEPC	(See Section 3.4, FPU Function Registers)	CU0 and SV
SR8, 0	FPST	(See Section 3.4, FPU Function Registers)	CU0
SR9, 0	FPCC	(See Section 3.4, FPU Function Registers)	CU0
SR10, 0	FPCFG	(See Section 3.4, FPU Function Registers)	CU0
SR13, 0	EIIC	El level exception cause	SV
SR14, 0	FEIC	FE level exception cause	SV
SR16, 0	CTPC	CALLT execution status save register	UM
SR17, 0	CTPSW	CALLT execution status save register	UM
SR20, 0	CTBP	CALLT base pointer	UM
SR28, 0	EIWR	El level exception working register	SV
SR29, 0	FEWR	FE level exception working register	SV
SR31, 0	(BSEL)	(Reserved for backward compatibility with V850E2 series)* ²	SV
SR0, 1	MCFG0	Machine configuration	SV
SR2, 1	RBASE	Reset vector base address	SV
SR3, 1	EBASE	Exception handler vector address	SV
SR4, 1	INTBP	Base address of the interrupt handler table	SV
SR5, 1	MCTL	CPU control	SV
SR6, 1	PID	Processor ID	SV
SR11, 1	SCCFG	SYSCALL operation setting	SV
SR12, 1	SCBP	SYSCALL base pointer	SV
SR0, 2	HTCFG0	Thread configuration	SV
SR6, 2	MEA	Memory error address	SV
SR7, 2	ASID	Address space ID	SV
SR8, 2	MEI	Memory error information	SV



- Note 1. The access permission differs depending on the bit. For details, see (5), PSW Program status word in Section 3.2, Basic System Registers.
- Note 2. This bit is reserved to maintain backward compatibility with V850E2 series. This bit is always 0 when read. Writing to this bit is ignored.

(1) EIPC — Status save register when acknowledging El level exception

When an EI level exception is acknowledged, the address of the instruction that was being executed when the EI level exception occurred, or of the next instruction, is saved to the EIPC register (see **Section 4.1.3, Types of Exceptions**).

Because there is only one pair of EI level exception status save registers, when processing multiple exceptions, the contents of these registers must be saved by a program.

Be sure to set an even-numbered address to the EIPC register. An odd-numbered address must not be specified.



Table 3.4EIPC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	EIPC31 to EIPC1	These bits indicate the PC saved when an EI level exception is acknowledged.	R/W	Undefined
0	EIPC0	This bit indicates the PC saved when an EI level exception is acknowledged. Always set this bit to 0. Even if it is set to 1, the value transferred to the PC when the EIRET instruction is executed is 0.	R/W	Undefined



(2) EIPSW — Status save register when acknowledging EI level exception

When an EI level exception is acknowledged, the current PSW setting is saved to the EIPSW register.

Because there is only one pair of EI level exception status save registers, when processing multiple exceptions, the contents of these registers must be saved by a program.

CAUTION

Bits 11 to 9 are related to the debug function and therefore cannot normally be changed.



Table 3.5	FIPSW	Register	Contents
		Negister	Contents

Bit	Name	Description	R/W	Value after Reset
31		(Reserved for future expansion. Be sure to set to 0.)	R	0
30	UM	This bit stores the PSW.UM bit setting when an EI level exception is acknowledged.	R/W	0
29 to 19		(Reserved for future expansion. Be sure to set to 0.)	R	0
18 to 16	CU2 to CU0	These bits store the PSW.CU2-0 field setting when an EI level exception is acknowledged. (CU2-1 are reserved for future expansion. Be sure to set to 0.)	R/W	0
15	EBV	This bit stores the PSW.EBV bit setting when an EI level exception is acknowledged.	R/W	0
14 to 12	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
11 to 9	Debug	These bits store the PSW.Debug field setting when an EI level exception is acknowledged.	R/W	0
8	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
7	NP	This bit stores the PSW.NP bit setting when an EI level exception is acknowledged.	R/W	0
6	EP	This bit stores the PSW.EP bit setting when an EI level exception is acknowledged.	R/W	0
5	ID	This bit stores the PSW.ID bit setting when an EI level exception is acknowledged.	R/W	1
4	SAT	This bit stores the PSW.SAT bit setting when an EI level exception is acknowledged.	R/W	0
3	CY	This bit stores the PSW.CY bit setting when an EI level exception is acknowledged.	R/W	0
2	OV	This bit stores the PSW.OV bit setting when an EI level exception is acknowledged.	R/W	0
1	S	This bit stores the PSW.S bit setting when an EI level exception is acknowledged.	R/W	0
0	Z	This bit stores the PSW.Z bit setting when an EI level exception is acknowledged.	R/W	0



(3) FEPC — Status save register when acknowledging FE level exception

When an FE level exception is acknowledged, the address of the instruction that was being executed when the FE level exception occurred, or of the next instruction, is saved to the FEPC register (see **Section 4.1.3, Types of Exceptions**). Because there is only one pair of FE level exception status save registers, when processing multiple exceptions, the contents of these registers must be saved by a program.

Be sure to set an even-numbered address to the FEPC register. An odd-numbered address must not be specified.



Table 3.6 FEPC Regist	ter Contents
-----------------------	--------------

Bit	Name	Description	R/W	Value after Reset
31 to 1	FEPC31 to FEPC1	These bits indicate the PC saved when an FE level exception is acknowledged.	R/W	Undefined
0	FEPC0	This bit indicates the PC saved when an FE level exception is acknowledged. Always set this bit to 0. Even if it is set to 1, the value transferred to the PC when the FERET instruction is executed is 0.	R/W	Undefined



(4) FEPSW — Status save register when acknowledging FE level exception

When an FE level exception is acknowledged, the current PSW setting is saved to the FEPSW register.

Because there is only one pair of FE level exception status save registers, when processing multiple exceptions, the contents of these registers must be saved by a program.

CAUTION

Bits 11 to 9 are related to the debug function and therefore cannot normally be changed.



ontents
1

Bit	Name	Description	R/W	Value after Reset
31		(Reserved for future expansion. Be sure to set to 0.)	R	0
30	UM	This bit stores the PSW.UM bit setting when an FE level exception is acknowledged.	R/W	0
29 to 19		(Reserved for future expansion. Be sure to set to 0.)	R	0
18 to 16	CU2 to CU0	These bits store the PSW.CU2-0 field setting when an FE level exception is acknowledged. (CU2-1 are reserved for future expansion. Be sure to set to 0.)	R/W	0
15	EBV	This bit stores the PSW.EBV bit setting when an FE level exception is acknowledged.	R/W	0
14 to 12	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
11 to 9	Debug	These bits store the PSW.Debug field setting when an FE level exception is acknowledged.	R/W	0
8		(Reserved for future expansion. Be sure to set to 0.)	R	0
7	NP	This bit stores the PSW.NP bit setting when an FE level exception is acknowledged.	R/W	0
6	EP	This bit stores the PSW.EP bit setting when an FE level exception is acknowledged.	R/W	0
5	ID	This bit stores the PSW.ID bit setting when an FE level exception is acknowledged.	R/W	1
4	SAT	This bit stores the PSW.SAT bit setting when an FE level exception is acknowledged.	R/W	0
3	CY	This bit stores the PSW.CY bit setting when an FE level exception is acknowledged.	R/W	0
2	OV	This bit stores the PSW.OV bit setting when an FE level exception is acknowledged.	R/W	0
1	S	This bit stores the PSW.S bit setting when an FE level exception is acknowledged.	R/W	0
0	Z	This bit stores the PSW.Z bit setting when an FE level exception is acknowledged.	R/W	0



(5) PSW — Program status word

PSW (program status word) is a set of flags that indicate the program status (instruction execution result) and bits that indicate the operation status of the CPU (flags are bits in the PSW that are referenced by a condition instruction (Bcond, CMOV, etc.)).

CAUTIONS

- 1. When the LDSR instruction is used to change the contents of bits 7 to 0 in this register, the changed contents become valid from the instruction following the LDSR instruction.
- 2. The access permission for the PSW register differs depending on the bit. All bits can be read, but some bits can only be written under certain conditions. See Table 3.8 for the access permission for each bit.

		Access Permission When	
Bit		Reading	Access Permission When Writing
30	UM	UM	SV ^{*1}
18 to 16	CU2 to CU0		SV*1
15	EBV		SV*1
11 to 9	Debug		Special ^{*1}
7	NP		SV*1
6	EP		SV ^{*1}
5	ID		SV ^{*1}
4	SAT		UM
3	CY		UM
2	OV		UM
1	S	_	UM
0	Z		UM

Table 3.8 Access Permission for PSW Register

Note 1. The access permission for the whole PSW register is UM, so the PIE exception does not occur even if the register is written by using an LDSR instruction when PSW.UM is 1. In this case, writing is ignored.





			-	
Table 3.9	PSW	Register	Contents	(1/2)

Bit	Name	Description	R/W	Value after Reset
31	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
30	UM	This bit indicates that the CPU is in user mode (in UM mode). 0: Supervisor mode 1: User mode	R/W	0
29 to 19		(Reserved for future expansion. Be sure to set to 0.)	R	0
18 to 16	CU2 to CU0	These bits indicate the coprocessor use permissions. When the bit corresponding to the coprocessor is 0, a coprocessor unusable exception occurs if an instruction for the coprocessor is executed or a coprocessor resource (system register) is accessed. CU2 bit 18: (Reserved for future expansion. Be sure to set to 0.) CU1 bit 17: (Reserved for future expansion. Be sure to set to 0.) CU0 bit 16: FPU	R/W	000
15	EBV	This bit indicates the reset vector and exception vector operation. See the description on RBASE and EBASE in this section.	R/W	0
14 to 12	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
11 to 9	Debug	This bit is used for the debug function for the development tool. Always set this bit to 0.	_	0
8	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
7	NP	This bit disables the acknowledgement of FE level exception. When an FE level exception is acknowledged, this bit is set to 1 to disable the acknowledgement of EI level and FE level exceptions. As for the exceptions which the NP bit disables the acknowledgment, see Table 4.1, Exception Cause List . 0: The acknowledgement of FE level exception is enabled. 1: The acknowledgement of FE level exception is disabled.	R/W	0
6	EP	This bit indicates that an exception other than an interrupt is being serviced. It is set to 1 when the corresponding exception occurs. This bit does not affect acknowledging an exception request even when it is set to 1. 0: An exception other than an interrupt is not being serviced. 1: An exception other than an interrupt is being serviced.	R/W	0
5	ID	This bit disables the acknowledgement of EI level exception. When an EI level or FE level exception is acknowledged, this bit is set to 1 to disable the acknowledgement of EI level exception. As for the exceptions which the ID bit disables the acknowledgment, see Table 4.1, Exception Cause List . This bit is also used to disable EI level exceptions from being acknowledged as a critical section while an ordinary program or interrupt is being serviced. It is set to 1 when the DI instruction is executed, and cleared to 0 when the EI instruction is executed. The change of the ID bit by the EI or ID instruction will be enabled from the next instruction. 0: The acknowledgement of EI level exception is enabled. 1: The acknowledgement of EI level exception is disabled.	R/W	1
4	SAT* ¹	This bit indicates that the operation result is saturated because the result of a saturated operation instruction operation has overflowed. This is a cumulative flag, so when the operation result of the saturated operation instruction becomes saturated, this bit is set to 1, but it is not later cleared to 0 when the operation result for a subsequent instruction is not saturated. This bit is cleared to 0 by the LDSR instruction. This bit is neither set to 1 nor cleared to 0 when an arithmetic operation instruction is executed. 0: Not saturated 1: Saturated	R/W	0

RENESAS

Bit	Name	Description	R/W	Value after Reset
3	CY	This bit indicates whether a carry or borrow has occurred in the operation result. 0: Carry and borrow have not occurred. 1: Carry or borrow has occurred.	R/W	0
2	OV* ¹	This bit indicates whether or not an overflow has occurred during an operation. 0: Overflow has not occurred. 1: Overflow has occurred.	R/W	0
1	S* ¹	This bit indicates whether or not the result of an operation is negative. 0: Result of operation is positive or 0. 1: Result of operation is negative.	R/W	0
0	Z	This bit indicates whether or not the result of an operation is 0. 0: Result of operation is not 0. 1: Result of operation is 0.	R/W	0

Table 3.9 PSW Register Contents (2/2)

Note 1. The operation result of the saturation processing is determined in accordance with the contents of the OV flag and S flag during a saturated operation. When only the OV flag is set to 1 during a saturated operation, the SAT flag is set to 1.

		Operation Result after		
Operation Result Status	SAT	OV S		Saturation Processing
Exceeded positive maximum value	1	1	0	7FFF FFFF _H
Exceeded negative maximum value	1	1	1	8000 0000 _H
Positive (maximum value not exceeded)	Value prior to	0	0	Operation result itself
Negative (maximum value not exceeded)	operation is retained.		1	

(6) EIIC — EI level exception cause

The EIIC register retains the cause of any EI level exception that occurs. The value retained in this register is an exception code corresponding to a specific exception cause (see **Table 4.1, Exception Cause List**).



Table 3.10EIIC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	EIIC31 to EIIC0	These bits store the exception cause code when an EI level exception occurs. The EIIC15-0 field stores the exception cause codes shown in Table 4.1 . The EIIC31-16 field stores detailed exception cause codes defined individually for each exception. If there is no particular definition, these bits are set to 0.	R/W	0

(7) FEIC — FE level exception cause

The FEIC register retains the cause of any FE level exception that occurs. The value retained in this register is an exception code corresponding to a specific exception cause (see **Table 4.1, Exception Cause List**).



Table	3 11	FFIC	Register	Contents
lanc	3.11		Negister	Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	FEIC31 to FEIC0	These bits store the exception cause code when an FE level exception occurs. The FEIC15-0 field stores the exception cause codes shown in Table 4.1 . The FEIC31-16 field stores detailed exception cause codes defined individually for each exception. If there is no particular definition, these bits are set to 0.	R/W	0

(8) CTPC — Status save register when executing CALLT

When a CALLT instruction is executed, the address of the next instruction after the CALLT instruction is saved to CTPC.

Be sure to set an even-numbered address to the CTPC register. An odd-numbered address must not be specified.



Table 3.12 CTPC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	CTPC31 to CTPC1	These bits indicate the PC of the instruction after the CALLT instruction.	R/W	Undefined
0	CTPC0	This bit indicates the PC of the instruction after the CALLT instruction. Always set this bit to 0. Even if it is set to 1, the value transferred to the PC when the CTRET instruction is executed is 0.	R/W	Undefined



(9) CTPSW — Status save register when executing CALLT

When a CALLT instruction is executed, some of the PSW (program status word) settings are saved to CTPSW.



Bit	Name	Description	R/W	Value after Reset
31 to 5	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
4	SAT	This bit stores the PSW.SAT bit setting when the CALLT instruction is executed.	R/W	0
3	CY	This bit stores the PSW.CY bit setting when the CALLT instruction is executed.	R/W	0
2	OV	This bit stores the PSW.OV bit setting when the CALLT instruction is executed.	R/W	0
1	S	This bit stores the PSW.S bit setting when the CALLT instruction is executed.	R/W	0
0	Z	This bit stores the PSW.Z bit setting when the CALLT instruction is executed.	R/W	0

Table 3.13 CTPSW Register Contents

(10) CTBP — CALLT base pointer

The CTBP register is used to specify table addresses of the CALLT instruction and generate target addresses.

Be sure to set the CTBP register to a halfword address.



Table 3.14 CTBP Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	CTBP31 to CTBP1	These bits indicate the base pointer address of the CALLT instruction. These bits indicate the start address of the table used by the CALLT instruction.	R/W	Undefined
0	CTBP0	This bit indicates the base pointer address of the CALLT instruction. These bits indicate the start address of the table used by the CALLT instruction. Always set this bit to 0.	R	0



(11) ASID — Address space ID

This is the address space ID. This is used to identify the address space provided by the memory management function.



Table 3.15 ASID Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 10	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
9 to 0	ASID	This is the address space ID.	R/W	Undefined

(12) EIWR — EI level exception working register

The EIWR register is used as a working register when an EI level exception has occurred.



Bit	Name	Description	R/W	Value after Reset
31 to 0	EIWR31 to EIWR0	These bits constitute a working register that can be used for any purpose during the processing of an EI level exception. Use this register for purposes such as storing the values of general-purpose registers.	R/W	Undefined

(13) FEWR — FE level exception working register

The FEWR register is used as a working register when an FE level exception has occurred.



 Table 3.17
 FEWR Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	FEWR31 to FEWR0	These bits constitute a working register that can be used for any purpose during the processing of an FE level exception. Use this register for purposes such as storing the values of general-purpose registers.	R/W	Undefined

(14) HTCFG0 — Thread configuration register



Table 3.18 HTCFG0 Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 19	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
18 to 16	PEID	These bits indicate the processor element number.	R	*2
15	_	(Reserved for future expansion. Be sure to set to 1.)	R	1
14 to 0	_	(Reserved for future expansion. Be sure to set to 0.)	R	0

Note 1. The value after reset depends on the hardware specifications. For details, see the hardware manual of the product used.

Note 2. When these bits are read, the CPU processor identifier defined in the product specifications is read. These bits cannot be written. For details, see the hardware manual of the product used.

(15) MEA — Memory error address register



Table 3.19MEA Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MEA	These bits store the violation address when an MAE (misaligned) or MPU occurs.	R/W	Undefined



(16) MEI — Memory error information register

This register is used to store information about the instruction that caused the exception when a misaligned (MAE) or memory protection (MDP) exception occurs.



Bit	Name	Description	R/W	Value after Reset
31 to 21	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
20 to 16	REG	These bits indicate the number of the source or destination register accessed by the instruction that caused the exception. For details, see Table 3.21	R/W	Undefined
15 to 11	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
10, 9	DS	These bits indicate the type of data handled by the instruction that caused the exception. ^{Note} 0: Byte (8 bits) 1: Halfword (16 bits) 2: Word (32 bits) 3: Double-word (64 bits) For details, see Table 3.21	R/W	Undefined
8	U	This bit indicates the sign extension method of the instruction that caused the exception. 0: Signed 1: Unsigned For details, see Table 3.21	R/W	Undefined
7, 6	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
5 to 1	ITYPE	These bits indicate the instruction that caused the exception. For details, see Table 3.21	R/W	Undefined
0	RW	This bit indicates whether the operation of the instruction that caused the exception was read (Load-memory) or write (Store-memory). 0: Read (Load-memory) 1: Write (Store-memory) For details, see Table 3.21	R/W	Undefined

Table 3.20 MEI Register Contents

Note: Even if the data is divided and access is made several times due to the specifications of the hardware, the original data type indicated by the instruction is stored.

 Table 3.21
 Instructions Causing Exceptions and Values of MEI Register (1/2)

		-	-	-	
Instruction	REG	DS	U	RW	ITYPE
SLD.B	dst	0 (Byte)	0 (Signed)	0 (Read)	00000b
SLD.BU	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00000b
SLD.H	dst	1 (Half-word)	0 (Signed)	0 (Read)	00000b
SLD.HU	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00000b
SLD.W	dst	2 (Word)	0 (Signed)	0 (Read)	00000b
SST.B	SrC	0 (Byte)	0 (Signed)	1 (Write)	00000b
SST.H	SrC	1 (Half-word)	0 (Signed)	1 (Write)	00000b
SST.W	SrC	2 (Word)	0 (Signed)	1 (Write)	00000b
LD.B (disp16)	dst	0 (Byte)	0 (Signed)	0 (Read)	00001b
LD.BU (disp16)	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00001b
LD.H (disp16)	dst	1 (Half-word)	0 (Signed)	0 (Read)	00001b
LD.HU (disp16)	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00001b



RH850G3MH Software

Table	3.21	
-------	------	--

Instructions Causing Exceptions and Values of MEI Register (2/2)

Instruction	REG	DS	U	RW	ITYPE
LD.W (disp16)	dst	2 (Word)	0 (Signed)	0 (Read)	00001b
ST.B (disp16)	src	0 (Byte)	0 (Signed)	1 (Write)	00001b
ST.H (disp16)	src	1 (Half-word)	0 (Signed)	1 (Write)	00001b
ST.W (disp16)	src	2 (Word)	0 (Signed)	1 (Write)	00001b
LD.B (disp23)	dst	0 (Byte)	0 (Signed)	0 (Read)	00010b
LD.BU (disp23)	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00010b
LD.H (disp23)	dst	1 (Half-word)	0 (Signed)	0 (Read)	00010b
LD.HU (disp23)	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00010b
LD.W (disp23)	dst	2 (Word)	0 (Signed)	0 (Read)	00010b
ST.B (disp23)	src	0 (Byte)	0 (Signed)	1 (Write)	00010b
ST.H (disp23)	src	1 (Half-word)	0 (Signed)	1 (Write)	00010b
ST.W (disp23)	src	2 (Word)	0 (Signed)	1 (Write)	00010b
LD.DW (disp23)	dst	3 (Double-word)	0 (Signed)	0 (Read)	00010b
ST.DW (disp23)	src	3 (Double-word)	0 (Signed)	1 (Write)	00010b
LDL.W	dst	2 (Word)	0 (Signed)	0 (Read)	00111b
STC.W	src	2 (Word)	0 (Signed)	1 (Write)	00111b
CAXI	dst	2 (Word)	0 (Signed)	0 (Read) ^{*1}	01000b
SET1	_	0 (Byte)	0 (Signed)	0 (Read) ^{*1}	01001b
CLR1	_	0 (Byte)	0 (Signed)	0 (Read) ^{*1}	01001b
NOT1	_	0 (Byte)	0 (Signed)	0 (Read) ^{*1}	01001b
TST1	_	0 (Byte)	0 (Signed)	0 (Read)	01001b
PREPARE	src	2 (Word)	0 (Signed)	1 (Write)	01100b
DISPOSE	dst	2 (Word)	0 (Signed)	0 (Read)	01100b
PUSHSP	src	2 (Word)	0 (Signed)	1 (Write)	01101b
POPSP	dst ^{*3}	2 (Word)	0 (Signed)	0 (Read)	01101b
SWITCH	_	1 (Half-word)	0 (Signed)	0 (Read)	10000b
CALLT	_	1 (Half-word)	1 (Unsigned)	0 (Read)	10001b
SYSCALL	_	2 (Word)	0 (Signed)	0 (Read)	10010b
CACHE	_	_	_	0 (Read)	10100b
Interrupt (table reference method) ^{*2}	_	2 (Word)	0 (Signed)	0 (Read)	10101b

Note 1. This exception occurs when the instruction executes a read access.

Note 2. When the interrupt vector of the table reference method is read.

Note 3. When the destination is r3, 0 is stored.

Note: dst: Destination register number, src: Source register number



(17) RBASE — Reset vector base address

This register indicates the reset vector address when there is a reset. If the PSW.EBV bit is 0, this vector address is also used as the exception vector address.



Table 3.22	RBASE R	egister	Contents
------------	----------------	---------	----------

Bit	Name	Description	R/W	Value after Reset
31 to 9	RBASE31 to RBASE9	These bits indicate the reset vector when there is a reset. When PSW.EBV = 0, this address is also used as the exception vector. The RBASE8-0 bits are not assigned as names because these bits are always 0.	R	Note
8 to 1	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	RINT	When the RINT bit is set, the exception handler address for interrupt processing is reduced. See 4.4.1 (1), Direct vector method . This bit is valid when PSW.EBV = 0.	R	Note

Note 1. The value after reset depends on the hardware specifications. For details, see the hardware manual of the product used.

(18) EBASE — Exception handler vector address

This register indicates the exception handler vector address. This register is valid when the PSW.EBV bit is 1.



Table 3.23 EBASE Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 9	EBASE31 to EBASE9	The exception handler routine address is changed to the address resulting from adding the offset address of each exception to the base address specified for this register. The EBASE8-0 bits are not assigned as names because these bits are always 0.	R/W	Undefined
8 to 1	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	RINT	When the RINT bit is set, the exception handler address for interrupt processing is reduced. See 4.4.1 (1), Direct vector method .	R/W	Undefined

(19) INTBP — Base address of the interrupt handler address table

This register indicates the base address of the table when the table reference method is selected as the interrupt handler address selection method.



Table 3.24 INTBP Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 9	INTBP31 to INTBP9	These bits indicate the base pointer address for an interrupt when the table reference method is used. The value indicated by these bits is the first address in the table used to determine the exception handler when the interrupt specified by the table reference method (EIINT0 to EIINT511) is acknowledged. The INTBP8-0 bits are not assigned as names because these bits are always 0.	R/W	Undefined
8 to 0	_	(Reserved for future expansion. Be sure to set to 0.)	R	0



(20) PID — Processor ID

The PID register retains a processor identifier that is unique to the CPU. The PID register is a read-only register.

CAUTION

The PID register indicates information used to identify the incorporated CPU core and CPU core configuration. Usage such that the software behavior varies dynamically according to the PID register information is not assumed.

Table 3.25PID Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 24	PID	Architecture Identifier This identifier indicates the architecture of the processor.	R	*1
23 to 8	_	Function Identifier This identifier indicates the functions of the processor. These bits indicate whether or not functions defined per bit are implemented (1: implemented, 0: not implemented). Bit 23 to 11 : Reserved Bit 10 : Double-precision floating-point operation function Bit 9 : Single-precision floating-point operation function Bit 8 : Memory protection unit (MPU) function NOTE	R	*1
		If a double-precision floating-point operation function is implemented (when bit 10 is 1), a single-precision floating-point operation function is also always implemented (bit 9 is 1).		
7 to 0	_	Version Identifier This identifier indicates the version of the processor.	R	*1

Note 1. For details, see the hardware manual of the product used.



(21) SCCFG — SYSCALL operation setting

This register is used to set operations related to the SYSCALL instruction. Be sure to set an appropriate value to this register before using the SYSCALL instruction.



Table 3.26	SUCEC	Pogistor	Contonte
Table 3.20	JUCLG	Register	Contents

Bit	Name	Description	R/W	Value after Reset
31 to 8	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
7 to 0	SIZE	These bits specify the maximum number of entries of a table that the SYSCALL instruction references. The maximum number of entries the SYSCALL instruction references is 1 if SIZE is 0, and 256 if SIZE is 255. By setting the maximum number of entries appropriately in accordance with the number of functions branched by the SYSCALL instruction, the memory area can be effectively used. If a vector exceeding the maximum number of entries is specified for the SYSCALL instruction, the first entry is selected. Place an error processing routine at the first entry.	R/W	Undefined

(22) SCBP — SYSCALL base pointer

The SCBP register is used to specify a table address of the SYSCALL instruction and generate a target address. Be sure to set an appropriate value to this register before using the SYSCALL instruction.

Be sure to set a word address to the SCBP register.



Table 3.27 SCBP Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 2	SCBP31 to SCBP2	These bits indicate the base pointer address of the SYSCALL instruction. These bits indicate the start address of the table used by the SYSCALL instruction.	R/W	Undefined
1, 0	SCBP1, SCBP0	These bits indicate the base pointer address of the SYSCALL instruction. These bits indicate the start address of the table used by the SYSCALL instruction. Always set these bits to 0.	R	0



(23) MCFG0 — Machine configuration

This register indicates the CPU configuration.



Table 3.28 MCFG0 Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 24	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
23 to 16	SPID	These bits indicate the system protection number. The SPID bit width depends on the product and the value that can be written might therefore be restricted. For details, see the hardware manual of the product used.	R/W	*1
15 to 3	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
2	—	(Reserved for future expansion. Be sure to set to 1.)	R	1
1, 0	_	(Reserved for future expansion. Be sure to set to 0.)	R	0

Note 1. For details, see the hardware manual of the product used.

(24) MCTL — Control of the CPU

This register is used to control the CPU.



Table 3.29	MCTL	Register	Contents
------------	------	----------	----------

Bit	Name	Description	R/W	Value after Reset
31	_	(Reserved for future expansion. Be sure to set to 1.)	R	1
30 to 1	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	UIC	This bit is used to control the interrupt enable/disable operation in user mode. When this bit is set to 1, executing the EI/DI instruction in user mode become possible.	R/W	0



3.3 Interrupt Function Registers

3.3.1 Interrupt Function System Registers

Interrupt function system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID.

Register No. (regID, seIID)	Symbol	Function	Access Permission
SR10, 2	ISPR	Priority of interrupt being serviced	SV
SR11, 2	PMR	Interrupt priority masking	SV
SR12, 2	ICSR	Interrupt control status	SV
SR13, 2	INTCFG	Interrupt function setting	SV

Table 3.30 Interrupt Function System Registers



(1) ISPR — Priority of interrupt being serviced

This register holds the priority of the EIINTn interrupt being serviced. This priority value is then used to perform priority ceiling processing when multiple interrupts are generated.



Bit	Name	Description		R/W	Value after Reset
31 to 16	—	(Reserved for future exp	pansion. Be sure to set to 0.)	R	0
15 to 0	ISP15 to ISP0	priority* ¹ that correspon- 0 : An interrupt reques relevant bit position 1 : An interrupt reques relevant position is	acknowledgment status of an EIINT <i>n</i> interrupt with a ds to the relevant bit position. It for an interrupt whose priority corresponds to the a has not been acknowledged. It for an interrupt whose priority corresponds to the being serviced by the CPU core.	R ^{*3}	0
		Bit	pond to the following priority levels: Priority		
		0	Priority 0 (highest)		
			Priority 1		
		14	Priority 14		
		15	Priority 15		
		to the acknowledged into 0 when the EIRET instru- among the ISP15-0 bits 0* ² . While a bit in this register (EIINT <i>n</i>) are masked. P when the system is deter meaning that exceptions For details, see Section Masking . When performing softwar	est (EIINT <i>n</i>) is acknowledged, the bit corresponding errupt request is automatically set to 1. If PSW.EP is action is executed, the bit with the highest priority that are set (0 is the highest priority) is cleared to er is set to 1, same or lower priority interrupts riority level judgment is therefore not performed ermining whether to acknowledge an exception, s will not be acknowledged. a 4.1.5, Interrupt Exception Priority and Priority are-based priority control using the PMR register, be er by using the INTCFG.ISPC bit.		
	Note 1. Note 2.	Interrupt acknowledgment by setting (1) the INTCFG cases, the INTCFG.ISPC		ion is exec g of values	s, so in normal
	Note 3.	This is R or R/W, dependin as a read-only (R) register	ng on the setting of the INTCFG.ISPC bit. It is recom r.	mended to	o use this regist

Table 3.3	1 ISPR	Register	Contents
-----------	--------	----------	----------



(2) PMR — Interrupt priority masking

This register is used to mask the specified interrupt priority.



Table 3.32 PMR Register Contents

Bit	Name	Descriptior	ı		R/W	Value after Reset
31 to 16	_	(Reserved f	or future expa	ansion. Be sure to set to 0.)	R	0
15 to 0	PM15 to PM0	the relevant 0 : Servici relevan 1 : Servici relevan	bit position. ng of an intern t bit position i ng of an intern t bit position i	rupt request with a priority that corresponds to the	R/W	0
			Bit	Priority		
			0	Priority 0 (highest)		
			1	Priority 1		
			14	Priority 14		
			15	Priority 15 and priority 16 (lowest)		

performed when the system is determining whether to acknowledge an exception, meaning that exceptions will not be acknowledged* 1 .

Note 1. Specify the masks by setting the bits to 1 in order from the lowest-priority bit. For example, $FF00_H$ can be set, but $F0F0_H$ or $00FF_H$ cannot.

(3) ICSR — Interrupt control status

This register indicates the interrupt control status in the CPU.



Table 3.33	ICSR Register Contents	\$
------------	------------------------	----

Bit	Name	Description	R/W	Value after Reset
31 to 1	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	PMEI	This bit indicates that an interrupt (EIINT <i>n</i>) with the priority level masked by the PMR register exists.	R	0

(4) INTCFG — Interrupt function setting

This register is used to specify settings related to the CPU's internal interrupt function.



 Table 3.34
 INTCFG Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	ISPC	 This bit changes how the ISPR register is written. 0: The ISPR register is automatically updated. Updates triggered by the program (via execution of LDSR instruction) are ignored. 1: The ISPR register is not automatically updated. Updates triggered by the program (via execution of LDSR instruction) are performed. If this bit is cleared to 0, the bits of the ISPR register are automatically set to 1 when an interrupt (EIINT<i>n</i>) is acknowledged, and cleared to 0 when the EIRET instruction is executed. In this case, the bits are not updated by an LDSR instruction executed by the program. If this bit is set to 1, the bits of the ISPR register are not updated by the acknowledgement of an interrupt (EIINT<i>n</i>) or by execution of the EIRET instruction. In this case, the bits can be updated by an LDSR instruction executed by the program. In normal cases, the ISPC bit should be cleared. When performing softwarebased priority control, however, set this bit (1) and perform priority control by using the PMR register. 	R/W	0



3.4 FPU Function Registers

3.4.1 Floating-Point Registers

The FPU uses the CPU general-purpose registers (r0 to r31). There are no register files used only for floating-point operations.

- Single-precision floating-point instruction: Thirty-two 32-bit registers can be specified. These general-purpose registers correspond to r0 to r31.
- Double-precision floating-point instruction:

Sixteen 64-bit registers can be specified. Paired general-purpose registers are used as register pairs ($\{r1, r0\}, \{r3, r2\} \dots \{r31, r30\}$). Each register pair is specified in the instruction format with an even numbered register. Because r0 is a zero register (always holds 0), in principle $\{r1, r0\}$ cannot be used by a double-precision floating-point instruction.

3.4.2 Floating-Point Function System Registers

The FPU can use the following system registers to control floating-point operations. Floating-point function system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID.

- FPSR : This register is used to control and monitor exceptions. It also holds the result of compare operations, and sets the FPU operation mode. Its bits are used to set condition code, subnormal number flush enable, rounding mode control, cause, exception enable, and preservation.
- FPEPC : This register stores the program counter value for the instruction where a floating-point operation exception has occurred.
- FPST : This register reflects the contents of the FPSR register bits related to the operation status.
- FPCC : This register reflects the contents of the FPSR.CC(7:0) bits.
- FPCFG : This register reflects the contents of the FPSR register bits related to the operation settings.

Register No. (regID, seIID)	Symbol	Function	Access Permission
SR6, 0	FPSR	Floating-point operation configuration/status	CU0 and SV
SR7, 0	FPEPC	Floating-point operation exception program counter	CU0 and SV
SR8, 0	FPST	Floating point operation status	CU0
SR9, 0	FPCC	Floating-point operation comparison result	CU0
SR10, 0	FPCFG	Floating-point operation configuration	CU0

Table 3.35 FPU System Registers



(1) FPSR — Floating-point configuration/status

This register indicates the execution status of floating-point operations and any exceptions that occur.

For details about exception, see Section 6.1.5, Floating-Point Operation Exceptions.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
FPSR	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	FN	IF	1	0	R	Μ	FS	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
			Cause b	bits (XC	5)			Enab	le bits	(XE)		F	Preserv	ation b	oits (XP)	Value after reset
	Е	V	Z	0	U	I	V	Z	0	U	I	V	Z	0	U	I	*1

Table 3.36 FPSR Register Contents (1/2)

Bit	Name	Descri	ption			R/W	Value after Reset
31 to 24	CC(7:0)	compar except 0: Cor	ison in the cor npariso	structions. The		R/W	Undefined
23	FN	roundin	ig mod r is flus	e is RN and the hed to the nea	rest mode. When the FN bit is set to 1, if the e operation result is a subnormal number, the rest number. For details, see Section 6.1.9 ,	R/W	0
22	IF	operan	ds. For		licates information about the flushing of input lushing subnormal numbers, see Section 6.1.8, pers .	R/W	0
21		(Reserv	ved for	future expansi	on. Be sure to set to 1.)	R	0
20	_	(Reserv	ed for	future expansi	on. Be sure to set to 0.)	R	0
19, 18	RM			•	e control bits. The RM bits define the rounding all floating-point instructions.	R/W	00
		RN	I Bits	_			
		19	18	Mnemonic	Description		
		0	0	RN	Rounds the result to the nearest representable value. If the value is exactly in-between the two nearest representable values, the result is rounded toward the value whose least significant bit is 0.		
		0	0	RN	Rounds the result to the nearest representable value. If the value is exactly in-between the two nearest representable values, the result is rounded toward the value whose least significant		
			-		Rounds the result to the nearest representable value. If the value is exactly in-between the two nearest representable values, the result is rounded toward the value whose least significant bit is 0. Rounds the result toward 0. The result is the nearest to the value that does not exceed the		



Name	Description					R/W	Value after Reset
FS	be flushed. If the FS bit is a subnormal numbers are flu operation exception (E). An flushed to 0 with the same	R/W	1				
	Operation result that	Round	ling mode a	fter flushing			
	is a subnormal number	RN* ¹	RZ	RP	RM		
	Positive	+0	+0	+2 ^{Emin}	+0		
	Negative	-0	-0	-0	-2 ^{Emin}		
	will occur in the	direction	of higher ad		•		
—	(Reserved for future expar	sion. Be s	sure to set to	o 0.)		R	0
XC (E, V, Z, O, U, I)	These are the cause bits. I	For details	s, see 3.4.2	(1) (a), Caus	se bits (XC).* ²	R/W	Undefined
XE (V, Z, O, U, I)	These are the enable bits.	For detail	s, see 3.4.2	(1) (b), Ena	ble bits (XE).	R/W	0
XP (V, Z, O, U, I)	These are the preservation bits (XP) .* ²	n bits. For	details, see	3.4.2 (1) (c)	, Preservation	R/W	Undefined
	•		ovact ovac	ation (I) bit m	ay be set by flect	ina noint	divido instructior
	FS 	FS This bit enables values that be flushed. If the FS bit is a subnormal numbers are flu operation exception (E). An flushed to 0 with the same numbers either become 0 Operation result that is a subnormal number Positive Positive Note 1. If the rounding will occur in the Section 6.1.9, (Reserved for future expanding the cause bits. In the cause bits.	FS This bit enables values that could no be flushed. If the FS bit is set, input or subnormal numbers are flushed with operation exception (E). An input op flushed to 0 with the same sign. Open numbers either become 0 or the mining the same sign. Open numbers either become 0 or the mining is a subnormal number Operation result that is a subnormal number Rown RN*1 Positive + 0 Negative -0 Note 1. If the rounding mode is R will occur in the direction Section 6.1.9, Flush to I	FS This bit enables values that could not be normali be flushed. If the FS bit is set, input operands an subnormal numbers are flushed without causing operation exception (E). An input operand that is flushed to 0 with the same sign. Operation result numbers either become 0 or the minimum . Operation result that is a subnormal number Rounding mode at RN*1 Positive +0 Positive +0 Note 1. If the rounding mode is RN and the F will occur in the direction of higher ac Section 6.1.9, Flush to Nearest. — (Reserved for future expansion. Be sure to set to XC (E, V, Z, O, U, I) XE (V, Z, O, U, I) These are the enable bits. For details, see 3.4.2 XP (V, Z, O, U, I) These are the preservation bits. For details, see XP (V, Z, O, U, I) These are the preservation bits. For details, see XP (V, Z, O, U, I) These are the preservation bits. For details, see	FS This bit enables values that could not be normalized (subnorm be flushed. If the FS bit is set, input operands and operation subnormal numbers are flushed without causing an unimplet operation exception (E). An input operand that is a subnorm flushed to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation result that is a subnormal number #0 operation result that is a subnormal number #0 operation result that is a subnormal number #0 operation result that are surprised to 0 with the same sign. Operation result that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Operation results that are surprised to 0 with the same sign. Note 1. If the rounding mode is RN and the FPSR.FN bit will occur in the direction of higher accuracy. For Section 6.1.9, Flush to Nearest. — (Reserved for future expansion. Be sure to set to 0.) XC (E, V, Z, O,	FS This bit enables values that could not be normalized (subnormal numbers) to be flushed. If the FS bit is set, input operands and operation results that are subnormal numbers are flushed without causing an unimplemented operation exception (E). An input operand that is a subnormal number is flushed to 0 with the same sign. Operation results that are subnormal numbers either become 0 or the minimum . Operation result that	FS This bit enables values that could not be normalized (subnormal numbers) to be flushed. If the FS bit is set, input operands and operation results that are subnormal numbers are flushed without causing an unimplemented operation exception (E). An input operand that is a subnormal number is flushed to 0 with the same sign. Operation results that are subnormal numbers either become 0 or the minimum . RW Operation result that number is flushed to 0 with the same sign. Operation results that are subnormal numbers either become 0 or the minimum . Operation result that number is flushed to 0 with the same sign. Operation results that are subnormal numbers either become 0 or the minimum . Note 1. Note 1. Result that number is flushed to 0 mode and value after flushing is a subnormal number is either become 0 or the minimum . Depration result that number is a subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation results that are subnormal number is flushed to 0 with the same sign. Operation flushed to 0

Table 3.36 FPSR Register Contents (2/2)

(DIVF.S and DIVF.D). For details, see Section 7.4.4, Floating-Point Instruction Set.



(a) Cause bits (XC)

Bits 15 to 10 in the FPSR register are cause bits, which indicate the occurrence and cause of a floatingpoint operation exception. If an exception defined by IEEE754 is generated, when an enable bit is set to 1 corresponding to the exception, a cause bit is set, and the exception then occurs. When two or more exceptions occur during a single instruction, each corresponding bit is set to 1.

If two or more exceptions are detected, as long as the enable bit corresponding to one of the exceptions is set to 1, the exception occurs. In this case, the cause bits of all the detected exceptions, including exceptions whose enable bits are cleared to 0, are set to 1.

The cause bits are rewritten by a floating-point instruction (except the TRFSR instruction) where the floating-point operation exception occurred. The E bit is set to 1 when software emulation is required, otherwise it is cleared to 0. Other bits are set to 1 or cleared to 0 depending on whether or not an IEEE754-defined exception has occurred.

When a floating-point operation exception has occurred, the operation result is not stored, and only the cause bits are affected.

When the cause bits are set to 1 by an LDSR instruction, a floating-point operation exception does not occur.

(b) Enable bits (XE)

Bits 9 to 5 in the FPSR register are the enable bits, which enable floating-point operation exceptions. When an IEEE754-defined exception occurs, a floating-point operation exception occurs if the enable bit corresponding to the exception has been set to 1.

There are no enable bits corresponding to an unimplemented operation exception (E). An unimplemented operation exception (E) always occurs as a floating-point operation exception.

If the corresponding enable bit has not been set to 1, no exception occurs and the default result defined by IEEE754 is stored.

(c) Preservation bits (XP)

Bits 4 to 0 in the FPSR register are preservation bits. These bits store and indicate the detected exception after reset. An exception defined by IEEE754 occurs, and if a floating-point operation exception is not generated, the preservation bit is set to 1, otherwise it does not change. The preservation bits are not cleared to 0 by the floating-point operation. However, these bits can be set and cleared by software when an LDSR instruction is used to write a new value to the FPSR register.

There are no preservation bits corresponding to unimplemented operation exceptions (E). An unimplemented operation exception (E) always occurs as a floating-point operation exception.

NOTE

For details about the exception types and how they relate to particular bits, see **Figure 6.6**, **Cause, Enable, and Preservation Bits of FPSR Register**.



(2) FPEPC — Floating-point exception program counter

When an exception that is enabled by an enable bit occurs, the program counter (PC) of the instruction that caused the exception is stored.



Table 3.37 FPEPC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	FPEPC31 to FPEPC1	These bits store the program counter (PC) of the floating-point instruction that caused the exception when a floating-point operation exception that is enabled by an enable bit occurs.	R/W	Undefined
0	FPEPC0	This bit stores the program counter (PC) of the floating-point instruction that caused the exception when a floating-point operation exception that is enabled by an enable bit occurs. Always set this bit to 0.	R	0

(3) FPST — Floating-point operation status

This register reflects the contents of the FPSR register bits related to the operation status.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
FPST	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	_
	0	0	E	V	Cause b Z	oits (XC O	C) U	I	0	0	IF	V	Preserv Z	vation b O	oits (XP U)	Value after reset Undefined

Bit	Name	Description	R/W	Value after Reset
31 to 14	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
13 to 8	XC (E, V, Z, O, U, I)	These are cause bits. For details, see 3.4.2 (1) (a), Cause bits (XC) . Values written to these bits are reflected in FPSR.XC bits.	R/W	Undefined
7, 6	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
5	IF	This bit accumulates and indicates information about the flushing of input operands. For details about flushing subnormal numbers, see Section 6.1.8 , Flushing Subnormal Numbers . Value written to this bit is reflected in FPSR.IF bit.	R/W	0
4 to 0	XP (V, Z, O, U, I)	These are preservation bits. For details, see 3.4.2 (1) (c) , Preservation bits (XP) . Values written to these bits are reflected in FPSR.XP bits.	R/W	Undefined

Table 3.38 FPST Register Contents

(4) FPCC — Floating-point operation comparison result

This register reflects the contents of the FPSR.CC(7:0) bits.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
FPCC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
l																	J
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	_
	0	0	0	0	0	0	0	0	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	Value after reset Undefined

Bit	Name	Description	R/W	Value after Reset
31 to 8	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
7 to 0	CC (7:0)	These are CC (condition) bits. They store the result of a floating-point comparison instruction. The CC(7:0) bits are not affected by any instructions except the comparison instruction and LDSR instruction. Values written to these bits are reflected in the CC(7:0) bits of FPSR. 0: Comparison result is false 1: Comparison result is true	R/W	Undefined

Table 3.39 FPCC Register Contents



(5) FPCFG — Floating-point operation configuration

This register reflects the contents of the FPSR register bits related to the operation settings.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
FPCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	R	M	0	0	0	V	Enat Z	ole bits O	(XE) U	Ι	Value after reset 0000 0000 _H

Table 3.40 FPCFG Register Contents

Bit	Name	Descr	iption			R/W	Value after Reset
31 to 10	_	(Rese	ved for	future expans	ion. Be sure to set to 0.)	R	0
9, 8	RM	mode	that the	FPU uses for	ontrol bits. The RM bits define the rounding all floating-point instructions. Values written to I bits of FPSR.	R/W	0
		R	M Bits				
		9	8	Mnemonic	Description		
		0	0	RN	Rounds the result to the nearest representable value. If the value is exactly in-between the two representable values, the result is rounded toward the value whose least significant bit is 0.		
		0	1	RZ	Rounds the result toward 0. The result is the nearest to the value that does not exceed the absolute value of the result with infinite accuracy.		
		1	0	RP	Rounds the result toward +∞. The result is nearest to a value greater than the accurate result with infinite accuracy.		
		1	1	RM	Rounds the result toward The result is nearest to a value less than the accurate result with infinite accuracy.		
7 to 5	_	(Rese	ved for	future expans	ion. Be sure to set to 0.)	R	0
4 to 0	XE (V, Z, O, U, I)				or details, see 3.4.2 (1) (b), Enable bits (XE) . are reflected in the FPSR.XE bits.	R/W	0



3.5 MPU Function Registers

3.5.1 MPU Function System Registers

MPU function system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID.

 Table 3.41
 MPU Function System Registers (1/2)

Register No. (regID, seIID)	Symbol	Function	Access Permission
SR0, 5	MPM	Memory protection operation mode setting	SV
SR1, 5	MPRC	MPU region control	SV
SR4, 5	MPBRGN	MPU base region number	SV
SR5, 5	MPTRGN	MPU end region number	SV
SR8, 5	MCA	Memory protection setting check address	SV
SR9, 5	MCS	Memory protection setting check size	SV
SR10, 5	MCC	Memory protection setting check command	SV
SR11, 5	MCR	Memory protection setting check result	SV
SR0, 6	MPLA0	Protection area minimum address	SV
SR1, 6	MPUA0	Protection area maximum address	SV
SR2, 6	MPAT0	Protection area attribute	SV
SR4, 6	MPLA1	Protection area minimum address	SV
SR5, 6	MPUA1	Protection area maximum address	SV
SR6, 6	MPAT1	Protection area attribute	SV
SR8, 6	MPLA2	Lower address of the protection area	SV
SR9, 6	MPUA2	Protection area maximum address	SV
SR10, 6	MPAT2	Protection area attribute	SV
SR12, 6	MPLA3	Protection area minimum address	SV
SR13, 6	MPUA3	Protection area maximum address	SV
SR14, 6	MPAT3	Protection area attribute	SV
SR16, 6	MPLA4	Protection area minimum address	SV
SR17, 6	MPUA4	Protection area maximum address	SV
SR18, 6	MPAT4	Protection area attribute	SV
SR20, 6	MPLA5	Protection area minimum address	SV
SR21, 6	MPUA5	Protection area maximum address	SV
SR22, 6	MPAT5	Protection area attribute	SV
SR24, 6	MPLA6	Protection area minimum address	SV
SR25, 6	MPUA6	Protection area maximum address	SV
SR26, 6	MPAT6	Protection area attribute	SV
SR28, 6	MPLA7	Protection area minimum address	SV
SR29, 6	MPUA7	Protection area maximum address	SV
SR30, 6	MPAT7	Protection area attribute	SV
SR0, 7	MPLA8	Protection area minimum address	SV
SR1, 7	MPUA8	Protection area maximum address	SV
SR2, 7	MPAT8	Protection area attribute	SV
SR4, 7	MPLA9	Protection area minimum address	SV
SR5, 7	MPUA9	Protection area maximum address	SV



Register No.				
(regID, seIID)	Symbol	Function	Access Permission	
SR6, 7	MPAT9	Protection area attribute	SV	
SR8, 7	MPLA10	Protection area minimum address	SV	
SR9, 7	MPUA10	Protection area maximum address	SV	
SR10, 7	MPAT10	Protection area attribute	SV	
SR12, 7	MPLA11	Protection area minimum address	SV	
SR13, 7	MPUA11	Protection area maximum address	SV	
SR14, 7	MPAT11	Protection area attribute	SV	
SR16, 7	MPLA12	Protection area minimum address	SV	
SR17, 7	MPUA12	Protection area maximum address	SV	
SR18, 7	MPAT12	Protection area attribute	SV	
SR20, 7	MPLA13	Protection area minimum address	SV	
SR21, 7	MPUA13	Protection area maximum address	SV	
SR22, 7	MPAT13	Protection area attribute	SV	
SR24, 7	MPLA14	Protection area minimum address	SV	
SR25, 7	MPUA14	Protection area maximum address	SV	
SR26, 7	MPAT14	Protection area attribute	SV	
SR28, 7	MPLA15	Protection area minimum address	SV	
SR29, 7	MPUA15	Protection area maximum address	SV	
SR30, 7	MPAT15	Protection area attribute	SV	

 Table 3.41
 MPU Function System Registers (2/2)

Note: The number of incorporated MPLAn, MPUAn, and MPATn (n = 0 to 15) registers depends on the hardware specifications. For details, see the hardware manual of the product used.



(1) MPM — Memory protection operation mode

The memory protection mode register is used to define the basic operating state of the memory protection function.



Bit	Name	Description	R/W	Value after Reset
31 to 11	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
10	DX	 This bit specifies the default operation when an instruction is executed at an address that does not exist in a protection area. "0" is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0 : Disable executing an instruction at an address that does not exist in a protection area. 1 : Enable executing an instruction at an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see Section 5.1.3, Caution Points for Protection Area Setup. 	R	0
9	DW	 This bit specifies the default operation when writing to an address that does not exist in a protection area. "0" is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0 : Disable writing to an address that does not exist in a protection area. 1 : Enable writing to an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see Section 5.1.3, Caution Points for Protection Area Setup. 	R	0
8	DR	This bit specifies the default operation when reading from an address that does not exist in a protection area. "0" is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0 : Disable reading from an address that does not exist in a protection area. 1 : Enable reading from an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see Section 5.1.3, Caution Points for Protection Area Setup .	R	0
7 to 2	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
1	SVP	 In SV mode (when PSW.UM = 0), this bit is used to specify whether to restrict access according to the SX, SW, and SR bits of the MPAT register for each protection area.*¹ 0 : As usual, implicitly enable all access in SV mode. 1 : Restrict access according to the SX, SW, and SR bits even in SV mode. 	R/W	0
0	MPE	This bit is used to specify whether to enable or disable MPU function. 0 : Disable 1 : Enable	R/W	0
	Note 1.	When the SVP bit is set to 1, access is restricted according to the setting of eac SV mode. Therefore, specify protection areas before setting the SVP bit to prev program itself from being restricted.	•	
	Note 2.	If access is restricted in SV mode, execution of MDP exceptions or the MIP exc not be possible depending on the settings. Be careful to specify settings so that necessary for the exception handler and exception handling is permitted.	•	0 0

Table 3.42 MPM Register Contents



(2) MPRC — MPU region control

Bits used to perform special memory protection function operations are located in this register.



Bit	Name	Description	R/W	Value after Reset
31 to 16	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
15 to 0	E15 to E0	These are the enable bits for each protection area. Bit En is a copy of bit MPAT <i>n</i> .E (where n = 15 to 0). For the number of protection areas, see the hardware manual of the product used.	R/W	0

(3) MPBRGN — MPU base region number

This register indicates the minimum usable MPU area number.



Table 3.44 MPBRGN Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 5	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
4 to 0	MPBRGN	These bits indicate the smallest number of an MPU area. These bits always indicate 0.	R	0

(4) MPTRGN — MPU end region number

This register indicates the maximum usable MPU area number + 1.



ble	3.45	MPTRGN	Register	Contents
-----	------	---------------	----------	----------

Bit	Name	Description	R/W	Value after Reset
31 to 5	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
4 to 0	MPTRGN	These bits indicate the largest number of an MPU area + 1. These bits indicate the maximum number of MPU areas incorporated into the hardware.	R	*1

Note 1. For details, see the hardware manual of the product used.

Та
(5) MCA — Memory protection setting check address

This register is used to specify the base address of the area for which a memory protection setting check is to be performed.



Table 3.46 MCA Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MCA31 to MCA0	These bits are used to specify the starting address of the memory area which subjects to a memory protection setting check in bytes.	R/W	Undefined

(6) MCS — Memory protection setting check size

This register is used to specify the size of the area for which a memory protection setting check is to be performed.



Table 3.47 MCS Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MCS31 to MCS0	These bits are used to specify the size of the memory area which subjects to a memory protection setting check and the size of the target area in bytes. Because the specified size is assumed to represent an unsigned integer, it is not possible to check an area in the direction in which the address value decreases relative to the MCA register value. Do not specify 0000 $0000_{\rm H}$ for the MCS register.	R/W	Undefined

(7) MCC — Memory protection setting check command

This command register is used to start a memory protection setting check.



Table 3.48 MCC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MCC31 to MCC0	When any value is written to the MCC register, a memory protection setting check starts. By setting up the MCA / MCS register and then writing to the MCC register, results are stored in MCR. Because the check is started by any written value, a check can be started by using r0 as the source register without using any unnecessary registers. Note that, for the check, the results are applied according to each area setting regardless of the state of the PSW.UM bit. When the MCC register is read, value 0000 0000 _H is always returned.	R/W	0



(8) MCR — Memory protection setting check result

This register is used to store the results of a memory protection setting check.

Be sure to clear bits 31 to 9, 7, and 6.

CAUTIONS

- 1. If the specified area to be checked crosses $0000\ 0000_{H}$ or 7FFF FFFF_H, it is judged as an area setting error, and the MCR.OV bit is set to 1. This means that the MCR.OV bit must be checked to access the check results. Do not use the check result until it is confirmed that the result is not invalid (OV = 0).
- 2. When the default set (MPM.DX, DW, DR) is set to 1, it disables sometimes to get the correct result. If enabling the specified default operation, do not use the memory protection setting check function.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
MCR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	ov	0	0	SXE	SWE	SRE	UXE	UWE	URE	Value after reset Undefined

Bit	Name	Description	R/W	Value after Reset
31 to 9	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
8	OV	If the specified area includes $0000\ 0000_H$ or 7FFF FFFF _H , 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
7,6	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
5	SXE	If the specified area is contained within one protection area and execution is permitted for that area in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
4	SWE	If the specified area is contained within one protection area and writing to that area is permitted in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
3	SRE	If the specified area is contained within one protection area and reading from that area is permitted in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.		Undefined
2	UXE	If the specified area is contained within one protection area and execution is permitted for that area in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.		Undefined
1	UWE	If the specified area is contained within one protection area and writing to that area is permitted in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
0	URE	If the specified area is contained within one protection area and reading from that area is permitted in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined

Table 3.49 MCR Register Contents



(9) MPLAn — Protection area minimum address

These registers indicate the minimum address of area n (where n = 0 to 15). The number of protection area n depends on the hardware specifications. For details, see the hardware manual of the product used.



Table 3.	50 MI	Pl Δn F	?eaister (Contents
Table J.	30 1911		vegister v	Contents

Bit	Name	Description	R/W	Value after Reset
31 to 2	MPLA31 to MPLA2	These bits indicate the minimum address of area n. The MPLAn.MPLA1-0 bits are used implicitly set to 0.	R/W	Undefined
1, 0	—	(Reserved for future expansion. Be sure to set to 0.)	R	0

(10) MPUAn — Protection area maximum address

These registers indicate the maximum address of area n (where n = 0 to 15). The number of protection area n depends on the hardware specifications. For details, see the hardware manual of the product used.



Table 3.51	MPII An	Register	Contents
		Negister	Contents

Bit	Name	Description	R/W	Value after Reset
31 to 2	MPUA31 to MPUA2	These bits indicate the maximum address of area n. The MPUAn.MPUA1-0 bits are used implicitly set to 1.	R/W	Undefined
1, 0	_	(Reserved for future expansion. Be sure to set to 0.)	R	0



These registers indicate the attributes of area n (where n = 0 to 15). The number of protection area n depends on the hardware specifications. For details, see the hardware manual of the product used.



Bit	Name	Description	R/W	Value after Reset
31 to 26		(Reserved for future expansion. Be sure to set to 0.)	R	0
25 to 16	ASID	These bits indicate the ASID value to be used as the area match condition.	R/W	Undefined
15 to 8	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
7	E	This bit indicates whether area n is enabled or disabled. 0: Area n is disabled. 1: Area n is enabled.	R/W	0
6	G	0: ASID match is used as the condition. 1: ASID match is not used as the condition. If this bit is 0, MPATn.ASID = ASID.ASID is used as the area match condition. If this bit is 1, the values of MPATn.ASID and ASID.ASID are not used as the area match condition.	R/W	Undefined
5	SX	This bit indicates the execution privilege for the supervisor mode. *1 0: Execution is disabled. 1: Execution is enabled.	R/W	Undefined
4	SW	This bit indicates the write permission for the supervisor mode. ^{*1} 0: Writing is disabled. 1: Writing is enabled.	R/W	Undefined
3	SR	This bit indicates the read permission for the supervisor mode. ^{*1} 0: Reading is disabled. 1: Reading is enabled.	R/W	Undefined
2	UX	This bit indicates the execution privilege for the user mode. 0: Execution is disabled. 1: Execution is enabled.	R/W	Undefined
1	UW	This bit indicates the write permission for the user mode. 0: Writing is disabled. 1: Writing is enabled.	R/W	Undefined
0	UR	This bit indicates the read permission for the user mode. 0: Reading is disabled. 1: Reading is enabled	R/W	Undefined

Table 3.52	MPAT n	Register	Contents
------------	---------------	----------	----------

Note 1. If access is restricted in SV mode, execution of MDP exceptions or the MIP exception handling itself might not be possible depending on the settings. Be careful to specify settings so that access to the memory area necessary for the exception handler and exception handling is permitted.



3.6 Cache Operation Function Registers

3.6.1 Cache Control Function System Registers

Cache control function system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID.

Register No. **Access Permission** (regID, selID) Symbol Function SR16, 4 ICTAGL Instruction cache tag Lo access SV SR17, 4 ICTAGH sv Instruction cache tag Hi access SR18, 4 ICDATL sv Instruction cache data Lo access SR19, 4 ICDATH Instruction cache data Hi access SV SR24, 4 ICCTRL sv Instruction cache control SR26, 4 ICCFG sv Instruction cache configuration SR28, 4 **ICERR** Instruction cache error SV

Table 3.53 Cache Control System Registers



(1) ICTAGL — Instruction cache tag Lo access

This register is used by the CIST/CILD instruction in relation to the instruction cache. During execution of CIST, values that are stored to the tag RAM for the instruction cache are stored. During execution of CILD, values read from the tag RAM for the instruction cache are stored.



Bit	Name	Description	R/W	Value after Reset
31 to 11	LPN	These bits store physical page number bits 24 to 11. Be sure to set bits 31 to 25, and 0.	R/W	Undefined
10 to 6		(Reserved for future expansion. Be sure to set to 0.)	R	0
5, 4	LRU	These bits indicate LRU information of specified cache line. LRU information cannot be freely changed to any value by the CIST instruction.	R/W	Undefined
3	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
2	L	This bit stores the lock information.	R/W	Undefined
1		(Reserved for future expansion. Be sure to set to 0.)	R	0
0	V	This bit stores valid/invalid information of specified cache line.	R/W	Undefined

Table 3.54 ICTAGL Register Contents



(2) ICTAGH — Instruction cache tag Hi access

This register is used by the CIST/CILD instruction in relation to the instruction cache. During execution of CIST, values that are stored to the tag RAM for the instruction cache are stored. During execution of CILD, values read from the tag RAM for the instruction cache are stored.



Bit	Name	Description	R/W	Value after Reset
31	WD When this bit is set to 1 during CIST execution, data RAM of cache is updated.		R/W	Undefined
30	PD	When this bit is set to 1 during CIST execution, values in the DATAECC field are overwritten to ECC for data RAM. When this value is 0, ECC is generated automatically from the write data.	R/W	Undefined
29	WT	When this bit is set to 1 during CIST execution, tag RAM of cache is updated.	R/W	Undefined
28	PT	When this bit is set to 1 during CIST execution, values in the TAGECC field are overwritten to ECC for tag RAM. When this value is 0, ECC is generated automatically from the write data.	R/W	Undefined
27 to 24	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
23 to 16	DATAECC	These bits store ECC for data RAM.	R/W	Undefined
15 to 8	TAGECC	These bits store ECC for tag RAM. Write 0 to bits 15 and 14.	R/W	Undefined
7	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
6	_	(Reserved for future expansion. Be sure to set to 0.)	R	Undefined
5 to 2	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
1, 0	_	(Reserved for future expansion. Be sure to set to 0.)	R	Undefined

Table 3.55 ICTAGH Register Contents

(3) ICDATL — Instruction cache data Lo access

This register is used by the CIST/CILD instruction in relation to the instruction cache. During execution of CIST, values that are stored to the data RAM for the instruction cache are stored. During execution of CILD, values read from the data RAM for the instruction cache are stored.



Table 3.56	ICDATL	Register	Contents
------------	---------------	----------	----------

Bit	Name	Description	R/W	Value after Reset
31 to 0	DATAL	Bits 31 to 0, 95 to 64, 159 to 128, or 223 to 192 are stored among the instruction data of a block in the specified cache line. The stored bits are specified by the offset of index. Offset of index = 00000: Bits 31 to 0 Offset of index = 01000: Bits 95 to 64 Offset of index = 10000: Bits 159 to 128 Offset of index = 11000: Bits 223 to 192	R/W	Undefined



(4) ICDATH — Instruction cache data Hi access

This register is used by the CIST/CILD instruction in relation to the instruction cache. During execution of CIST, values that are stored to the data RAM for the instruction cache are stored. During execution of CILD, values read from the data RAM for the instruction cache are stored.



Table 2.57		Decleter	Contonto
Table 3.57	ICDAIN	Register	Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	DATAH	Bits 63 to 32, 127 to 96, 191 to 160, or 255 to 224 are stored among the instruction data of a block in the specified cache line. The stored bits are specified by the offset of index. Offset of index = 00000: Bits 63 to 32 Offset of index = 01000: Bits 127 to 96 Offset of index = 10000: Bits 191 to 160 Offset of index = 11000: Bits 255 to 224	R/W	Undefined

(5) ICCTRL — Instruction cache control

This register is used to control the instruction cache.



Bit	Name	Description	R/W	Value after Reset
31 to 17	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
16	_	(Reserved for future expansion. Be sure to set to 1.)	R	1
15 to 9	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
8	ICHCLR	When this bit is set to 1, the entire instruction cache is cleared. After this bit is set to 1, it is read as 1 until clearing is completed. The bit is cleared to 0 once clearing is completed.	R/W	0
7 to 3	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
2	ICHEIV	When this bit is set to 1, the instruction cache is automatically set as invalid (the ICHEN bit is cleared to 0) whenever a cache error occurs.	R/W	0
1	ICHEMK	When this bit is set to 1, it masks notification of cache error exceptions for	R/W	1

 Table 3.58
 ICCTRL Register Contents

		(
2	ICHEIV	When this bit is set to 1, the instruction cache is automatically set as invalid (the ICHEN bit is cleared to 0) whenever a cache error occurs.	R/W	0
1	ICHEMK	When this bit is set to 1, it masks notification of cache error exceptions for the CPU after a cache error has occurred.	R/W	1
0	ICHEN	This bit indicates valid/invalid status of instruction cache. 0: Instruction cache is invalid 1: Instruction cache is valid This bit is read as the previous value until the setting is actually reflected in the instruction cache.	R/W	1

(6) ICCFG — Instruction cache configuration

This register indicates the instruction cache configuration.



 Table 3.59
 ICCFG Register Contents

	R/W	Value after Reset
re expansion. Be sure to set to 0.)	R	0
e the size (in Kbytes) of the instruction cache. struction cache rte rtes rtes	R	*1
tes bytes bytes e: Setting prohibited		
e the number of lines for each way in the instruction cache. tion cache e: Setting prohibited	R	*1
e the number of ways in the instruction cache. tion cache e: Setting prohibited	R	*1
~	ve: Setting prohibited	

Note 1. The value after reset depends on the hardware specifications. For details, see the hardware manual of the product used.



(7) ICERR — Instruction cache error

This register is used to store cache error information for the instruction cache.

After the ICHERR bit is set to 1, any subsequent cache error information that is generated is not stored until this setting is explicitly cleared to 0. However, error status bits (ESMH, ESPBSE, ESTE1, ESTE2, ESDC, ESDE) are accumulated. In addition, the ICERR register is not updated while cache operation by the CILD instruction is executed.



Bit	Name	Description	R/W	Value after Reset
instru writir read		This bit is set to indicate that the destination way specified for a CISTI instruction was in error. Although the entry information is overwritten so that writing is completed, the V bit will be cleared the next time the cache line is read (i.e. reading will be judged to have missed the cache). However, setting of this bit is not accompanied by an exception for the CPU.		Undefined
30	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
29	ESMH	Error status: Multi hit	R/W	Undefined
28	ESPBSE	Error status: WAY error	R/W	Undefined
27	ESTE1	Error status: Tag RAM 1-bit error	R/W	Undefined
26	ESTE2	Error status: Tag RAM 2-bit error	R/W	Undefined
25	ESDC	Error status: Data RAM 1-bit correction	R/W	Undefined
24	ESDE	Error status: Data RAM 2-bit error	R/W	Undefined
23, 22	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
21	ERMMH	Error exception notification mask : Multi bit	R/W	0
20	ERMPBSE	Error exception notification mask : WAY error	R/W	0
19	ERMTE1	Error exception notification mask : Tag RAM 1-bit error	R/W	0
18	ERMTE2	Error exception notification mask : Tag RAM 2-bit error	R/W	0
17	ERMDC	Error exception notification mask : Data RAM 1-bit correction	R/W	0
16	ERMDE	Error exception notification mask : Data RAM 2-bit error	R/W	0
15	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
14, 13	ICHEWY	These bits retain the way number where a cache error occurred.	R/W	Undefined
12 to 5	ICHEIX	These bits retain the cache index where a cache error occurred.	R/W	Undefined
4	ICHERQ	Q When this bit is set to 1, this bit indicates that cache error exception notification is in progress. However, if cache error exception notification has been masked, the CPU is not notified even when 1 has been set to this bit.		0
3	ICHED	This bit indicates that an error has occurred in data RAM.	R/W	0
2	ICHET	This bit indicates that an error has occurred in tag RAM.	R/W	0
1	_	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	ICHERR	This bit is set to 1 when a cache error has occurred.	R/W	0

Table 3.60 ICERR Register Contents

3.7 Data Buffer Operation Registers

3.7.1 Data Buffer Control System Registers

Data buffer control system registers are read from or written to by using the LDSR and STSR instructions and specifying the system register number, which is made up of a register number and selection ID. For data buffer functions, see the hardware manual of the product used.

Table 3.61 List of Data Buffer Operation Registers

Register No. (regID, seIID)	Symbol	Function	Access Permission
SR 24, 13	CDBCR	Data buffer control register	SV

(1) CDBCR — Data buffer control register

This is the register for controlling data buffer.



Bit	Name	Description	R/W	Value after Reset
31 to 2	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
1	CDBCLR	When this bit is set to 1, data buffer is all cleared. This bit is always read as 0.	W	0
0	CDBEN	This bit specifies enables or disables of the data buffer. 0: Data buffer is disabled. 1: Data buffer is enabled.	R/W	1



Section 4 EXCEPTIONS AND INTERRUPTS

An exception is an unusual event that forces a branch operation from the current program to another program, due to certain causes.

A program at the branch destination of each exception is called an "exception handler".

CAUTION

This CPU handles interrupts as types of exceptions.

4.1 Outline of Exceptions

This section describes the elements that assign properties to exceptions, and shows how exceptions work.

4.1.1 Exception Cause List



	Table 4.1	Exception Cause List		(1/2)											
							Priority (Priority Order * ²	Acknowl Conditio	Acknowledgment Condition (PSW)	Update (PSW)	PSW)			
Exception	Name	Source	Type *1	Saved Resource	Return/ Restoration	Exception Cause Code * ⁵	Priority Level	Priority	₽	ЧN	M	₽	đ	Б	EBV
RESET	Reset	Reset input* ³	Terminating	Ι	1	None	-	I	×	×	0	-	0	0	0
FENMI	FENMI interrupt	Interrupt controller ³	Terminating	표	No	ЕОН	ę	-	×	×	0	-		0	S
SYSERR	System error	System error input ^{*3}	Terminating	E	No	10 _H -1F _H *3	e	N	×	×	0	-		-	S
FEINT	FEINT interrupt	Interrupt controller ^{*3}	Terminating	E	Yes	Fо _H	e	ę	×	0	0	-		0	S
EINT0-511	User interrupt	Interrupt controller ^{*3}	Terminating	Ξ	Yes	1000 _H -11FF _H * ⁶	4	*4	0	0	0	-	S	0	S
MIP	Memory protection exception (execution privilege)	Memory protection violation	Resumable	붠	Yes	Н06	10	~	×	×	0			~	S
SYSERR	System error	Error input during instruction fetch *3	Resumable	붠	N	10 _H -1F _H * ³	10	б	×	×	0			-	S
RIE	Reserved instruction exception	Execution of a reserved instruction	Resumable	Ë	Yes	60 _H	10	4	×	×	0			-	S
UCPOP	Coprocessor unusable exception	Execution of a coprocessor instruction/ access permission violation	Resumable	FE	Yes	80 _H -82 _H * ⁹	10	ы	×	×	0	.	~		w
ЫE	Privilege instruction exception	Execution of a privileged instruction/ access permission violation	Resumable	FE	Yes	ЧОН	10	Q	×	×	0	-	.	.	w
MAE	Misalignment exception	Misaligned access occurrence	Resumable	H	Yes	COH	11	*7	×	×	0	-	-	-	s
MDP	Memory protection exception (access privilege)	Memory protection violation	Resumable	H	Yes	91 _H	11	*	×	×	0	-	-	-	s
FPINT	Floating-point operation exception	Execution of an FPU instruction	Resumable	EI	Yes	71 _H	11	L*	×	×	0	1	s	-	s
SYSCALL	System call	Execution of the SYSCALL instruction	Pending	Ξ	Yes	8000 _H -80FF _H	12	8 *	×	×	0	-	S	-	S

RH850G3MH Software

R01US0143EJ0130 Rev.1.30 Dec 22, 2016 RENESAS

		EBV														
			S	S	S						eption.					
		8	-	-	-						ach exc					
		Å	-	S	S						ed for e					
	(PSW)	₽	-		-						de defin			evel.		
	Update (PSW)	MU	0	0	0			r priority			ailed co			priority l		
	dgment (PSW)	đ	Ļ	Ļ	Ļ			i a highe			n the det			e same		
	Acknowledgment Condition (PSW)		×	×	×			alue has			e contair			within th		
		Priority ID	×	×	×			naller v:			ise code			priority		
	Priority Order * ²	~	œ	œ	œ			rity. A sr rder .			tion cau			re is no		
	Priori	Priority Level	12	12	12			ten prior iority O		jā.	e excep			on. The	vely.	
		Exception Cause Code * ⁵	31 _H -3F _H	40 _H -4F _H	50 _H -5F _H			ns is checked by the priority level, and then priority. A smaller value has a higher priority. Acknowledgment Conditions and Priority Order.		depending on the register setting. Exception Priority and Priority Masking.	r 16 bits of the f the function.	annel.		ise they occur due to instruction execution. There is no priority within the same priority level.	:U2), respecti	
			с	4	ъ 2			e priorit t Condi		egister s and Pri	ie highe iption of	o the ch		to instru	U0 to C	
		Return/ Restoration	Yes	Yes	Yes		ions.	necked by th owledgmen	ne product used.	depending on the register setting. Exception Priority and Priority I	e shown. Th in the descr	according to	ns.	y occur due	ermission (C	
2/2)		Saved Resource	Ë	Ξ	Ξ	condition	of Excepti	eptions is ch otion Ackne	of the prod	ary depend	use code ar	ire selected	of instructio	ecause the	ssor use pe	
use List (Type *1	Pending	Pending	Pending	owledgment	4.1.3, Types	ority for exce 4.1.4, Excel	ware manua	o EIINT511 v 4.1.5, Interr	exception ca less otherwis	ls 0 to 511) ¿	ration order	exclusively t	o the coproc	
Exception Cause List (2/2)		Source	Execution of the F FETRAP instruction	Execution of the F TRAP instruction	Execution of the F TRAP instruction	s: Retained, x: Not an acknowledgment condition	For details, see Section 4.1.3, Types of Exceptions.	The acknowledgment priority for exception For details, see Section 4.1.4, Exception	For details, see the hardware manual of the	The priorities of EIINT0 to EIINT511 vary For details, see Section 4.1.5, Interrupt I	The lower 16 bits of the exception cause code are shown. The higher 16 bits of the exception cause code contain the detailed code defined for each exception. These bits are 0000 _H unless otherwise specified in the description of the function.	$1000_{\textrm{H}}$ to $11\textrm{FF}_{\textrm{H}}$ (channels 0 to 511) are selected according to the channel.	This depends on the operation order of instructions.	These exceptions occur exclusively becau	$80_{ m H}$ to $82_{ m H}$ correspond to the coprocessor use permission (CU0 to CU2), respectively.	
N						Retain	For det	The ac For det	For det	The pri For det	The lov These	1000 _H	This de	These	80 _H to	
Table 4.1		Name	FE level trap	El level trap 0	El level trap 1	Note: s:	Note 1.	Note 2.	Note 3.	Note 4.	Note 5.	Note 6.	Note 7.	Note 8.	Note 9.	
		Exception	FETRAP	TRAPO	TRAP1											

RENESAS

Page 86 of 426

4.1.2 Overview of Exception Causes

The following is an overview of the exception causes handled in this CPU.

(1) RESET

These are signals generated when inputting a reset. For details, see Section 8, RESET.

(2) FENMI, FEINT, and EIINT

These are interrupt signals that are input from the interrupt controller to activate a certain program. For details about the interrupt functions, see **Section 3.3, Interrupt Function Registers** and the specifications of the interrupt controller incorporated in your product.

(3) SYSERR

This is a system error exception. This exception occurs when an error defined by the hardware specifications is detected. An error that occurs at an instruction fetch access is reported as a resumable-type SYSERR exception. Other errors are reported as a terminating-type SYSERR exception.

CAUTION

The cause of an SYSERR exception is determined according to the hardware functions. For details, see the hardware manual of the product used.

(4) FPINT

These are exceptions that occur when a floating-point instruction is being executed. For details, see Section 6.1, Floating-Point Operation.

(5) MIP and MDP

These are exceptions that occur when the MPU detects a violation. Detecting an exception is performed when the address at which the instruction will access the memory is calculated. For details, see **Section 5.1, Memory Protection Unit (MPU)**.

(6) RIE

This is a reserved instruction exception. This exception occurs when an attempt is made to execute the opcode of an instruction other than an instruction whose operation is defined. The operation is the same as a RIE instruction whose operation is defined. For details, see **7.1.3**, **Reserved Instructions** in **Section 7**, **INSTRUCTION**.

(7) PIE

This is a privilege instruction exception. This exception occurs when an attempt is made to execute an instruction that does not have the required privilege. For details, see Section 2.1.3, CPU Operating Modes and Privileges, Section 2.2, Instruction Execution, and Section 2.5.3(1) LDSR and STSR.

(8) UCPOP

This is an exception that occurs when an attempt is made to execute a coprocessor instruction when the coprocessor in question is not usable. For details, see **Section 2.4, Coprocessors**.

(9) MAE

This is an exception that occurs when the result of address calculation is a misaligned address. For details, see **Section 2.6.3, Data Alignment**.

(10) TRAP, FETRAP, and SYSCALL

These are exceptions that occur according to the result of instruction execution. For details, see **Section 7, INSTRUCTION**.

4.1.3 Types of Exceptions

This CPU divides exceptions into the following three types according how they are executed.

- Terminating exceptions
- Resumable exceptions
- Pending exceptions

(1) Terminating exceptions

In the case of an exception of this type, the exception is acknowledged by interrupting the current instruction before its operation is executed. These exceptions include interrupts and exceptions that are generated by sources that are unrelated to the program currently running, such as hardware errors.

These interrupts do not occur as a result of executing the current instruction and are not related to the instruction. When an interrupt occurs, the PSW.EP bit is cleared to 0, unlike other exceptions. Consequently, termination of the exception handler routine is reported to the external interrupt controller when the return instruction is executed. Be sure to execute an instruction that returns execution from an interrupt while the PSW.EP bit is cleared to 0.

CAUTION

The PSW.EP bit is cleared to 0 only when an interrupt (INT0 to INT511, FEINT, or FENMI) is acknowledged. It is set to 1 when any other exception occurs.

If an instruction to return execution from the exception handler routine that has been started by generation of an interrupt is executed while the PSW.EP bit is set to 1, the resources on the external interrupt controller might not be released, causing malfunctioning.

The return PC of a terminating exception is the PC of the terminated instruction (current PC).

(2) Resumable exceptions

This is an exception acknowledged during the execution of instruction operation before the execution is finished. The floating-point operation exception is an example of an exception of this type. Generalpurpose registers and system registers are not updated due to the occurrence of an exception of this type. The PC value on return from the exception continues to point to the instruction where the exception occurred, so execution can be restarted from the state of before the exception occurred.

The return PC of a resumable exception is the PC of the instruction which caused the exception (current PC).



(3) Pending exceptions

This is an exception acknowledged after the execution of an instruction finishes as a result of executing the instruction operation. Pending exceptions include software exceptions. Because pending exceptions occur as a result of normal instruction execution, the processing resumes with the instruction following the instruction that caused the pending exceptions when processing control is returned. The original processing can be normally continued after the exception handling.

The return PC of a pending exception is the PC of the next instruction (next PC).

4.1.4 Exception Acknowledgment Conditions and Priority Order

The CPU acknowledges only one exception at specific timing based on the exception acknowledgment conditions and priority order. The exception to be acknowledged is determined based on the exception acknowledgment conditions and priority order, as shown in **Figure 4.1** below.



Figure 4.1 Exception Acknowledgment Conditions and Priority Order

In **Table 4.1**, an exception with "0" in the acknowledgment condition column can be acknowledged when the corresponding bit is "0". For this kind of exception, acknowledgment is held pending when the corresponding bit is "1". When it changes to "0" and the acknowledgment conditions are met, acknowledgment of the exception becomes possible. If no value is specified for a bit, it is not an acknowledgment condition. If multiple bits are specified as conditions, all the conditions must be met simultaneously.

If more than two exceptions satisfy the acknowledgment conditions simultaneously, one exception is selected according to the priority order. The priority order is determined in multiple stages; priority level, and then priority. A smaller number has a higher priority.

When a terminating exception is not acknowledged, it is held pending. If it occurs at the time of a reset, it is not held pending. For details, see **Section 4.2.1**, **Special Operations**.

For details about acknowledgment conditions, priority level, and priority, see **Table 4.1, Exception Cause List**.

4.1.5 Interrupt Exception Priority and Priority Masking

An interrupt (EIINT*n*) can be masked for each exception priority or interrupt priority by setting registers. This function allows the software implementation of an interrupt ceiling with a more flexible software structure and no maintenance.

Figure 4.2 shows an overview of the functions of interrupt exception priority and priority masking.



Figure 4.2 Interrupt Exception Priority and Priority Masking



(1) Interrupt priority

For an interrupt (EIINT*n*), the exception priority can be changed by setting registers.

(2) Interrupt priority mask

EIINT*n* might be masked at different priorities by the ISPR register and PMR register. These registers should be used as follows.

For the ISPR register, the bit corresponding to the priority is set (1) when the hardware acknowledges an interrupt, and interrupts with the same or lower priority are masked. When the EIRET instruction corresponding to the interrupt is executed, the corresponding bit of the ISPR register is cleared (0) to clear the mask.

This automatic interrupt ceiling makes multiplexed interrupt servicing easy without using software control.

The PMR register allows you to mask specific interrupt priorities with software. Use it to raise the level of the interrupt ceiling temporarily in a program. The mask setting specified by the ISPR register and the mask setting of PMR might overlap, and an interrupt is masked if it is masked with one or the other of them. Normally, use the PMR register to raise the ceiling value from the ceiling value of the ISPR register.

The function of the INTCFG register allows you to disable auto update of the ISPR register upon acknowledgment of and return from an interrupt. To perform interrupt ceiling control by using software without using the function of the ISPR register, set (1) the ISPC bit of the INTCFG register, clear the ISPR register, and then control the ceiling value with software by using the PMR register.

Also, when you are using the PMR register, you can check if any interrupt is masked with the PMR register by using the ICSR register.

4.1.6 Return and Restoration

When exception handling has been performed, it might affect the original program that was interrupted by the acknowledged exception. This effect is indicated from two perspectives: "Return" and "Restoration".

- Return : Indicates whether or not the original program can be re-executed from where it was interrupted.
- Restoration : Indicates whether or not the processor statuses (status of processor resources such as general-purpose registers and system registers) can be restored as they were when the original program was interrupted.

An exception that cannot be returned or restored from ("No" in **Table 4.1**) might cause the return PC to be lost, making it impossible to return from the exception to the original processing by using a return instruction. An exception whose trigger cannot be selected is an unreturnable or unrestorable exception.

For an unrestorable exception, it is possible to return to the original program flow. However, because the state before the occurrence of the exception cannot be restored at that point, care must be taken in continuing subsequent program operation.



4.1.7 Context Saving

To save the current program sequence when an exception occurs, appropriately save the following resources according to the function definitions.

- Program counter (PC)
- Program status word (PSW)
- Exception cause code (EIIC, FEIC)
- Work system register (EIWR, FEWR)

The resource to use as the saving destination is determined according to the exception type. Saved resource determination is described below.

(1) Context saving

Exceptions with certain acknowledgment conditions might not be acknowledged at the start of exception handling, based on the pending bits (PSW.ID and NP bits) that are automatically set when another exception is acknowledged.

To enable processing of multiple exceptions of the same level that can be acknowledged again, certain information about the corresponding return registers and exception causes must be saved, such as to a stack. This information that must be saved is called the "context".

In principle, before saving the context, caution is needed to avoid the occurrence of exceptions at the same level.

The work system registers that can be used for work to save the context, and the system registers that must be at least saved to enable multiple exception handling are called basic context registers. These basic context registers are provided for each level.

 Table 4.2
 Basic Context Registers

Exception Level	Basic Context Registers
El level	EIPC, EIPSW, EIIC, EIWR
FE level	FEPC, FEPSW, FEIC, FEWR



4.2 Operation When Acknowledging an Exception

Check whether each exception that is reported during instruction execution is acknowledged according to the priority. The procedure for exception-specific acknowledgment operation is shown below.

- (1) Check whether the acknowledgment conditions are satisfied and whether exceptions are acknowledged according to their priority.
- (2) Calculate the exception handler address according to the current PSW value^{*1}.
- (3) For FE level exceptions, the following processing is performed.
- Saving the PC to FEPC
- Saving the PSW to FEPSW
- Storing the exception cause code in FEIC
- Updating the PSW^{*2}
- Store the exception handler address calculated in (2) in the PC, and then pass control to the exception handler.
- (4) For EI level exceptions, the following processing is performed.
- Saving the PC to EIPC
- Saving the PSW to EIPSW
- Storing the exception cause code in EIIC
- Updating the PSW^{*2}
- Store the exception handler address calculated in (2) in the PC, and then pass control to the exception handler.
- Note 1. For details, see Section 4.4, Exception Handler Address.
- Note 2. For the values to be updated, see Table 4.1, Exception Cause List.

The following figure shows steps (1) to (4).





Figure 4.3 Operation When Acknowledging an Exception



4.2.1 Special Operations

(1) EP bit of PSW register

If an interrupt is acknowledged, the PSW.EP bit is cleared to 0. If an exception other than an interrupt is acknowledged, the PSW.EP bit is set to 1.

Depending on the EP bit setting, the operation changes when the EIRET or FERET instruction is executed. If the EP bit is cleared to 0, the bit with the highest priority (0 is the highest) among the bits set to 1 in ISPR.ISP15 to ISPR.ISP0 is cleared to 0. Also, the end of the exception handling routine is reported to the external interrupt controller. This function is necessary for correctly controlling resources, such as a request flag, on the interrupt controller when an interrupt is acknowledged or when execution returns from the interrupt.

To return from an interrupt, be sure to execute the return instruction with the EP bit cleared to 0.

(2) Coprocessor unusable exception

For coprocessor unusable exceptions, the exception occurrence opcode corresponding to the status of the CU bit of the PSW register differs according to the specifications of each product.

For coprocessor instructions and defined opcodes, if an attempt is made to execute a coprocessor instruction that is not included in the product or for which the operation state prevents use, or an LDSR or STSR instruction attempts to access a coprocessor system register, a coprocessor unusable exception (UCPOP) immediately occurs.

For details, see Section 2.4.3, Coprocessor Unusable Exceptions.

(3) Reserved instruction exception

If an opcode that is reserved for future function extension and for which no instruction is defined is executed, a reserved instruction exception (RIE) occurs.

However, which of the following two types of operations each opcode is to perform might be defined by the hardware specifications.

- Reserved instruction exception occurs.
- Operates as a defined instruction.

An opcode for which a reserved instruction exception occurs is always defined as an RIE instruction.

(4) Reset

Reset is performed in the same way as exception handling, but it is not regarded as EI level exception or FE level exception. The reset operation is the same that of an exception without acknowledgment conditions, but the value of each register is changed to the value after reset. In addition, execution does not return from the reset status.

All exceptions that have occurred at the same time as CPU initialization are canceled and not acknowledged even after CPU initialization.

For details, see Section 8, RESET.



4.3 Return from Exception Handling

To return from exception handling, execute the return instruction (EIRET or FERET) corresponding to the relevant exception level.

When a context has been saved, such as to a stack, the context must be restored before executing the return instruction. When execution is returned from an irrecoverable exception, the status before the exception occurs in the original program cannot be restored. Consequently, the execution result might differ from that when the exception does not occur.

The EIRET instruction is used to return from EI level exception handling and the FERET instruction is used to return from FE level exception handling.

When the EIRET or FERET instruction is executed, the CPU performs the following processing and then passes control to the return PC address.

- When the EIRET instruction is executed, return PC and PSW are loaded from the EIPC and EIPSW registers.
 When the FERET instruction is executed, return PC and PSW are loaded from the FEPC and FEPSW registers.
- (2) Control is passed to the address indicated by the return PC that were loaded.
- (3) When the EIRET instruction is executed while EP = 0 and INTCFG.ISPC = 0, the CPU updates the ISPR register.

When the FERET instruction is executed, the CPU does not update the ISPR register.

The flow for returning from exception handling using the EIRET or FERET instruction is shown below.





Figure 4.4 Return Instruction-Based Exception Return Flow



4.4 Exception Handler Address

For this CPU, the exception handler address used for execution during reset input, exception acknowledgment, or interrupt acknowledgment can be changed according to the settings.

4.4.1 Resets, Exceptions, and Interrupts

The exception handler address for resets and exceptions is determined by using the direct vector method, in which the reference point of the exception handler address can be changed by using the PSW.EBV bit, RBASE register, and EBASE register. For interrupts, the direct vector method and table reference method can be selected for each channel. If the table reference method is selected, execution can branch to the address indicated by the exception handler table allocated in the memory.

(1) Direct vector method

The CPU uses the result of adding the exception cause offset shown in **Table 4.3**, **Selection of Base Register/Offset Address** to the base address indicated by the RBASE or EBASE register as the exception handler address.

Whether to use the RBASE or EBASE register as the base address is selected according to the PSW.EBV bit*¹. If the PSW.EBV bit is set to 1, the EBASE register value is used as the base address. If the bit is cleared to 0, the RBASE register value is used as the base address.

However, reset input and some exceptions^{*2} always refer to the RBASE register.

In addition, user interrupts refer to the RINT bit of the corresponding base register, and reduce the offset address according to the bit status. If the RBASE.RINT bit or EBASE.RINT bit is set to 1, all user interrupts are handled using an offset of $100_{\rm H}$. If the bit is cleared to 0, the offset address is determined according to **Table 4.3, Selection of Base Register/Offset Address**.

- Note 1. Exception acknowledgment itself sometimes updates the status of the PSW.EBV bit. In this case, the base register is selected based on the new bit value. For details, see Section 4.4, Exception Handler Address.
- **Note 2.** The exceptions that always reference RBASE are determined according to the hardware specifications.







The table below shows how base register selection and offset address reduction function for each exception to determine the exception handler address. The PSW bit value determines the exception handler, based on the value after being updated due to the acknowledgment of an exception.

 Table 4.3
 Selection of Base Register/Offset Address

	PSW.EBV = 0	PSW.EBV = 1	RINT = 0	RINT = 1
	Base	e Register	Of	ffset Address
RESET	RBASE	None*1	000 _H	000 _H
SYSERR	_	EBASE	010 _H	010 _H
FETRAP	_		030 _H	030 _H
TRAP0	_		040 _H	040 _H
TRAP1	_		050 _H	050 _H
RIE	_		060 _H	060 _H
FPINT	_		070 _H	070 _H
UCPOP	_		080 _H	080 _H
MIP/MDP	_		090 _H	090 _H
PIE	_		0A0 _H	0A0 _H
Debug* ²	_		0B0 _H	0B0 _H
MAE	_		0C0 _H	0C0 _H
(R.F.U.)	_		0D0 _H	0D0 _H
FENMI	_		0E0 _H	0E0 _H
FEINT	_		0F0 _H	0F0 _H
EIINTn (priority 0)	_		100 _H	100 _H
EIINTn (priority 1)	_		110 _H	
EIINTn (priority 2)	_		120 _H	
EIINTn (priority 3)	_		130 _H	
EIINTn (priority 4)	_		140 _H	
EIINTn (priority 5)	_		150 _H	
EIINTn (priority 6)	_		160 _H	
EIINTn (priority 7)	_		170 _H	
EIINTn (priority 8)	_		180 _H	
EIINTn (priority 9)	_		190 _H	
EIINTn (priority 10)	_		1A0 _H	
EIINTn (priority 11)	_		1B0 _H	
EIINTn (priority 12)	_		1C0 _H	
EIINTn (priority 13)	_		1D0 _H	
EIINTn (priority 14)	_		1E0 _H	
EIINTn (priority 15)	_		1F0 _H	

Note 1. An exception generated to update EBV to 0.

Note 2. The exception for debug function.

Base register selection is used to execute the exception handling for resets and some hardware errors by using programs in a relatively reliable area such as ROM instead of areas that are easily affected by soft errors such as RAM and cache areas. The user interrupt offset address reduction function is used to reduce the memory size required by the exception handler for specific system-internal operating modes. The main purpose of this is to minimize the amount of memory consumed in operating modes that use only the minimum functionality, which are used, for example, during system maintenance and diagnosis.

(2) Table reference method

In the direct vector method, there is one user-interrupt exception handler for each interrupt priority level, and interrupt channels that indicate multiple interrupts with the same priority branch to the same interrupt handler, but some users might want to use code areas that differ from the start time for each interrupt handler.

When using the table reference method, if the table reference method is specified as the interrupt channel vector selection method for the interrupt controller, the method for determining the exception handler address when an interrupt request corresponding to that interrupt channel is acknowledged differs as follows.

- (1) In any of the following cases, the exception handler address is determined by using the direct vector method.
 - When PSW.EBV = 0 and RBASE.RINT = 1
 - When PSW.EBV = 1 and EBASE.RINT = 1
 - When the interrupt channel setting is not the table reference method
- (2) In cases other than (1), calculate the table reference position.Exception handler address read position = INTBP register + channel number × 4 bytes
- (3) Read word data starting at the interrupt handler address read position calculated in (2).
- (4) Use the word data read in (3) as the exception handler address.

CAUTION

For details about the interrupt channel settings, see the hardware manual of the product used.



A table of exception handler address read positions corresponding to interrupt channels and an overview of the placement in memory are shown below.

 Table 4.4
 Exception Handler Address Expansion

Туре	Exception Handler Address Read Position
EIINT interrupt channel 0	INTBP + 0 × 4
EIINT interrupt channel 1	INTBP + 1 × 4
EIINT interrupt channel 510	INTBP + 510 × 4
EIINT interrupt channel 511	INTBP + 511 × 4



Figure 4.6 Overview of Using the Table Reference Method

For details about the exception handler address selection method settings for each interrupt channel, see the hardware manual of the product used.



4.4.2 System Calls

For system call exceptions, the referenced table entry is selected according to the value of the vector specified based on the opcode and the value of the SCCFG.SIZE bit, and the exception handler address is calculated according to the contents of the table entry and the SCBP register value.

As an example, if table size *n* is specified by SCCFG.SIZE, the table entry is selected as shown below. Note that if the vector specified by the SYSCALL instruction (vector 8) is greater than table size *n*, the table entry referenced by vector n + 1 to 255 is table entry 0.

Table 4.5	System Calls		
Vector	Exception Cause Code	Referenced Table Entry	
0	0000 8000 _H	Table entry 0	
1	0000 8001 _H	Table entry 1	
2	0000 8002 _H	Table entry 2	
n – 1	0000 8000 _H + (n – 1) _H	Table entry n – 1	
n	0000 8000 _H + n _H	Table entry n	
n + 1	0000 8000H + (n + 1) _H	Table entry 0	
254	0000 80FE _H	Table entry 0	
255	0000 80FF _H	Table entry 0	

CAUTION

Because table entry 0 is selected even if a vector that exceeds n, which is specified for SCCFG.SIZE, is specified, allocate the error processing routine.



4.4.3 Models for Application

The following describes the relations among the RBASE, EBASE, and PSW.EBV bit, and the models intended for application. Principally, in cases where a reset occurs and there is no main code in the address space, this main code is first expanded into the address space (which is often in DRAM) by bootstrapping to enable execution, or it is used to when inserting an instruction cache into an exception handling routine.

Immediately after a reset, when PSW.EBV = 0, operations use the ROM area where the minimum maintenance code was placed as specified in RBASE. After bootstrapping, and after the required code has been expanded in RAM, the code position in the RAM is set to the EBASE register and the PSB.EBV bit is set to 1^{*1} .

Normally, this is the mode of software operations. As for exceptions or interrupts in the range of normal operations, because they are acknowledged when PSW.EBV = 1, the code operates in the RAM area indicated by EBASE, but in cases where phenomena (such as RAM errors or cache errors) occur that would indicate the RAM code itself has not remained correct, an exception is triggered to clear to 0 the PSB.EBV bit^{*2}. In such cases, there is a possibility that the exception handler itself might not be executed correctly using the code at the position indicated by EBASE, so control is moved to the exception handler in the ROM code indicated by RBASE and the PSW.EBV bit is cleared to 0.

Once the PSW.EBV bit is cleared to 0, even if an ordinary exception were to occur while in this mode, the status of the PSW.EBV bit is handed over, so that a mode enabling correct execution of RAM code is maintained, and operation uses code in the ROM area indicated by RBASE until the PSW.EBV bit is set to 1 by the maintenance code.

- Note 1. Normally, an EIRET or FERET instruction should be used to set the PSW.EBV bit to 1.
- **Note 2.** The hardware specifications determine which exception has which cause, and whether or not an exception is needed to clear PSW.EBV to 0.





Figure 4.7 Example of Model for Application (Operation Flow)



Figure 4.8 Example of Model for Application (Address Map)

Section 5 MEMORY MANAGEMENT

This CPU provides the following functions for managing the memory.

- Memory protection unit (MPU)
- Instruction cache function
- Mutual exclusion function
- Synchronization function

5.1 Memory Protection Unit (MPU)

Memory protection functions are provided in an MPU (memory protection unit) to maintain a smooth system by detecting and preventing unauthorized use of system resources by unreliable programs, runaway events, etc.

5.1.1 Features

(1) Memory access control

Multiple protection areas can be assigned to the address space. Consequently, unauthorized program execution or data manipulation by user programs can be detected and prevented. The upper and lower limit addresses of each area can be specified so that the address space can be used precisely and efficiently.

(2) Access management for each CPU operation mode

In this CPU, several status bits are used to control access to resources, and these bits are used in combination to perform protection that is appropriate, according to each program's level of reliability.

The initial settings are set as appropriate values in the MPM register. Always use the MPE bit to validate the MPU. The SVP bit should be set to 1 only when protection is also being performed by a supervisor such as an OS.



5.1.2 Protection Area Settings

(1) Protection area settings

Set the respective protection areas appropriately. For details about registers, see **Section 3**, **REGISTER SET**.

Some additional description is provided below regarding certain caution points.

(a) E bit

This sets the target protection area setup as enabled or disabled. When disabled, all settings are disabled. Make sure valid setting values have been stored for other protection areas (MPUA, MPLA, and MPAT) at the time when this bit is set to 1.

(b) UX, UR, and UW bits

These bits indicate the access privileges for the target protection area during user mode.

(c) SX, SR, and SW bits

These bits indicate the access privileges for the target protection area during supervisor mode. These bits are valid only when the MPM.SVP bit has been set to 1. If the MPM.SVP bit has been cleared to 0, protection is not performed while in supervisor mode, regardless of the values of the SX, SR, and SW bits, and the entire address space becomes access-enabled.

(d) G bit and ASID field

These are the G (Global) bit and the ASID field for comparison. When the G bit is cleared to 0, the values in the ASID register are compared to those in the MPAT.ASID field, and protection area settings are applied to determine accessibility only when these values match. When the G bit is set to 1, protection area settings are applied regardless of the ASID values.



5.1.3 Caution Points for Protection Area Setup

(1) Crossing protection area boundaries

When the specified protection areas overlap, the access control settings for the overlapping parts differ depending on the MPM.DX, DW, and DR bits. If access to the protection area is disabled by default, access is enabled by priority; if access to the protection area is enabled by default, access is prohibited by priority.

In other words, when access to protection areas is disabled by default and multiple protection areas have been specified, if access is enabled for either of the protection areas, access is judged to be enabled. If access to the protection area is enabled by default and access is prohibited for either of the protection areas, access is judged to be prohibited.

In addition, the bits for MPM.DX, DW, and DR in this CPU are fixed to 0, and default operation is prohibited.

(2) Invalid protection area settings

Protection area settings are invalid in the following case.

• When value set to lower-limit address is larger than value set to upper-limit address

CAUTION

Note, however, that addresses are handled as unsigned integers (0_H to FFFF FFF_H).

(3) Memory access spanning contiguous areas to which access is enabled by the MPU

Access to load values from or store values in areas that may be under protection by the MPU should be handled entirely in single areas. Even if access is enabled in contiguous areas for access control by the MPU, access spanning the access-control areas is prohibited. In the case of this CPU, memory access spanning areas under access control only possible in response to double-word access by the ld.dw or st.dw instruction. The prepare, dispose, pushsp, and popsps instructions are handled as repeated rounds of word access, so access by these instructions that spans areas for access control by the MPU is permitted.

In the case of prefetching, when a whole instruction spans MPU access-control areas and the MPU is enabling access for the area containing the entry point, memory protection allows the access to fetch the instruction.


5.1.4 Access Control

In this CPU, accesses are controlled appropriately according to the settings specified as of the step described in **Section 5.1.3, Caution Points for Protection Area Setup**. In any of the cases listed below, the CPU ensures logical integrity by limiting actual access, detecting violations before instruction execution is completed, and setting up exceptions.

- When about to execute an instruction that includes opcode, at an address outside the executable range
- When about to execute an instruction that reads from an address outside the read-accessible range
- When about to execute an instruction that writes to an address outside the write-accessible range

The specifics of access control vary depending on the hardware specifications, but all have the following points in common.

- When the access result is a prohibit judgment, it is not reflected in memory or I/O devices.
- When the access result is an enabled judgment, it is reflected in memory or I/O devices.

CAUTIONS

- 1. Even when access is enabled, there might be cases where access is blocked by another function that prohibits it.
- 2. In some cases, access judged to be prohibited may be executed for a memory or I/O device. The cases are as listed below.
 - Reading local RAM
 - Reading of code flash memory by an instruction prefetched from the instruction cache

Since execution in response to exceptions due to instructions that read from the local RAM or execute the results of prefetching and so on is inhibited, such access does not affect the execution of instructions. However, when a debugger is monitoring access to local RAM or code flash memory, it may observe access judged to be prohibited.



5.1.5 Violations and Exceptions

In this CPU, violations are detected during instruction fetch access or operand access according to the protection area settings, and an exception is generated.

- Execution protection violation (during instruction access)
- Data protection violation (during operand access)

(1) Execution protection violation (MIP exception)

This violation is detected when an instruction is executed. An execution protection violation such as this is detected when attempting to execute an instruction that has been placed in a non-executable area within the program area.

When an execution protection violation is detected, an MIP exception always occurs.

(2) Data protection violation (MDP exception)

This violation is detected during data access by an instruction. A data protection violation such as this is detected when a memory access instruction attempts to access data from an access-prohibited part of the data area.

When a data protection violation is detected, an MDP exception always occurs.

(3) Exception cause code and exception address

When an execution protection violation or data protection violation has been detected, the exception cause code is determined as shown in **Table 5.1**. The determined exception cause code is set to the FEIC register.

The MEA register is used to store either the PC of the instruction that detected the execution protection violation or the access address used when the data protection violation occurred. The MEA register is shared in order to prevent simultaneous occurrence of MIP and MDP exceptions. Also, when a data protection violation occurs, the information of the instruction that caused the violation is stored in the MEI register.

			Bit Number and Bit Name									
	Operation Mode When Violation	31 to 25	24	23	22	21	20	19	18	17	16	15 to 0
Exception	Occurred	-	MS	BL	RMW	SX	SW	SR	UX	UW	UR	_
MIP	User mode	0	0	0	0	—	_	—		_	_	90 _H
	Supervisor mode	0	0	0	0	—		—		—		90 _H
MDP	User mode	0	*5	*4	*3	0	0	0	0	*2	*1	91 _H
	Supervisor mode	0	_			0	*2	*1	0	0	0	91 _H

Table 5.1 Exception Cause Code of Memory Protection Violation

Note 1. When a read violation is caused by an instruction that includes a read operation, either the SR or UR bit is set to 1.

Note 2. When a write violation is caused by an instruction that includes a write operation, either the SW or UW bit is set to 1.

- Note 3. This bit is set to 1 when a violation is caused by the SET1, NOT1, CLR1, or CAXI instruction.
- Note 4. This bit is set to 1 when a violation is caused by the PREPARE, DISPOSE, PUSHSP, or POPSP instruction.

Note 5. This bit is set to 1 when the instruction causing the violation performs a misaligned access.

Note: UR : A violation is detected during a read operation in user mode (PSW.UM = 1).

UW : A violation is detected during a write operation in user mode (PSW.UM = 1).

SR : A violation is detected during a read operation in supervisor mode (PSW.UM = 0).

SW : A violation is detected during a write operation in supervisor mode (PSW.UM = 0).

SX : A violation is detected during instruction execution in supervisor mode (PSW.UM = 0).

UX : A violation is detected during instruction execution in user mode (PSW.UM = 1).

RMW : Set to 1 when the instruction causing the violation contains a read-modify-write operation (SET1, NOT1, CLR1, or CAXI). BL : Set to 1 when the instruction causing the violation performs a block transfer (PREPARE, DISPOSE, PUSHSP, or POPSP). MS : Set to 1 when the instruction causing the violation performs a misaligned access.

5.1.6 Memory Protection Setting Check Function

When configuring programs that provide a service for the OS (etc.), this CPU provides a memory protection setting check function to enable implementation of a service protection function that checks in advance whether or not the data area to be used for the requested operations is within an area that is accessible by the source that called the service. The OS can use this function to verify the suitability of parameters set for system services provided by the user. Also, this verification processing can be completed quickly when compared to software-based area setting read and comparison operations.

(1) Procedure

Set the base address (lower limit) of the target address range to the MCA register and the size of the target range to the MCS register, then use the LDSR instruction (r0 specification is recommended) to access the MCC register and execute a check. The results can be read from the MCR register by the STSR instruction.

CAUTIONS

- 1. If the specified area to be checked crosses $0000\ 0000_{H}$ or 7FFF FFFF_H, it is judged as an area setting error, and the MCR.OV bit is set to 1. This means that the MCR.OV bit must be checked to access the check results. Do not use the check result until it is confirmed that the result is not invalid (OV = 0).
- 2. If the default operations specified by using the MPM.DX, DW, and DR bits are enabled (1), the correct result might not be able to be obtained. If enabling the specified default operation, do not use the memory protection setting check function.

(2) Sample code

It is assumed that the memory protection setting check function will be used for the following operations.

```
service protection:
   ori
          0x1000, r0, r12
          ADDRESS, r10
                          // Store the start address of the area to be checked to r10
   mov
  mov
          SIZE, r11
                          // Store the size of the area to be checked to r11
   di
          r10, sr8, 5
                           // Set the address to MCA
   ldsr
  ldsr r11, sr9, 5
                          // Set the size to MCS
   ldsr
         r0, sr10, 5
                          // Start checking with MCC
          sr11, r12, 5
                          // Get the results from MCR
   stsr
   ei
          0x0100, r12, r0
   andi
   bnz
          _overflow
                           // Processing of invalid input when OV = 1
          _result_check // Otherwise, result is determined
  br
```



5.2 Cache

For information regarding the specific functions of mounted cache memory and which functions are mounted, see the hardware manual of the product used.

5.2.1 Cache Operation Registers

Figure 5.1 shows the system registers for cache operation. The supervisor privilege is required for the operation.



Figure 5.1 Cache Operation Registers

5.2.2 Change Cache Use Mode

(1) Change use mode of instruction cache

The instruction cache use mode can be changed by using the ICCTRL.ICHEN bit. To enable an instruction cache, set the ICHEN bit to 1.

To disable the instruction cache, clear the ICHEN bit to 0.

Completion in executing the LDSR instruction that sets ICHEN might not coincide with completion of the instruction cache operations. As in the following sample code, the SYNCI instruction is executed after the settings are changed to ensure that the change in instruction cache settings take effect.

LDSR	r10, sr24, 4	// Change the instruction cache settings (ICCTRL)
		(setting value is stored in r10)
SYNCI		<pre>// Syncs refetch to completion of LDSR instruction</pre>



5.2.3 Cache Operations using CACHE Instruction

The CACHE instruction manipulates data in the specified cache memory.

Such data manipulation by the CACHE instruction starts after updating of the cache memory by all preceding memory access has been completed. Consequently, the result of preceding memory access is guaranteed to be the target for operations using the CACHE instruction. Additionally, a suitable synchronization period is needed following execution of the CACHE instruction to ensure that the results are reflected in subsequent instructions.

(1) Specification method for target of CACHE instruction

There are basically two ways to specify the target for operations.

- Directly specify the address to be accessed : In this CPU, this is called the address specification method. In this case, the cache line containing the specified address is subject to operation.
- Directly specify the cache memory's way number and line number : In this CPU, this is called the index specification method. In this case, no hit judgment for the cache is performed, and the operation is performed on the specified cache index. For details about the cache index specification method, see **Section 5.2.5, Cache Index Specification Method**.

(2) Operations performed using the CACHE instruction

The operations performed on the cache memory are broadly divided into the six types described below. Some of these operations might not be supported, depending on the cache memory to be manipulated (instruction, data, etc.). For details about each operation, see **Section 7, INSTRUCTION**.

(a) Cache Hit Block Invalidate / Cache Indexed Block Invalidate (CHBI / CIBI)

This disables the specified cache line. When using the address specification method, the cache line is disabled only when there is a hit. When using the index method, the cache line is disabled. If the specified cache line is locked, it is unlocked. This operation can be used in cases such as when the entire memory cache is initialized by software.

(b) Cache Fetch And Lock (CFAL)

This stores the data at the specified address to the cache memory. At this time, the cache line where the data is stored is locked. This prevents the cache line from being switched. If the target cache line has already been stored in the cache memory, it is simply locked. If the target cache line has already been stored in the cache memory and is not locked, this operation is not performed.

This operation can be used to improve execution efficiency by reducing variations in instruction execution time that occur due to cache misses in the specified memory area.

CAUTION

The target cache line might not be able to be locked, such as when all cache ways are locked. This operation can be used to efficiently monopolize the cache memory, so note with caution the cache locking specifications and the number of cache ways. For details, see the hardware manual of the product used.

(c) Cache Indexed Load / Cache Indexed Store (CILD / CIST)

This operation is used to directly access the cache memory. Values can be written and read, via a system register, at a position in the cache memory specified by using an index. Because cache data and cache tags can be accessed directly, this operation can be used for purposes such as software debugging.

(d) Other operations

Other special operations related to manipulating the bus and memory, such as deleting links to enable efficient exclusive access, might also be defined as cache operations. For details, see **Section 7**, **INSTRUCTION**.

5.2.4 Cache Operation when the PREF Instruction is Executed

The PREF instruction is provided to realize efficient cache access by advising the CPU that an address is likely to be used in a certain way in the near future. Getting the CPU to prefetch data into the cache memory before use in this way can reduce the read wait time when a cache miss occurs.

Assuming support by compilers and other tools, the PREF instruction can be executed regardless of the CPU operating mode. Execution of the PREF instruction does not cause an exception generated by the MPU, and has no effect on logical operations, just like a NOP instruction.

CAUTION

Because a data read request by the PREF instruction is rather speculative, it might not be executed depending on the cache control policy or system conditions. For details, see the hardware manual of the product used.



5.2.5 Cache Index Specification Method

For a cache instruction that uses the index specification method, explicitly specify the cache memory subject to operation in the format shown in **Figure 5.2**, instead of specifying an address. The bit positions (x, y, z) of each field depend on the size of the cache memory incorporated in the CPU core. Information about the incorporated cache memory and size can be read from the ICCFG register.





CAUTION

The Offset field indicates the byte position within the cache line. This setting is not required (i.e., ignored) in normal index specification operations. For a CILD/CIST operation, it is used to specify a position within the cache line when the ICDAT[HL] register is shorter than the cache line length.

5.2.6 Execution Privilege of the CACHE/PREF Instruction

Because the CACHE instruction directly manipulates the contents of the cache memory, privileges are specified according to the type of operation. When the CACHE instruction is executed without the privilege required for the CACHE operation, a privilege instruction exception (PIE) occurs.

On the other hand, the PREF instruction provides information for speculative execution, so it can be executed in any mode.

The privileges required by the different operations performed by the CACHE instruction are shown below.

(a) Operations allowed with the user privilege

Among address specification method operations, operations without a cache lock (CHBI) can be executed in any operation mode.

(b) Operations requiring the supervisor privilege

Among address specification method operations, operations with a cache lock (CFAL) require the supervisor privilege.

In addition, index specification method operations require the supervisor privilege.



5.2.7 Memory Protection for CACHE and PREF Instructions

When manipulating the cache by specifying an address for the CACHE instruction, it might become the target of memory protection by the MPU. Memory protection is judged based on the operating mode in which the CACHE instruction is executed, and it is handled as a data-side access.

No memory protection judgment is performed when using the index specification method or the PREF instruction.

Table 5.2 shows the correspondence between operations and access privileges.

Table 5.2 Relationship Between Cache Operations and Permissions

Instruction	Target	Address/Index	Instruction Execution Privilege	Access Permission
CHBII	Instruction	Address	UM	Read
CIBII	Instruction	Index	SV	_
CFALI	Instruction	Address	SV	Read
CISTI	Instruction	Index	SV	_
CILDI	Instruction	Index	SV	—
(CLL instruction)*1		_	—	_

Note 1. Functions as the CLL instruction. For details, see the description of the CLL instruction in Section 7, INSTRUCTION.



5.3 Mutual Exclusion

This CPU provides instructions that enable shared resources to be controlled mutually exclusively from multiple programs when the system is operating in a multi-processor environment.

When using mutual exclusion, mutual exclusion variables have to be defined in the memory and all programs must operate in accordance with the appropriate instruction flow.

CAUTION

Embedded CPUs in a single-processor configuration use a programming model in which data coherence is maintained by disabling the acknowledgment of maskable interrupts. This is a very easy and sure method of maintaining data coherence, but naturally in a multi-processor, multiple programs might be executing and attempting to use the data at the same time. In this case it is not possible to maintain data coherence simply by disabling maskable interrupt acknowledgment.

5.3.1 Shared Data that does not Require Mutual Exclusion Processing

This CPU maintains data access coherence even in a multi-processor environment by enabling the following types of access.

- Access in which the data is aligned to the size that matches the data type (aligned access)
 - LD, ST, SLD, SST, LDL, and STC instructions
- Access by using a bit manipulation instruction (SET1, CLR1, or NOT1) (read-modify-write)
- Access by using the CAXI instruction (read-modify-write)

With some exceptions, mutual exclusion is achieved by using these types of data access. In other words, it is guaranteed that while one CPU is executing the instructions that perform the above data accesses, another CPU is not accessing the data in question. This is known as an instruction being executed atomically or an instruction providing an atomic guarantee.

Note that the atomic execution of an instruction means that a data access bus transaction completes with no disruption; it does not necessarily mean that a series of transactions has been completed.

CAUTION

The extent to which coherency is guaranteed might be limited, depending on the hardware specifications. For example, for some memories, coherency might not be preserved even if aligned access is used. For details, see the hardware manual of the product used.



5.3.2 Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions

The LDL.W and STC.W instructions can be used to perform mutual exclusion over multiple data arrays.

When acquiring a lock by using the LDL.W and STC.W instructions in a pair, first a link is created by using the LDL.W instruction and then the STC.W instruction is executed.

At this time, if data is written to the address at which the link was created before the STC.W instruction is executed, the link is immediately deleted, the subsequent execution of the STC.W instruction fails, and a lock fails to be acquired.

(1) Link

Each link (LLbit) includes information on the address at which it was created, which is used to control whether the STC instruction executes successfully or fails, and whether the link is deleted.

A link is created when the LDL.W instruction is executed. If the LDL.W instruction is executed again after a link has been created, another link is created, which overwrites the first link. In other words, only one link exists at a time, and that link contains the address information of the LDL.W executed last.

Links are deleted when certain event or address conditions are satisfied. **Table 5.3** shows the link deletion conditions. A link is deleted if any of the conditions shown in **Table 5.3** are satisfied.

Target Link	Event Condition	Remark
All links in the system (including those in other CPU cores)	If a write operation occurs in a 32-byte-aligned address range that includes the address of the link in question	ST, SST, and STC instructions SET1, NOT1, CLR1, and CAXI instructions PREPARE and PUSHSP instructions
CPU core link	Execution of STC.W instruction	The link is deleted whether the instruction executes successfully or fails
	Execution of CLL instruction	Use a CLL instruction to clear a link in a function explicitly (abortion of an atomic operation).
	Exception acknowledgment	
	Execution of return instruction	Does not include CTRET instruction

Table 5.3 Link Deletion Conditions

CAUTION

Links that are deleted by a write operation are deleted in 32-byte units. Therefore, the best way to prevent execution of the STC.W instruction from failing in this case is to allocate only one mutual exclusion variable per 32 bytes of memory. If more than one mutual exclusion variable is allocated in a 32-byte range, thrashing might occur when an attempt is made to acquire a lock on a mutual exclusion variable.

(2) Sample code

The sample code of a spinlock executed by using the LDL.W and STC.W instructions is shown below.

Lock acquisition

	mov	lock_adr,	r20
Lock:	ldl.w	[r20], r21	
	cmp	r0, r21	
	bnz	Lock_wait	
	mov	1, r21	
	stc.w	r21, [r20]	
	cmp	r0, r21	
	bnz	Lock_success	
Lock_wa	it:		
	snooze		
	br	Lock	
Lock_su	ccess:		

Lock release

st.w r0, 0[r20]



5.3.3 Performing Mutual Exclusion by Using the SET1 Instruction

The SET1 instruction can be used to perform mutual exclusion over multiple data arrays. By executing the SET1 instruction on the same bit in the memory and then checking the PSW.Z flag, which indicates the execution result, it can be determined whether lock acquisition succeeded or failed.

CAUTIONS

- 1. Depending on the hardware specifications, the system performance might drop if exclusive control is executed frequently by using the SET1 instruction, because this causes the bus to be occupied for a long time. It is therefore recommended to execute exclusive control by using the LDL/STC instructions as much as possible.
- 2. When performing mutual exclusion by using the SET1 instruction, to prevent the problem of excessive bus occupancy described in Caution 1 above, execute the snooze instruction before attempting to acquire a lock again after lock acquisition has failed, and adjust the lock acquisition loop execution interval.

(1) Sample code

The sample code of a spinlock executed by using the SET1 instruction is shown below.

Lock acquisition

	mov	lock_adr, r20				
Lock:	set1	0, 0[r20]				
	bz	Lock_success				
	snooze					
	br	Lock				
Lock_su	Lock_success:					

Lock release

	clr1	0, 0[r20]
--	------	-----------



5.3.4 Performing Mutual Exclusion by Using the CAXI Instruction

The CAXI instruction can be used to perform mutual exclusion over multiple data arrays. By executing the CAXI instruction on the same word in the memory and then checking the destination register, it can be determined whether lock acquisition succeeded or failed.

CAUTIONS

- 1. Depending on the hardware specifications, the system performance might drop if exclusive control is executed frequently by using the CAXI instruction, because this causes the bus to be occupied for a long time. It is therefore recommended to execute exclusive control by using the LDL/STC instructions as much as possible.
- 2. When performing mutual exclusion by using the CAXI instruction, to prevent the problem of excessive bus occupancy described in Caution 1 above, execute the snooze instruction before attempting to acquire a lock again after lock acquisition has failed, and adjust the lock acquisition loop execution interval.

(1) Sample code

The sample code of a spinlock executed by using the CAXI instruction is shown below.

Lock acquisition

	mov	lock_adr, r20			
Lock:	mov	1, r21			
	caxi	[r20], r0, r21			
	bz	Lock_success			
	snooze				
	br	Lock			
Lock_suc	Lock_success:				

Lock release

st.w	r0, 0[r20]



5.4 Synchronization Function

In order to improve the processing performance, this CPU executes subsequent instructions before the operation of the preceding instruction is completed, when there is no dependency between the instructions. For this reason, when the subsequent instructions need to wait for the completion of the operation of the preceding instruction, the synchronization procedure is required. This CPU provides the following four special instructions for the synchronization.

The SYNCP instruction is the special instruction, which synchronizes the pipeline to reflect the result of the preceding instructions to the subsequent instructions. The SYNCP instruction waits for the result of load instructions (until the loaded data is stored in a register), but does not wait for the result of store instructions (until the destination memory or memory-mapped control register is updated). Therefore, when the result of store instruction needs to be reflected to the subsequent instructions, perform a dummy read of the destination memory or control register of the store instruction, and then execute the SYNCP instruction.

The SYNCM instruction is the special instruction, which synchronizes memory accesses. The SYNCM instruction waits for the result of all preceding load instructions (until the loaded data is stored in a register) and the result of all preceding store instructions (until the destination memory or memory-mapped control register is updated). However, the SYNCM instruction may not guarantee the completion of updating of the memory or control register if it is attached to the bus-system or peripheral device, which completes store operation speculatively (i.e., updating of the memory or control register is delayed). When the result of updating of such memory or control register needs to be reflected to the subsequent instructions, perform a dummy read of the destination memory or control register of the store instruction, and then execute the SYNCP instruction.

The SYNCI instruction is the special instruction, which synchronizes instruction fetches. The SYNCI instruction discards unexecuted instructions in the pipeline, and re-fetches the subsequent instructions. The SYNCI instruction is used to reflect the result of the preceding instructions to the instruction fetch of the subsequent instructions. When the result of the store instruction needs to be reflected to the instruction fetch of the subsequent instruction (e.g., when updating memory to realize self-programming program or updating the control register to switch the code flash memory area), perform a dummy read of the destination of the store instruction, execute the SYNCP instruction, and then execute the SYNCI instruction.

In this CPU, the SYNCE instruction is handled as the NOP instruction.

Table 5.4 shows the effect of the synchronization instructions.

For the hazard resolution procedure for system registers, see **APPENDIX A, HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS**.



	Synchronization Guaranteed by the SYNC Instruction							
	Synchronization	of Instruction Fetch	Synchronization	n of Execution of the P	receding Instruction			
SYNC Instruction	Re-fetch of Subsequent Instructions	Cache Instruction/ Instruction to Update Cache Operation Function Register	Calculation Instruction	Load Instruction	Store Instruction			
SYNCP	-	—	Completion of execution	Completion of execution* ¹	—			
SYNCM	-	—	Completion of execution	Completion of execution* ¹	Completion of execution* ²			
SYNCI	Re-fetch after synchronization of execution of the preceding instruction	Completion of execution	Completion of execution	_	-			
SYNCE	 _	 	 	_	—			

Table 5.4 Effect of Synchronization Instructions

Remark: "-": Not guaranteed

Note 1. The SYNC instruction waits until the loaded data is stored in a register.

Note 2. The SYNC instruction waits until the destination memory or control register is updated. However, there may exist destinations, whose update cannot be guaranteed by the SYNC instruction. For details, see the hardware manual of the product used.



Section 6 COPROCESSOR

6.1 Floating-Point Operation

The floating-point unit (FPU) operates as the CPU coprocessor, and executes floating-point instructions.

Either single-precision (32-bit) or double-precision (64-bit) data can be used. In addition, the conversion between a floating point type and an integer type is possible.

The FPU of this CPU conforms to ANSI/IEEE standard 754-2008 (IEEE Standard for Floating-Point Arithmetic).

6.1.1 Configuration of Floating-Point Operation Function

(1) Not implemented

If the floating-point operation function is not implemented, all the floating-point instructions cannot be used. If an attempt is made to execute such an instruction, a coprocessor unusable exception occurs. In addition, the operation of all the floating-point system registers is undefined. Therefore, do not manipulate these registers by LDSR and STSR.

(2) Implementing only single precision

If only the floating-point operation function with single precision is implemented, only floating-point instructions classified as single precision^{*1} can be used. If an attempt is made to execute a floating-point instruction classified as double precision^{*2}, a coprocessor unusable exception occurs. All the floating-point system registers supply the function described in **Section 3.4, FPU Function Registers**.

- **Note 1.** The single-precision floating-point instruction is the instruction described as (Single) in the description of each instruction in **Section 7.4.4, Floating-Point Instruction Set**.
- **Note 2.** The double-precision floating-point instruction is the instruction described as (Double) in the description of each instruction in **Section 7.4.4, Floating-Point Instruction Set**.

(3) Implementing single precision and double precision

All the floating-point instructions can be used when floating-point instructions of single precision and double precision are implemented. All the floating-point system registers supply the functions described in **Section 3.4, FPU Function Registers**.



6.1.2 Data Types

(1) Floating-point format

The FPU supports 32-bit (single precision) and 64-bit (double precision) IEEE754 floating-point operations.

The single-precision floating-point format consists of a 24-bit signed fraction (s + f) and an 8-bit exponent (e), as shown in **Figure 6.1**.

0	23 22 22	30
	e f	
	Exponent Fraction	jn
	8 23	
		JII

Figure 6.1 Single-precision Floating-point Format

The double-precision floating-point format consists of a 53-bit signed fraction (s + f) and an 11-bit exponent (e), as shown in **Figure 6.2**.



Figure 6.2 Double-precision Floating-Point Format

A numerical value in the floating-point format includes the following three areas.

- Sign bit: s
- Exponent: e = E + bias value
- Fraction: $f = .b_1b_2...b_{P-1}$ (value lower than the first decimal place)

The bias value for the single-precision format is 127. For double-precision format, the bias value is 1023.

The range of the exponent value E when unbiased covers all integers from Emin to Emax, along with two reserved values, Emin -1 (± 0 or subnormal number), and Emax +1 ($\pm \infty$ or NaN: not-a-number). A numeric value other than 0 is represented in one format, depending on the single-precision and double-precision formats.

The numeric value (v) represented in this format can be calculated by the expression shown in **Table 6.1**.



Туре	Calculation	n Expression
NaN (not-a-number)	If E = Emax + 1 and $f \neq 0$	then v = NaN regardless of s
±∞ (infinite number)	If $E = Emax + 1$ and $f = 0$	then v = (−1) ^s ∞
Normalized number	If $Emin \le E \le Emax$	then $v = (-1)^{s}2^{E} (1.f)$
Subnormal number	If E = Emin $- 1$ and f $\neq 0$	then $v = (-1)^{s} 2^{Emin} (0.f)$
±0 (zero)	If $E = Emin - 1$ and $f = 0$	then $v = (-1)^{s}0$

Table 6.1 Calculation Expression of Floating-Point Value

• NaN (not-a-number)

IEEE754 defines a floating-point value called NaN (not-a-number). Because this value is not a numerical value, it does not have any "greater than" or "less than" relationships to other values. If v is NaN in all of the floating-point formats, it might be either SignalingNaN (S-NaN) or QuietNaN (Q-NaN), depending on the value of the most significant bit of f. If the most significant bit of f is set, v is QuietNaN; if the most significant bit is cleared, it is SignalingNaN.

Table 6.2 shows the value of each parameter defined in floating-point formats.

Table 6.2 Floating-Point Formats and Parameter Values

	Format								
Parameter	Single Precision	Double Precision							
Emax	+127	+1023							
Emin	-126	-1022							
Bias value of exponent	+127	+1023							
Length of exponent (number of bits)	8	11							
Integer bits	Cannot be seen	Cannot be seen							
Length of fraction (number of bits)	23	52							
Length of format (number of bits)	32	64							

Table 6.3 shows the minimum and maximum values that can be represented in floating-point formats.

Table 6.3	Floating-Point	Minimum and	Maximum	Values
-----------	----------------	-------------	---------	--------

Туре	Value
Minimum value of single-precision floating point	1.40129846e – 45
Minimum value of single-precision floating point (normal)	1.17549435e – 38
Maximum value of single-precision floating point	3.40282347e + 38
Minimum value of double-precision floating point	4.9406564584124654e - 324
Minimum value of double-precision floating point (normal)	2.2250738585072014e - 308
Maximum value of double-precision floating point	1.7976931348623157e + 308



(2) Fixed-point formats

The value of a fixed point is held in the format of 2's complement. **Figure 6.3** shows a 32-bit fixed-point format and **Figure 6.4** shows a 64-bit fixed-point format. No signed bits exist in the unsigned fixed-point format, and all bits represent the integer value.



Figure 6.3 32-bit Fixed-Point Format



Figure 6.4 64-bit Fixed-Point Format

(3) Expanded floating-point format

This CPU supports the 16-bit (half-precision) IEEE754 floating-point format as a floating-point format for storing data. The half-precision floating-point format is used to decrease the amount of data; it is not supported for arithmetic operations. Instructions are available for converting single-precision floating-point format data into half-precision floating-point data and vice-versa. The half-precision floating-point format consists of an 11-bit signed fraction (s + f) and a 5-bit exponent (e), as shown in **Figure 6.5**.







Like other floating-point formats, the numeric values represented in this format can be calculated by using the expressions shown in **Table 6.1**. The values of the parameters defined by the half-precision floating-point format are shown in **Table 6.4**.

Table 6.4 Half-Precision Floating-Point Format and Parameter Values

Parameter	Half Precision	
Emax	+15	
Emin	-14	
Bias value of exponent	+15	
Length of exponent (number of bits)	5	
Integer bits	Cannot be seen	
Length of fraction (number of bits)	10	
Length of format (number of bits)	16	

Table 6.5 shows the minimum and maximum values that can be represented in the half-precision floating-point format.

Table 6.5 Half-Precision Floating-Point Minimum and Maximum Values
--

Туре	Value
Minimum value of half-precision floating point	5.96046e ⁻⁸
Maximum value of half-precision floating point (normal)	6.10352e ⁻⁵
Maximum value of half-precision floating point	65504

6.1.3 Register Set

For details about the register set, see Section 3.4, FPU Function Registers.

6.1.4 Floating-Point Instructions

Floating-point instructions are divided into single-precision instructions (single) and double-precision instructions (double).

For details about the floating-point instructions, see Section 7.4, Floating-Point Instructions.



6.1.5 Floating-Point Operation Exceptions

This section describes how the FPU processes floating-point operation exceptions.

(1) Types of exceptions

When floating-point operations or processing of operation results cannot be done using the ordinary method, a floating-point operation exception occurs.

One of the following two operations is performed when a floating-point operation exception has occurred.

• When exceptions are enabled

The cause bit is set in the floating-point configuration/status register (FPSR), and processing (by software) is passed to the exception handler routine.

• When exceptions are prohibited

The preservation bit is set in the floating-point configuration/status register (FPSR), an appropriate value (initial value) is stored in the FPU destination register, then execution is continued.

The FPU uses cause bits, enable bits, and preservation bits (status flags) to support the following five types of IEEE754-defined exception causes.

- Inexact operation (I)
- Overflow (O)
- Underflow (U)
- Division by zero (Z)
- Invalid operation (V)

A sixth type of exception cause is unimplemented operation (E), which causes an exception when a floating-point operation cannot be executed. This exception requires processing by software. An unimplemented operation exception (E) occurs when exceptions are always enabled, rather than by using properties, enable bits, or preservation bits.





Figure 6.6 shows the FPSR register bits that are used to support exceptions.

Figure 6.6 Cause, Enable, and Preservation Bits of FPSR Register

The five exceptions (V, Z, O, U, and I) defined by IEEE754 are enabled when the corresponding enable bits are set. When an exception occurs, if the corresponding enable bit has been set, the FPU sets the corresponding cause bit. If the exception can be acknowledged, processing is passed to the exception handler routine. If exceptions are prohibited, the exception corresponding preservation bit is set, and processing is not passed to the exception handler routine.



(2) Exception handling

When a floating-point operation exception occurs, the cause bits of the FPSR register indicate the cause of the floating-point operation exception.

(a) Status flag

A corresponding preservation bit is available for each IEEE754-defined exception. The preservation bit is set when the corresponding exception is prohibited but the exception condition has been detected. The preservation bit is set or reset whenever new values are written to the FPSR register by the LDSR instruction.

If an exception is prohibited by an enable bit, predetermined processing is performed by the FPU. This processing provides an initial value as the result, rather than a floating-point operation result. This initial value is determined according to the type of exception. For an overflow exception or underflow exception, the initial value also differs depending on the current rounding mode. **Table 6.6** shows the initial values provided for each of the FPU IEEE754-defined exceptions.

Area	Description	Rounding Mode	Initial Value
V	Invalid operation		Quiet not-a-number (Q-NaN)
Z	Division by zero		Correctly signed ∞
0	Overflow	RN	∞ with sign of intermediate result
		RZ	Maximum normalized number with sign of intermediate result
		RP	Negative overflow: Maximum negative normalized number Positive overflow: +∞
		RM	Positive overflow: Maximum positive normalized number Negative overflow: $-\infty$
U	Underflow ^{*1}	RN* ²	0 with sign of intermediate result
		RZ	0 with sign of intermediate result
		RP	Positive underflow: Minimum positive normalized number Negative underflow: 0
		RM	Negative underflow: Minimum negative normalized number Positive underflow: 0
I	Inexact operation	_	Rounded result

 Table 6.6
 FPU Initial Values for IEEE754-Defined Exceptions

Note 1. If the FPSR.FS bit is cleared, an unimplemented operation exception (E) will occur if an underflow occurs in the rounded result; an underflow exception (U) will not occur. If the FS bit of the FPSR register is set, the flushed result is used as the default value

Note 2. If the rounding mode is RN and the FN bit of the FPSR register is set, flushing will occur in the direction of higher accuracy. For details, see Section 6.1.9, Flush to Nearest.



6.1.6 Exception Details

The following describes the conditions under which each of the FPU exceptions occurs and the FPU responses.

(1) Inexact exception (I)

In the following cases, the FPU detects an inexact exception.*1

- When the precision of the rounded result is dropped
- When the rounded result overflows while overflow exceptions are prohibited
- When the rounded result underflows while underflow exceptions are prohibited
- When the operand that is a subnormal number is flushed, neither an invalid operation exception (V) nor a division by zero exception (Z) is detected, and the other operands are not Q-NaN
- Note 1. Even when there is no remainder, the inexact exception (I) bit may be set by floating-point divide instructions (DIVF.S and DIVF.D). For details, see Section 7.4.4, Floating-Point Instruction Set.

CAUTION

If the FS bit of the FPSR register is cleared and the operation result underflows, an unimplemented operation exception (E) occurs. In such cases, the underflow exception is not detected, so the inexact exception is not detected either.

(a) If exception is enabled

The contents of the destination register are not changed, contents of the source register are saved, and an inexact exception occurs.

(b) If exception is not enabled

If no other exception occurs, the rounded result or the result that underflows or overflows is stored in the destination register.



(2) Invalid operation exception (V)

An invalid operation exception occurs when one of both of the operands is invalid.

- Arithmetic operation with S-NaN included in operands. The conditional move instruction (CMOV), absolute value (ABS), and arithmetic negation (NEG) are not handled as arithmetic operations, but minimum value (MIN) and maximum value (MAX) are handled as arithmetic operations.
- Multiplication: $\pm 0 \times \pm \infty$ or $\pm \infty \times \pm 0$
- Fused-multiply-add: $(\pm 0 \times \pm \infty) + c$ or $(\pm \infty \times \pm 0) + c$. But only if c is not Q-NaN.
- Addition/subtraction or multiply-add operation*¹:
 Addition of infinite values with different signs or subtraction of infinite values with the same sign
- Division: $\pm 0 \div \pm 0$ or $\pm \infty \div \pm \infty$
- Square root: When operand is less than 0
- Conversion to integer when source is outside of integer range.
- Comparison: With condition codes 8 to 15, if the operand is unordered (see **Table 7.10**, **Definitions of Condition Code Bits and Their Logical Inversions**)

Note 1. When the multiplication result is infinite or when adding or subtracting between infinities

(a) If exception is enabled

The contents of the destination register are not changed, contents of the source register are saved, and an invalid operation exception occurs.

(b) If exception is not enabled

If no other exception occurs, and the destination is a floating-point format, Q-NaN is stored in the destination register. If the destination has an integer format, see the operation result description of each instruction for the value to be stored in the destination register.

(3) Division by zero exception (Z)

A division by zero exception occurs when a divisor is 0 and a dividend is a finite number other than 0.

(a) If exception is enabled

The contents of the destination register are not changed, contents of the source register are saved, and a division by zero exception occurs.

(b) If exception is not enabled

If no other exception occurs, a correctly signed infinite number $(\pm \infty)$ is stored in the destination register.

(4) Overflow exception (O)

An overflow exception is detected if the exponent range is infinite and if the result of the rounded floating point is greater than maximum finite number in the destination format.

(a) If exception is enabled

The contents of the destination register are not changed, the contents of the source register are saved, and an overflow exception occurs.

(b) If exception is not enabled

If no other exception occurs, the initial value that is determined by the rounding mode and the sign of the intermediate result is stored in the destination register (see **Table 6.6, FPU Initial Values** for IEEE754-Defined Exceptions).

(5) Underflow exception (U)

If the operation result is -2^{Emin} to $+2^{\text{Emin}}$ (but not zero), an underflow exception is detected.

Although IEEE754 defines several methods for detecting an underflow, the same method should be used to detect underflows, regardless of the processing to be performed.

The following two methods can be used to detect an underflow for binary floating point numbers.

- The result calculated after rounding and using an infinite exponent range is not zero and is within $\pm 2^{\text{Emin}}$.
- The result calculated before rounding and using an infinite exponent range and precision is not zero and is within ±2^{Emin}.

In this CPU, an underflow is detected before rounding.

Or the rounded result is one of the following, an inexact result is detected.

• When a given result differs from the result calculated when the exponent range and precision are infinite)

In this CPU, the behavior when an inexact result is detected differs as follows depending on whether underflow exceptions are enabled or disabled:

(a) If exception is enabled

When the FS bit of the FPSR register has been set, if exceptions are enabled, an underflow exception (U) occurs. When the FS bit of the FPSR register has been set, if exceptions are not enabled but inexact exceptions are enabled, an inexact exception (I) occurs.

(b) If exception is not enabled

If the FS bit of the FPSR register has been set, the initial value determined according to the rounding mode and intermediate result value is stored in the destination register (see **Table 6.6**, **FPU Initial Values for IEEE754-Defined Exceptions**).

CAUTION

If the FS bit of the FPSR register has not been set, an unimplemented operation exception (E) occurs regardless of whether or not exceptions are enabled. Because an unimplemented operation exception (E) must occur, an underflow exception (U) does not occur.

(6) Unimplemented operation exception (E)

The E bit is set and an unimplemented operation exception (E) occurs when an abnormal operand or abnormal result that cannot be correctly processed by hardware has been detected. The operand and destination register contents do not change.

If the FS bit of the FPSR register has been set, an unimplemented operation exception (E) will not occur.

If the FS bit of the FPSR register has been cleared, an unimplemented operation exception (E) will occur under the following conditions (except for CMOVF.D, CMOVF.S, CMPF.D, CMPF.S, ABSF.D, ABSF.S, MAXF.D, MAXF.S, MINF.D, MINF.S, NEGF.D, NEGF.S and CVTF.HS instructions).

- When the operand is a subnormal number
- When the operation result is a subnormal number, or an underflow has occurred

CAUTION

If the FS bit of the FPSR register is set to 1, an unimplemented operation exception (E) will not occur under any circumstances.



6.1.7 Saving and Returning Status

When a floating-point operation exception occurs, the PC and PSW are saved to the EIPC and EIPSW registers respectively, and the exception code is saved to the EIIC register.

When an EI level exception is acknowledged while processing a floating-point operation exception, an EIPC register override occurs, which prevents the returning to the instruction that caused the floating-point operation exception to occur. When acknowledgment of EI level exceptions is required, the contents of the EIPC, EIPSW, and EIIC registers must be saved, such as to a stack.

When a floating-point instruction is used in a floating-point operation exception handler routine, the FPSR and FPEPC registers will be overridden if another floating-point operation exception occurs. In such cases, the FPSR and FPEPC registers should be saved at the start of the floating-point operation exception handler processing, and should be returned at the end of the handler processing.

The cause bits of the FPSR register hold the results from only one enabled exception. In any case, the previous results are held until the next enabled exception occurs.



6.1.8 Flushing Subnormal Numbers

This CPU can process subnormal numbers—very small numbers that are lower than the minimum normalized number—in one of the following two ways:

- Normalize the operand or operation result and continue executing arithmetic processing
- Generate an unimplemented operation exception (E) and execute exception handling

Executing software-based exception handling will obtain a more accurate result, but the amount of time required to obtain the result will vary depending on the input value. In control systems that require a real-time performance, therefore, this is usually unacceptable. In this case, it is important to obtain the result within a certain amount time rather than focus on accuracy.

(1) Normalize the subnormal numbers and continue executing arithmetic processing

By setting the FS bit of the FPSR register to 1, this CPU can normalize the operand or operation result to a specific value and continue executing arithmetic processing if a subnormal number is input as the operand or obtained as the operation result. At this time, extremely small differences in values might not appear in the operation result.

For the operand and operation result, the values to which subnormal numbers are flushed when the FS bit is set (1) are shown in **Table 6.7** and **Table 6.8** below.

Table 6.7	Rounding Mode ar	d Flush Value	of Input Operand
-----------	------------------	---------------	------------------

	Roundi	ng Mode and '	Value to Which Flushed	Input Operand Is
Sign of Subnormal Operand	RN	RZ	RP	RM
+			+0	
			-0	

Table 6.8 Rounding Mode and Flush Value of Operation Result

	Rounding	•	Value to Which O Is Flushed	peration Result
Sign of Subnormal Operation Result	RN ^{Note}	RZ	RP	RM
+	+0	+0	+2 ^{Emin}	+0
_	-0	-0	-0	-2 ^{Emin}

Note 1. If the rounding mode is RN and the FN bit of the FPSR register is set, flushing will occur in the direction of higher accuracy. For details, see Section 6.1.9, Flush to Nearest.

Whether an input operand that is a subnormal number has been flushed or not can be checked by referencing the IF bit of the FPSR register. Whether an operation result that is a subnormal number has been flushed or not can be checked by referencing the U bit of the FPSR register.

CAUTIONS

- 1. In control systems that require a real-time performance, it is recommended to always set the FS bit to 1.
- 2. If the FS bit of the FPSR register is set (1), an unimplemented operation exception (E) will not occur under any circumstances.
- 3. Whether the operation result is a subnormal number is judged by using the value before rounding.
- 4. The IF bit of the FPSR register also accumulates and indicates information about flushing instructions that have caused a floating-point operation exception.

(2) Generate an unimplemented operation exception (E) and execute exception handling

By clearing the FS bit of the FPSR register to 0, an unimplemented operation exception (E) will occur if a subnormal number is input as the operand or obtained as the operation result. When an unimplemented operation exception occurs, software-based progressive underflow processing is performed in the floating-point operation exception handling routine, enabling a more accurate result to be obtained. In this case, however, a real-time processing performance might not be realized due to the software processing load.

(3) Instructions that can handle subnormal numbers

The following instructions can be executed without causing an unimplemented operation exception even if an operand that is a subnormal number is input while the FS bit of the FPSR register is 0.

- Conditional move instruction (CMOV), absolute value (ABS), arithmetic negation (NEG)
- Minimum value (MIN), maximum value (MAX), compare (CMPF)
- Conversion from half-precision to single-precision (CVTF.HS)

(4) Instructions that are not affected by flushing subnormal numbers

For the following instructions, flushing does not occur even an operand that is a subnormal number is input while the FS bit of the FPSR register is 1.

- Conditional move instruction (CMOV), absolute value (ABS), arithmetic negation (NEG)
- Minimum value (MIN), maximum value (MAX), compare (CMPF)
- Conversion from half-precision to single-precision (CVTF.HS)



6.1.9 Flush to Nearest

This CPU provides flush-to-nearest mode, a feature for flushing to the nearest number with higher accuracy when a flushing operation results subnormal number. Flush-to-nearest mode is enabled when the rounding mode is RN and the FN bit of the FPSR register is set (1). When this mode is used, the FPU determines the value to which to flush the subnormal number based on the number of the operation result and not just the sign. However, the result is flushed to $\pm 2^{\text{Emin}}$, which is different from the value shown in **Table 6.9**, when the operation result of the subtract operation by SUBF, FMSF, FNMSF instructions and the add operation of a negative value by ADDF, FMAF, FNMAF instructions becomes $\pm 2^{(\text{Emin-2})}$. This feature has no effect in rounding modes other than RN or on the result of flushing an input operand.

	Rounding Mode and Value to Which Operation Result Is Flushed										
Value of Subnormal Operation		RN									
Result	FN = 1	FN = 0	RZ	RP	RM						
$+2^{\text{Emin-1}} \le \text{Operation result} < +2^{\text{Emin}}$	+2 ^{Emin}	+0	+0	+2 ^{Emin}	+0						
+0 < Operation result < +2 ^{Emin-1}	+0										
$-2^{\text{Emin-1}}$ < Operation result < -0	-0	-0	-0	-0	-2 ^{Emin}						
$-2^{\text{Emin}} < \text{Operation result} \le -2^{\text{Emin-1}}$	-2 ^{Emin}										

 Table 6.9
 Rounding Mode and Value to Which Operation Result is Flushed

CAUTION

Whether the operation result is a subnormal number is judged by using the value before rounding.



Section 7 INSTRUCTION

7.1 Opcodes and Instruction Formats

This CPU has two types of instructions: CPU instructions, which are defined as basic instructions, and coprocessor instructions, which are defined according to the application.

7.1.1 CPU Instructions

Instructions classified as CPU instructions are allocated in the opcode area other than the area used in the format of the coprocessor instructions shown in **Section 7.1.2, Coprocessor Instructions**.

CPU instructions are basically expressed in 16-bit and 32-bit formats. There are also several instructions that use option data to add bits, enabling the configuration of 48-bit and 64-bit instructions. For details, see the opcode of the relevant instruction in **Section 7.2.2, Basic Instruction Set**.

Opcodes in the CPU instruction opcode area that do not define significant CPU instructions are reserved for future function expansion and cannot be used. For details, see **Section 7.1.3, Reserved Instructions**.

(1) reg-reg instruction (Format I)

A 16-bit instruction format consists of a 6-bit opcode field and two general-purpose register specification fields.



(2) imm-reg instruction (Format II)

A 16-bit instruction format consists of a 6-bit opcode field, 5-bit immediate field, and a general-purpose register specification field.





(3) Conditional branch instruction (Format III)

A 16-bit instruction format consists of a 4-bit opcode field, 4-bit condition code field, and an 8-bit displacement field.



(4) 16-bit load/store instruction (Format IV)

A 16-bit instruction format consists of a 4-bit opcode field, a general-purpose register specification field, and a 7-bit displacement field (or 6-bit displacement field + 1-bit sub-opcode field).



In addition, a 16-bit instruction format consists of a 7-bit opcode field, a general-purpose register specification field, and a 4-bit displacement field.



(5) Jump instruction (Format V)

A 32-bit instruction format consists of a 5-bit opcode field, a general-purpose register specification field, and a 22-bit displacement field.



(6) 3-operand instruction (Format VI)

A 32-bit instruction format consists of a 6-bit opcode field, two general-purpose register specification fields, and a 16-bit immediate field.



(7) 32-bit load/store instruction (Format VII)

A 32-bit instruction format consists of a 6-bit opcode field, two general-purpose register specification fields, and a 16-bit displacement field (or 15-bit displacement field + 1-bit sub-opcode field).



(8) Bit manipulation instruction (Format VIII)

A 32-bit instruction format consists of a 6-bit opcode field, 2-bit sub-opcode field, 3-bit bit specification field, a general-purpose register specification field, and a 16-bit displacement field.



(9) Extended instruction format 1 (Format IX)

This is a 32-bit instruction format that has a 6-bit opcode field and two general-purpose register specification fields, and handles the other bits as a sub-opcode field.

CAUTION

Extended instruction format 1 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in Section 7.2.2, Basic Instruction Set.



(10) Extended instruction format 2 (Format X)

This is a 32-bit instruction format that has a 6-bit opcode field and uses the other bits as a subopcode field.

CAUTION

Extended instruction format 2 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in Section 7.2.2, Basic Instruction Set.



(11) Extended instruction format 3 (Format XI)

This is a 32-bit instruction format that has a 6-bit opcode field and three general-purpose register specification fields, and uses the other bits as a sub-opcode field.

CAUTION

Extended instruction format 3 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in Section 7.2.2, Basic Instruction Set.



(12) Extended instruction format 4 (Format XII)

This is a 32-bit instruction format that has a 6-bit opcode field and two general-purpose register specification fields, and uses the other bits as a sub-opcode field.

CAUTION

Extended instruction format 4 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in Section 7.2.2, Basic Instruction Set.





(13) Stack manipulation instruction format (Format XIII)

A 32-bit instruction format consists of a 5-bit opcode field, 5-bit immediate field, 12-bit register list field, 5-bit sub-opcode field, and one general-purpose register specification field (or 5-bit sub-opcode field).

The general-purpose register specification field is used as a sub-opcode filed, depending on the format of the instruction.

15	11	10		6	5		1	0	31					21	20		16
				1													1
sub-opc	ode		opcode			imm					list					reg2	

(14) Load/store instruction 48-bit format (Format XIV)

This is a 48-bit instruction format that has a 6-bit opcode field, two general-purpose register specification fields, and a 23-bit displacement field, and uses the other bits as a sub-opcode field.




7.1.2 Coprocessor Instructions

Instructions in the following format are defined as coprocessor instructions.



Coprocessor instructions define the functions of each coprocessor.

(1) Coprocessor unusable exception

If an attempt is made to execute a coprocessor instruction defined by an opcode that refers to a nonexistent coprocessor or a coprocessor that cannot be used due to the operational status of the device, a coprocessor unusable exception (UCPOP) immediately occurs.

For details, see Section 2.4.3, Coprocessor Unusable Exceptions.

7.1.3 Reserved Instructions

An opcode reserved for future function extension and for which no instruction is defined is defined as a reserved instruction. It is defined by the hardware specifications that either of the following two types of operations is performed on the opcode of a reserved instruction.

- A reserved instruction exception occurs
- The reserved instruction is executed as an instruction

In this CPU, the following opcodes define the RIE instruction, which always causes a reserved instruction exception to occur.



7.2 Basic Instructions

7.2.1 Overview of Basic Instructions

(1) Load instructions

Execute data transfer from memory to register. The following instructions (mnemonics) are provided.

- (a) LD instructions
- LD.B : Load byte
- LD.BU : Load byte unsigned
- LD.DW : Load double word
- LD.H : Load halfword
- LD.HU : Load halfword unsigned
- LD.W : Load word
- (b) SLD instructions
- SLD.B : Short format load byte
- SLD.BU : Short format load byte unsigned
- SLD.H : Short format load halfword
- SLD.HU : Short format load halfword unsigned
- SLD.W : Short format load word

(2) Store instructions

Execute data transfer from register to memory. The following instructions (mnemonics) are provided.

- (a) ST instructions
 - ST.B : Store byte
 - ST.DW : Store double word
 - ST.H : Store halfword
- ST.W : Store word
- (b) SST instructions
- SST.B : Short format store byte
- SST.H : Short format store halfword
- SST.W : Short format store word

(3) Multiply instructions

Execute multiplication in one clock cycle with the on-chip hardware multiplier. The following instructions (mnemonics) are provided.

- MUL : Multiply word
- MULH : Multiply halfword
- MULHI : Multiply halfword immediate
- MULU : Multiply word unsigned

(4) Multiply-accumulate instructions

After a multiplication operation, a value is added to the result. The following instructions (mnemonics) are available.

- MAC : Multiply and add word
- MACU : Multiply and add word unsigned

(5) Arithmetic instructions

Add, subtract, transfer, or compare data between registers. The following instructions (mnemonics) are provided.

- ADD : Add
- ADDI : Add immediate
- CMP : Compare
- MOV : Move
- MOVEA : Move effective address
- MOVHI : Move high halfword
- SUB : Subtract
- SUBR : Subtract reverse

(6) Conditional arithmetic instructions

Add and subtract operations are performed under specified conditions. The following instructions (mnemonics) are available.

- ADF : Add on condition flag
- SBF : Subtract on condition flag

(7) Saturated operation instructions

Execute saturated addition and subtraction. If the operation result exceeds the maximum positive value (7FFF FFFF_{H}), 7FFF FFFF_{H} returns. If the operation result exceeds the maximum negative value (8000 0000_{H}), 8000 0000_{H} returns. The following instructions (mnemonics) are provided.

- SATADD : Saturated add
- SATSUB : Saturated subtract
- SATSUBI : Saturated subtract immediate
- SATSUBR : Saturated subtract reverse

(8) Logical instructions

Include logical operation instructions. The following instructions (mnemonics) are provided.

- AND : AND
- ANDI : AND immediate
- NOT : NOT
- OR : OR

- ORI : OR immediate
- TST : Test
- XOR : Exclusive OR
- XORI : Exclusive OR immediate

(9) Data manipulation instructions

Include data manipulation instructions and shift instructions with arithmetic shift and logical shift. Operands can be shifted by multiple bits in one clock cycle through the on-chip barrel shifter. The following instructions (mnemonics) are provided.

- BINS : Bitfield insert
- BSH : Byte swap halfword
- BSW : Byte swap word
- CMOV : Conditional move
- HSH : Halfword swap halfword
- HSW : Halfword swap word
- ROTL : Rotate left
- SAR : Shift arithmetic right
- SASF : Shift and set flag condition
- SETF : Set flag condition
- SHL : Shift logical left
- SHR : Shift logical right
- SXB : Sign-extend byte
- SXH : Sign-extend halfword
- ZXB : Zero-extend byte
- ZXH : Zero-extend halfword

(10) Bit search instructions

The specified bit values are searched among data stored in registers.

- SCH0L : Search zero from left
- SCH0R : Search zero from right
- SCH1L : Search one from left
- SCH1R : Search one from right

(11) Divide instructions

Execute division operations. Regardless of values stored in a register, the operation can be performed using a constant number of steps. The following instructions (mnemonics) are provided.

- DIV : Divide word
- DIVH : Divide halfword



- DIVHU : Divide halfword unsigned
- DIVU : Divide word unsigned

(12) High-speed divide instructions

These instructions perform division operations. The number of valid digits in the quotient is determined in advanced from values stored in a register, so the operation can be performed using a minimum number of steps. The following instructions (mnemonics) are provided.

- DIVQ : Divide word quickly
- DIVQU : Divide word unsigned quickly

(13) Branch instructions

Include unconditional branch instructions (JARL, JMP, and JR) and a conditional branch instruction (Bcond) which accommodates the flag status to switch controls. Program control can be transferred to the address specified by a branch instruction. The following instructions (mnemonics) are provided.

- Bcond (BC, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BR, BSA, BV, BZ) : Branch on condition code
- JARL : Jump and register link
- JMP : Jump register
- JR : Jump relative

(14) Loop instruction

• LOOP : Loop

(15) Bit manipulation instructions

Execute logical operation on memory bit data. Only a specified bit is affected. The following instructions (mnemonics) are provided.

- CLR1 : Clear bit
- NOT1 : Not bit
- SET1 : Set bit
- TST1 : Test bit

(16) Special instructions

Include instructions not provided in the categories of instructions described above. The following instructions (mnemonics) are provided.

- CALLT : Call with table look up
- CAXI : Compare and exchange for interlock
- CLL : Clear load link
- CTRET : Return from CALLT
- DI : Disable interrupt
- DISPOSE : Function dispose
- EI : Enable interrupt

- EIRET : Return from trap or interrupt
- FERET : Return from trap or interrupt
- FETRAP : Software trap
- HALT : Halt
- LDSR : Load system register
- LDL.W : Load linked word
- NOP : No operation
- POPSP : Pop registers from stack
- PREPARE : Function prepare
- PUSHSP : Push registers from stack
- RIE : Reserved instruction exception
- SNOOZE : Snooze
- STSR : Store system register
- STC.W : Store conditional word
- SWITCH : Jump with table look up
- SYNCE : Synchronize exceptions
- SYNCI : Synchronize memory for instruction writers
- SYNCM : Synchronize memory
- SYNCP : Synchronize pipeline
- SYSCALL : System call
- TRAP : Trap



7.2.2 Basic Instruction Set

This section details each instruction, dividing each mnemonic (in alphabetical order) into the following items.

- Instruction format : Indicates how the instruction is written and its operand(s) (for symbols, see **Table 7.1**).
- Operation : Indicates the function of the instruction (for symbols, see Table 7.2).
- Format : Indicates the instruction format (see Section 7.1, Opcodes and Instruction Formats).
- Opcode : Indicates the bit field of the instruction opcode (for symbols, see Table 7.3).
- Flag : Indicates the change of flags of PSW (program status word) after the instruction execution. "0" is to clear (reset), "1" to set, and "—" to remain unchanged.
- Description : Describes the operation of the instruction.
- Supplement : Provides supplementary information on the instruction.
- Caution : Provides precautionary notes.

Table 7.1 Conventions of Instruction Format

Symbol	Meaning				
reg1	General-purpose register (as source register)				
reg2	General-purpose register (primarily as destination register with some as source registers)				
reg3	General-purpose register (primarily used to store the remainder of a division result and/or the higher 32 bits of a multiplication result)				
bit#3	3-bit data to specify bit number				
imm ×	x-bit immediate data				
disp ×	x-bit displacement data				
regID	System register number				
sellD	System register group number				
vector ×	Data to specify vector (x indicates the bit size)				
cond	Condition code (see Table 7.4 Condition Codes)				
CCCC	4-bit data to specify condition code (see Table 7.4 Condition Codes)				
sp	Stack pointer (r3)				
ер	Element pointer (r30)				
list12	Lists of registers				
rh-rt	Indicates multiple general-purpose registers, from the general-purpose register indicated by <i>rh</i> to the general-purpose register indicated by <i>rt</i> .				



Symbol	Meaning
←	Assignment
GR [a]	Value stored in general-purpose register a
SR [a, b]	Value stored in system register (RegID = a, SeIID = b)
(n:m)	Bit selection. Select from bit <i>n</i> to bit <i>m</i> .
zero-extend (n)	Zero-extends "n" to word
sign-extend (n)	Sign-extends "n" to word
load-memory (a, b)	Reads data of size b from address a
store-memory (a, b, c)	Writes data b of size c to address a
extract-bit (a, b)	Extracts value of bit b of data a
set-bit (a, b)	Sets value of bit b of data a
not-bit (a, b)	Inverts value of bit b of data a
clear-bit (a, b)	Clears value of bit b of data a
saturated (n)	Performs saturated processing of "n." If $n \ge 7FFFFFF_H$, $n = 7FFFFFF_H$. If $n \le 8000\ 0000_H$, $n = 8000\ 0000_H$.
result	Outputs results on flag
Byte	Byte (8 bits)
Halfword	Halfword (16 bits)
Word	Word (32 bits)
==	Comparison (true upon a match)
!=	Comparison (true upon a mismatch)
+	Add
-	Subtract
ll	Bit concatenation
×	Multiply
÷	Divide
%	Remainder of division results
AND	AND
OR	OR
XOR	Exclusive OR
NOT	Logical negate
logically shift left by	Logical left-shift
logically shift right by	Logical right-shift
arithmetically shift right by	Arithmetic right-shift

Table 7.2Conventions of Operation



Symbol	Meaning			
R	1-bit data of code specifying reg1 or regID			
r	1-bit data of code specifying reg2			
w	1-bit data of code specifying reg3			
D	1-bit data of displacement (indicates higher bits of displacement)			
d	1-bit data of displacement			
I	1-bit data of immediate (indicates higher bits of immediate)			
i	1-bit data of immediate			
V	1-bit data of code specifying vector (indicates higher bits of vector)			
v	1-bit data of code specifying vector			
CCCC	4-bit data for condition code specification (See Table 7.4 Condition Codes)			
bbb	3-bit data for bit number specification			
L	1-bit data of code specifying general-purpose register in register list			
S	1-bit data of code specifying EIPC/FEPC, EIPSW/FEPSW in register list			
Р	1-bit data of code specifying PSW in register list			

Table 7.3 Conventions of Opcode

Table 7.4Condition Codes

Condition Code (cccc)	Condition Name	Condition Formula
0000	V	OV = 1
1000	NV	OV = 0
0001	C/L	CY = 1
1001	NC/NL	CY = 0
0010	Z	Z = 1
1010	NZ	Z = 0
0011	NH	(CY or Z) = 1
1011	Н	(CY or Z) = 0
0100	S/N	S = 1
1100	NS/P	S = 0
0101	Т	Always (Unconditional)
1101	SA	SAT = 1
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0

<Arithmetic instruction>

ADD	
-----	--

Add register/immediate

Add

[Instruction format]	 (1) ADD re (2) ADD in 	
[Operation]		$2] \leftarrow GR [reg2] + GR [reg1]$ $2] \leftarrow GR [reg2] + sign-extend (imm5)$
[Format]	 Format I Format I 	
[Opcode]	15	0
	15	0 0 010010iiii
[Flags]	CY OV S Z SAT	 "1" if a carry occurs from MSB; otherwise, "0". "1" if overflow occurs; otherwise, "0". "1" if the operation result is negative; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]	register 1	word data of general-purpose register reg1 to the word data of general-purpose reg2 and stores the result in general-purpose register reg2. General-purpose reg1 is not affected.
		5-bit immediate data, sign-extended to word length, to the word data of purpose register reg2 and stores the result in general-purpose register reg2.

<Arithmetic instruction>

		Add immediate
		Add immediate
[Instruction format]	ADDI im	nm16, reg1, reg2
[Operation]	GR [reg2]	$] \leftarrow GR [reg1] + sign-extend (imm16)$
[Format]	Format V	Ι
[Opcode]	15 rrrr11	031 16 LOOOORRRRR iiiiiiiiiiiiiii
[Flags]	CY OV S Z SAT	 "1" if a carry occurs from MSB; otherwise, "0". "1" if overflow occurs; otherwise, "0". "1" if the operation result is negative; otherwise, "0". "1" if the operation result is "0"; otherwise "0".
[Description]	purpose re	16-bit immediate data, sign-extended to word length, to the word data of general- egister reg1 and stores the result in general-purpose register reg2. General-purpose eg1 is not affected.



<Conditional Operation Instructions>

ADF						Add on condition flag
						Conditional add
[Instruction format]	ADF cccc	c, reg1, reg2	, reg3			
[Operation]	then GR [•	ied [reg1] + GR [reg2] +1 [reg1] + GR [reg2] +0			
[Format]	Format X	I				
[Opcode]	15 rrrrr11	L1111RRRR	031 R wwww011101ccc	16 c0		
[Flags]						
	CY		arry occurs from MSB; oth			
	OV		erflow occurs; otherwise, "0 e operation result is negativ		٠ ٠ ٣	
	s z		e operation result is "0"; oth		0.	
	SAT	_				
[Description]	of general reg3, if the If the cond purpose re is stored in General-p	-purpose reg e condition s dition specif egister reg1 i n general-pu urpose regis	gister reg2 and stores the specified as condition co ied as condition code "c	e result of ad ode "cccc" is ccc" is not s of general-p ot affected. I	dition in ge satisfied. atisfied, the purpose reg Designate o	e word data of general- ister reg2, and the result ne of the condition
	Condition Code	n Name	Condition Formula	Condition Code	Name	Condition Formula
	0000	V		0100	S/N	S = 1

Coue	Name	Condition i ornidia	Coue	Name	Condition Formula
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	Т	Always (Unconditional)
1001	NC/NL	CY = 0	0110	LT	(S xor OV) = 1
0010	Z	Z = 1	1110	GE	(S xor OV) = 0
1010	NZ	Z = 0	0111	LE	((S xor OV) or Z) = 1
0011	NH	(CY or Z) = 1	1111	GT	((S xor OV) or Z) = 0
1011	Н	(CY or Z) = 0	(1101)	Setting pr	ohibited



<Logical instruction>

AND		AND
		AND
[Instruction format]	AND reg	1, reg2
[Operation]	GR [reg2]	← GR [reg2] AND GR [reg1]
[Format]	Format I	
[Opcode]	15 rrrr00	0 1010RRRR
[Flags]	CY OV S Z SAT	 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]		word data of general-purpose register reg2 with the word data of general-purpose g1 and stores the result in general-purpose register reg2. General-purpose register t affected.



<Logical instruction>

ANDI		AND immediate
		AND immediate
[Instruction format]	ANDI im	um16, reg1, reg2
[Operation]	GR [reg2]	\leftarrow GR [reg1] AND zero-extend (imm16)
[Format]	Format V	Ι
[Opcode]	15 rrrrr11	031 16 LOIIORRRRR IIIIIIIIIIIII
[Flags]	CY OV S Z SAT	— 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0". —
[Description]	extended	word data of general-purpose register reg1 with the 16-bit immediate data, zero- to word length, and stores the result in general-purpose register reg2. General- egister reg1 is not affected.



<Branch instruction>

Bcond		Branch on condition code with 9-bit displacemen
		Conditional branch
[Instruction format]	(1) Bc	ond disp9
	(2) Bco	ond disp17
[Operation]	(1) if c	onditions are satisfied
	the	$n PC \leftarrow PC + sign-extend (disp9)$
	(2) if c	onditions are satisfied
	the	$n PC \leftarrow PC + sign-extend (disp17)$
[Format]	(1) For	mat III
	(2) For	mat VII
[Opcode]	15	0
	(1) d	ddd1011dddcccc
	ddddd	ldd is the higher 8 bits of disp9.
		the condition code of the condition indicated by cond (see Table 7.5, Bcond
	15	<u> </u>
	(2) 0	0000111111DCCCC dddddddddddddd
		addadadad is the higher 16 bits of disp17.
		the condition code of the condition indicated by cond. (For details, see Table 7.5 , Instructions).
[Flags]	CY	_
	OV	_
	S	—
	Z	_
	SAT	_



[Description]

(1) Checks each PSW flag specified by the instruction and branches if a condition is met; otherwise, executes the next instruction. The PC of branch destination is the sum of the current PC value and the 9-bit displacement (= 8-bit immediate data shifted by 1 and sign-extended to word length).

(2) Checks each PSW flag specified by the instruction and then adds the result of logically shifting the 16-bit immediate data 1 bit to the left and sign-extending it to word length to the current PC value if the conditions are satisfied. Control is then transferred. If the conditions are not satisfied, the system continues to the next instruction. BR (0101) cannot be specified as the condition code.

[Supplement] Bit 0 of the 9-bit displacement is masked to "0". The current PC value used for calculation is the address of the first byte of this instruction. The displacement value being "0" signifies that the branch destination is the instruction itself.

Ins	truction	Condition Code (cccc)	Flag Status	Branch Condition
Signed	BGE	1110	(S xor OV) = 0	Greater than or equal to signed
integer	BGT	1111	((S xor OV) or Z) = 0	Greater than signed
	BLE	0111	((S xor OV) or Z) = 1	Less than or equal to signed
	BLT	0110	(S xor OV) = 1	Less than signed
Unsigned	BH	1011	(CY or Z) = 0	Higher (Greater than)
integer	BL	0001	CY = 1	Lower (Less than)
	BNH	0011	(CY or Z) = 1	Not higher (Less than or equal)
	BNL	1001	CY = 0	Not lower (Greater than or equal)
Common	BE	0010	Z = 1	Equal
	BNE	1010	Z = 0	Not equal
Others	BC	0001	CY = 1	Carry
	BF	1010	Z = 0	False
	BN	0100	S = 1	Negative
	BNC	1001	CY = 0	No carry
	BNV	1000	OV = 0	No overflow
	BNZ	1010	Z = 0	Not zero
	BP	1100	S = 0	Positive
	BR	0101	_	Always (unconditional) Cannot be specified when using instruction format (2).
	BSA	1101	SAT = 1	Saturated
	BT	0010	Z = 1	True
	BV	0000	OV = 1	Overflow
	BZ	0010	Z = 1	Zero

Table 7.5Bcond Instructions

CAUTIONS

- The branch condition loses its meaning if a conditional branch instruction is executed on a signed integer (BGE, BGT, BLE, or BLT) when the saturated operation instruction sets "1" to the SAT flag. In normal operations, if an overflow occurs, the S flag is inverted (0 → 1 or 1 → 0). This is because the result is a negative value if it exceeds the maximum positive value and it is a positive value if it exceeds the maximum negative value. However, when a saturated operation instruction is executed, and if the result exceeds the maximum positive value, the result is saturated with a positive value; if the result exceeds the maximum negative value, the result is not inverted even if an overflow occurs.
- 2. For Bcond disp17 (instruction format (2)), BR (0101) cannot be specified as the condition code.



	Bitfield Insert
BINS	Insert bit in register
[Instruction format]	BINS reg1, pos, width, reg2
[Operation]	$GR [reg2] \leftarrow GR [reg2](31:width+pos) \parallel GR [reg1](width-1:0) \parallel GR [reg2](pos-1:0)$
[Format]	Format IX
[Opcode]	$15 031 16$ $rrrr111111RRRR MMMK0001001LLL0 (msb \ge 16, lsb \ge 16)$ $15 031 16$ $rrrr111111RRRR MMMK0001011LLL0 (msb \ge 16, lsb < 16)$ $15 031 16$ $rrrr111111RRRR MMMK0001101LLL0 (msb < 16, lsb < 16)$ Most significant bit of field to be updated: msb = pos+width-1 Least significant bit of field to be updated: msb = pos+width-1 MMMM = lower 4 bits of msb, KLLL = lower 4 bits of lsb
[Flags]	CY—OV0S"1" if operation result word data MSB is "1"; otherwise, "0".Z"1" if operation result is "0"; otherwise, "0".SAT—
[Description]	Loads the lower width bits in general-purpose register reg1 and stores them from the bit position bit pos + width -1 in the specified field in general-purpose register reg2 in bit pos. This instruction does not affect any fields in general-purpose register reg2 except the specified field, nor does it affect general-purpose register reg1.
[Supplement]	The most significant bit (msb: bit pos + width – 1) in the field in general-purpose register reg2 to be updated and the least significant bit (lsb: bit pos) in this field are specified by using, respectively the lower 4 bits, the MMMM and KLLL fields in the BINS instruction. The lower 3 bits of the sub-opcode field (bits 23 to 21) differ depending on the msb and lsb values. The operation is undefined if msb < lsb.

RENESAS

BSH		Byte swap halfword
		Byte swap of halfword data
[Instruction format]	BSH reg	2, reg3
[Operation]	GR [reg3]] \leftarrow GR [reg2] (23:16) GR [reg2] (31:24) GR [reg2] (7:0) GR [reg2] (15:8)
[Format]	Format X	Π
[Opcode]	15 rrrr11	031 16 1111100000 wwww01101000010
[Flags]	CY	"1" when there is at least one byte value of zero in the lower halfword of the operation
	OV	result; otherwise; "0". 0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" when lower halfword of operation result is "0"; otherwise, "0".
	SAT	_
[Description]	Executes	endian swap.



BSW		Byte swap word
_		Byte swap of word data
[Instruction format]	BSW reg	2, reg3
[Operation]	GR [reg3]	$] \leftarrow GR [reg2] (7:0) \parallel GR [reg2] (15:8) \parallel GR [reg2] (23:16) \parallel GR [reg2] (31:24)$
[Format]	Format X	ΙΙ
[Opcode]	15 rrrr11	031 16 L111100000 wwww01101000000
[Flags]	CY OV S Z SAT	 "1" when there is at least one byte value of zero in the word data of the operation result; otherwise; "0". 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if operation result word data is "0"; otherwise, "0".
[Description]	Executes	endian swap.



CALLT	Call with table look up
	Subroutine call with table look up
[Instruction format]	CALLT imm6
[Operation]	$CTPC \leftarrow PC + 2 \text{ (return PC)}$ $CTPSW(4:0) \leftarrow PSW(4:0)$ adr $\leftarrow CTBP + \text{zero-extend (imm6 logically shift left by 1)}^{*1}$ $PC \leftarrow CTBP + \text{zero-extend (Load-memory (adr, Half-word))}$
	Caution 1. An MDP exception might occur depending on the result of address calculation.
[Format]	Format II
[Opcode]	15 0 000001000iiiii
[Flags]	CY — OV — S — Z — SAT —
[Description]	The following steps are taken.
	(1) Transfers the contents of both return PC and PSW to CTPC and CTPSW.
	(2) Adds the CTBP value to the 6-bit immediate data, logically left-shifted by 1, and zero- extended to word length, to generate a 32-bit table entry address.
	(3) Loads the halfword entry data of the address generated in step (2) and zero-extend to word length.
	(4) Adds the CTBP value to the data generated in step (3) to generate a 32-bit target address.
	(5) Jumps to the target address.



CAUTIONS

- 1. When an exception occurs during CALLT instruction execution, the execution is aborted after the end of the read/write cycle.
- 2. Memory protection is performed when executing a memory read operation to read the CALLT instruction table. When memory protection is enabled, the data for generating a target address from a table allocated in an area to which access from a user program is prohibited cannot be loaded
- 3. Be sure not to set the target address of the CALLT instruction less than 0000 $0800_{\rm H}$ nor to 0020 $0020_{\rm H}.$



Compare and exchange for interlock CAXI Comparison and swap

[Instruction format]	CAXI [reg1], reg2, reg3		
[Operation]	$\begin{aligned} & \operatorname{adr} \leftarrow \operatorname{GR}[\operatorname{reg}1]^{*1} \\ & \operatorname{token} \leftarrow \operatorname{Load-memory} (\operatorname{adr}, \operatorname{Word}) \\ & \operatorname{result} \leftarrow \operatorname{GR}[\operatorname{reg}2] - \operatorname{token} \\ & \operatorname{If} \operatorname{result} == 0 \\ & \operatorname{then} \operatorname{Store-memory} (\operatorname{adr}, \operatorname{GR}[\operatorname{reg}3], \operatorname{Word}) \\ & \operatorname{GR}[\operatorname{reg}3] \leftarrow \operatorname{token} \\ & \operatorname{else} \operatorname{Store-memory} (\operatorname{adr}, \operatorname{token}, \operatorname{Word}) \\ & \operatorname{GR}[\operatorname{reg}3] \leftarrow \operatorname{token} \\ & \operatorname{GR}[\operatorname{reg}3] \leftarrow \operatorname{token} \\ & \operatorname{Caution 1.} \operatorname{An} \operatorname{MAE}, \operatorname{or} \operatorname{MDP} \operatorname{exception} \operatorname{might} \operatorname{occur} \operatorname{depending} \operatorname{on} \operatorname{the} \operatorname{result} \operatorname{of} \operatorname{address} \operatorname{calculation.} \end{aligned}$		
[Format]	Format XI		
[Opcode]	15 031 16 rrrr111111RRRRR wwww00011101110		
[Flags]	CY"1" if a borrow occurs in the result operation; otherwise, "0"OV"1" if overflow occurs in the result operation; otherwise, "0"S"1" if result is negative; otherwise, "0"Z"1" if result is 0; otherwise, "0"SAT—		
[Description]	Word data is read from the specified address and compared with the word data in general- purpose register reg2, and the result is indicated by flags in the PSW. Comparison is performed by subtracting the read word data from the word data in general-purpose register reg2. If the comparison result is "0", word data in general-purpose register reg3 is stored in the generated address, otherwise the read word data is stored in the generated address. Afterward, the read word data is stored in general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.		



CAUTIONS

- 1. This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.
- 2. The CAXI instruction is included for backward compatibility. If you are using a multi-core system and require an atomic guarantee, use the LDL.W and STC.W instructions.



CLL	Clear Load Link
	Clear atomic manipulation link
[Instruction format]	CLL
[Operation]	Llbit $\leftarrow 0$
[Format]	Format X
[Opcode]	15 031 16 1111111111111111111111111111111111
[Flags]	CY – OV – S – Z –
[Description]	SAT — The thread link generated by the LDL.W instruction is deleted.
	For details about the link operation between the thread and core, see Section 5.3.2, Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions.
	CAUTION

In systems such as a multi-core system, how the CLL instruction operates depends on the system configuration of the product. For details, see the hardware manual of the product used.



	<u></u>	Clear bit
CLR1		Bit clear
[Instruction format]	(1) CLR1 bit#3, disp16 [reg1]	
	(2) CLR1 reg2, [reg1]	
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*¹ token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, bit#3)) token ← clear-bit (token, bit#3) Store-memory (adr, token, Byte) 	
	 (2) adr ← GR [reg1]*¹ token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, reg2)) token ← clear-bit (token, reg2) Store-memory (adr, token, Byte) 	
	Note 1. An MDP exception might occur depending on the result of address of	calculation.
[Format]	 (1) Format VIII (2) Format IX 	
[Opcode]	15 0 31 16	
	(1) 10bbb111110RRRRR ddddddddddddddddd	
	(2) 15 0 31 16 rrrr111111RRRRR 000000011100100	
[Flags]	CY — OV — S — Z "1" if bit specified by operand = "0", "0" if bit specified by operand =	= "1".
		= "1".



[Description]	(1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, then the bits indicated by the 3-bit bit number are cleared (0) and the data is written back to the original address.		
	(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the bits indicated by the lower three bits of reg2 are cleared (0), and the data is written back to the original address.		
[Supplement]	The Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is executed, and does not indicate the content of the specified bit after this instruction is executed.		
	CAUTION		
	This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any		

other cause.



смоу		Conditional move
		Conditional move
[Instruction format]	(1) CMOV cccc, reg1, reg2, reg3	
	(2) CMOV cccc, imm5, reg2, reg3	
[Operation]	 (1) if conditions are satisfied then GR [reg3] ← GR [reg1] else GR [reg3] ← GR [reg2] 	
	 (2) if conditions are satisfied then GR [reg3] ← sign-extended (imm5) else GR [reg3] ← GR [reg2] 	
[Format]	(1) Format XI	
	(2) Format XII	
[Opcode]		
	15 0 31 16 (1) rrrrr111111RRRRR wwwww011001cccc0	
	(2) 15 031 16 rrrr111111iiii wwww011000cccc0	
[Flags]	СҮ —	
	OV – S –	
	Z —	
	SAT —	



[Description]

(1) When the condition specified by condition code "cccc" is met, data in general-purpose register reg1 is transferred to general-purpose register reg3. When that condition is not met, data in general-purpose register reg2 is transferred to general-purpose register reg3. Specify one of the condition codes shown in the following table as "cccc".

Condition			Condition		
Code	Name	Condition Formula	Code	Name	Condition Formula
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	Т	Always (unconditional)
1001	NC/NL	CY = 0	1101	SA	SAT = 1
0010	Z	Z = 1	0110	LT	(S xor OV) = 1
1010	NZ	Z = 0	1110	GE	(S xor OV) = 0
0011	NH	(CY or Z) = 1	0111	LE	((S xor OV) or Z) = 1
1011	Н	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0

(2) When the condition specified by condition code "cccc" is met, 5-bit immediate data signextended to word-length is transferred to general-purpose register reg3. When that condition is not met, the data in general-purpose register reg2 is transferred to generalpurpose register reg3. Specify one of the condition codes shown in the following table as "cccc".

Condition			Condition		
Code	Name	Condition Formula	Code	Name	Condition Formula
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	Т	Always (unconditional)
1001	NC/NL	CY = 0	1101	SA	SAT = 1
0010	Z	Z = 1	0110	LT	(S xor OV) = 1
1010	NZ	Z = 0	1110	GE	(S xor OV) = 0
0011	NH	(CY or Z) = 1	0111	LE	((S xor OV) or Z) = 1
1011	Н	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0

[Supplement]

See the description of the SETF instruction.



<arithmetic instruction=""></arithmetic>	Compare register/immediate (5-bit
CMP	
	Compare
[Instruction format]	(1) CMP reg1, reg2
	(2) CMP imm5, reg2
[Operation]	(1) result \leftarrow GR [reg2] – GR [reg1]
	(2) result \leftarrow GR [reg2] – sign-extend (imm5)
[Format]	(1) Format I
	(2) Format II
[Opcode]	
	15 0 (1) rrrrr001111RRRRR
	(2) 15 0 rrrr010011iiiii
[Flags]	
	CY "1" if a borrow occurs from MSB; otherwise, "0".
	OV "1" if overflow occurs; otherwise, "0". ************************************
	 S "1" if the operation result is negative; otherwise, "0". Z "1" if the operation result is "0"; otherwise, "0".
	SAT —
[Description]	(1) Compares the word data of general-purpose register reg2 with the word data of general- purpose register reg1 and outputs the result through the PSW flags. Comparison is performed by subtracting the reg1 contents from the reg2 word data. General-purpose registers reg1 and reg2 are not affected.
	(2) Compares the word data of general-purpose register reg2 with the 5-bit immediate data, sign-extended to word length, and outputs the result through the PSW flags. Comparison is performed by subtracting the sign-extended immediate data from the reg2 word data. General-purpose register reg2 is not affected.



CTRET	Return from CALLT
	Return from subroutine call
[Instruction format]	CTRET
[Operation]	$PC \leftarrow CTPC$ $PSW(4:0) \leftarrow CTPSW(4:0)$
[Format]	Format X
[Opcode]	15 031 16 0000011111100000 000000101000100
[Flags]	CYValue read from CTPSW is set.OVValue read from CTPSW is set.SValue read from CTPSW is set.ZValue read from CTPSW is set.SATValue read from CTPSW is set.
[Description]	 Loads the return PC and PSW (the lower 5 bits) from the appropriate system register and returns from a routine under CALLT instruction. The following steps are taken: (1) The return PC and the return PSW (the lower 5 bits) are loaded from the CTPC and CTPSW. (2) The values are restored in PC and PSW (the lower 5 bits) and the control is transferred to the return address.
	CAUTION



<special instruction=""></special>	
	Disable interrupt
וט	
	Disable EI level maskable exception
[Instruction format]	DI
[Operation]	PSW.ID $\leftarrow 1$ (Disables EI level maskable interrupt)
[Format]	Format X
[Opcode]	15 031 16 0000011111100000 000000101100000
[Flags]	CY — OV — S — Z — SAT — ID 1
[Description]	Sets "1" to the ID flag of the PSW to disable the acknowledgement of EI level maskable exceptions after the execution of this instruction.
[Supplement]	Overwrite of flags in the PSW by this instruction becomes valid as of the next instruction. If the MCTL.UIC bit has been cleared to 0, this instruction is a supervisor-level instruction. If the MCTL.UIC bit has been set to 1, this instruction can always be executed.



DISPOSE		Function dispose
		Stack frame deletion
[Instruction format]	(1) DISPOSE imm5, list12	
	(2) DISPOSE imm5, list12, [reg1]	
[Operation]	(1) tmp \leftarrow sp + zero-extend (imm5 logically shift by 2)	
	foreach (all regs in list12) { adr \leftarrow tmp ^{*1, *2}	
	$GR[reg in list12] \leftarrow Load-memory (adr, Word)$)
	$tmp \leftarrow tmp + 4$	
	}	
	$sp \leftarrow tmp$	
	(2) tmp \leftarrow sp + zero-extend (imm5 logically shift by 2)	
	foreach (all regs in list12) { adr \leftarrow tmp* ^{1, *2}	
	GR[reg in list12] ← Load-memory (adr, Word))
	$tmp \leftarrow tmp + 4$	
	}	

 $PC \leftarrow GR[reg1]$

 $sp \leftarrow tmp$

Note 1. An MDP exception might occur depending on the result of address calculation.

Note 2. When loading to memory, the lower 2 bits of adr are masked to 0.

[Format]

Format XIII

[Opcode]

	15 0	31 16
(1)	0000011001iiiiL	LLLLLLLLL00000
	15 0	31 16
(2)	0000011001iiiiL	LLLLLLLLLRRRRR

RRRRR \neq 00000 (Do not specify r0 for reg1.)

The values of LLLLLLLLLL are the corresponding bit values shown in register list "list12" (for example, the "L" at bit 21 of the opcode corresponds to the value of bit21 in list12). list12 is a 32-bit register list, defined as follows.

RENESAS

31	30	29	28	27	26	25	24	23	22	21	20 1	0	_
r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31	_	r30	

Bits 31 to 21 and bit 0 correspond to general-purpose registers (r20 to r31), so that when any of these bits is set (1), it specifies a corresponding register operation as a processing target. For example, when r20 and r30 are specified, the values in list12 appear as shown below (register bits that do not correspond, i.e., bits 20 to 1 are set as "Don't care").

- When all of the register's non-corresponding bits are "0": $0800\ 0001_{\rm H}$
- When all of the register's non-corresponding bits are "1": $081F FFFF_H$

[Flags]

CY	—
OV	—
S	—
Z	_
SAT	_

[Description]

- (1) Adds the 5-bit immediate data, logically left-shifted by 2 and zero-extended to word length, to sp; returns to general-purpose registers listed in list12 by loading the data from the address specified by sp and adds 4 to sp.
- (2) Adds the 5-bit immediate data, logically left-shifted by 2 and zero-extended to word length, to sp; returns to general-purpose registers listed in list12 by loading the data from the address specified by sp and adds 4 to sp; and transfers the control to the address specified by general-purpose register reg1.

[Supplement] General-purpose registers in list12 are loaded in descending order (r31, r30, ... r20). The imm5 restores a stack frame for automatic variables and temporary data. The lower 2 bits of the address specified by sp is always masked to "0" and aligned to the word boundary.

CAUTIONS

- 1. If an exception occurs while this instruction is being executed, execution of the instruction might be stopped after the read/write cycle and the register value write operation are completed, but sp will retain its original value from before the start of execution. The instruction will be executed again later, after a return from the exception.
- 2. For instruction format (2) DISPOSE imm5, list12, [reg1], do not specify r0 for reg1.

<Divide instruction>

DIV	Divide word
	Division of (signed) word data
[Instruction format]	DIV reg1, reg2, reg3
[Operation]	$GR [reg2] \leftarrow GR [reg2] \div GR [reg1]$ $GR [reg3] \leftarrow GR [reg2] \% GR [reg1]$
[Format]	Format XI
[Opcode]	15 031 16 rrrr111111RRRRR wwww01011000000
[Flags]	CY—OV"1" if overflow occurs; otherwise, "0"S"1" if the operation result quotient is negative; otherwise, "0".Z"1" if the operation result quotient is "0"; otherwise, "0".SAT—
[Description]	Divides the word data of general-purpose register reg2 by the word data of general-purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.
[Supplement]	Overflow occurs when the maximum negative value $(8000\ 0000_{\rm H})$ is divided by -1 with the quotient = $8000\ 0000_{\rm H}$ and when the data is divided by 0 with quotient being undefined. If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during the DIV instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.
	CAUTION



operation result quotient is not stored in reg2, so the flag is undefined.

<Divide instruction>

DIVH	Divide halfword
	Division of (signed) halfword data
[Instruction format]	(1) DIVH reg1, reg2
	(2) DIVH reg1, reg2, reg3
[Operation]	(1) GR [reg2] \leftarrow GR [reg2] \div sign-extend (GR [reg1] (15:0))
	(2) GR [reg2] \leftarrow GR [reg2] \div sign-extend (GR [reg1] (15:0)) GR [reg3] \leftarrow GR [reg2] % sign-extend (GR [reg1] (15:0))
[Format]	(1) Format I
	(2) Format XI
[Opcode]	
	15 0 (1) rrrr000010RRRR
	RRRRR \neq 00000 (Do not specify r0 for reg1.)
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
	(2) 15 0 31 16 rrrr111111RRRRR wwww0101000000
[Flage]	
[Flags]	CY —
	OV"1" if overflow occurs; otherwise, "0".S"1" if the operation result quotient is negative; otherwise, "0".
	Z "1" if the operation result quotient is "0"; otherwise, "0".
	SAT —
[Description]	(1) Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1 and stores the quotient to general-purpose register reg2. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.
	(2) Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

RENESAS
[Supplement] (1) The remainder is not stored. Overflow occurs when the maximum negative value (8000 0000_H) is divided by -1 with the quotient = 8000 0000_H and when the data is divided by 0 with quotient being undefined. When an exception occurs during the DIVH instruction execution, the execution is aborted to process the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.
(2) Overflow occurs when the maximum negative value (8000 0000_H) is divided by -1 with the quotient = 8000 0000_H and when the data is divided by 0 with quotient being undefined. If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during the DIVH instruction execution, the execution is

When an exception occurs during the DIVH instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and generalpurpose register reg2 retain their values prior to execution of this instruction.

CAUTIONS

- 1. If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.
- 2. Do not specify r0 as reg1 and reg2 for DIVH reg1 and reg2 in instruction format (1).



<Divide instruction>

DIVHU	Divide halfword unsigned		
	Division of (unsigned) halfword data		
[Instruction format]	DIVHU reg1, reg2, reg3		
[Operation]	$GR [reg2] \leftarrow GR [reg2] \div zero-extend (GR [reg1] (15:0))$ $GR [reg3] \leftarrow GR [reg2] \% zero-extend (GR [reg1] (15:0))$		
[Format]	Format XI		
[Opcode]	15 031 16 rrrr111111RRRRR wwww01010000010		
[Flags]	CY—OV"1" if overflow occurs; otherwise, "0".S"1" when the operation result quotient word data is "1"; otherwise, "0"Z"1" if the operation result quotient is "0"; otherwise, "0".SAT—		
[Description]	Divides the word data of general-purpose register reg2 by the lower halfword data of general- purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.		
[Supplement]	Overflow occurs by division by zero (with the operation result being undefined). If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during the DIVHU instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.		
	CAUTION		
	If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.		



<High-speed divide instructions>

CHIgh-speed divide instruct	Divide word quickly
	Division of (signed) word data (variable steps)
[Instruction format]	DIVQ reg1, reg2, reg3
[Operation]	$GR [reg2] \leftarrow GR [reg2] \div GR [reg1]$ $GR [reg3] \leftarrow GR [reg2] \% GR [reg1]$
[Format]	Format XI
[Opcode]	15 031 16 rrrr111111RRRRR wwww01011111100
[Flags]	CY—OV"1" when overflow occurs; otherwise, "0".S"1" when operation result quotient is a negative value; otherwise, "0".Z"1" when operation result quotient is a "0"; otherwise, "0".SAT—
[Description]	Divides the word data in general-purpose register reg2 by the word data in general-purpose register reg1, stores the quotient in reg2, and stores the remainder in general-purpose register reg3. General-purpose register reg1 is not affected. The minimum number of steps required for division is determined from the values in reg1 and reg2, then this operation is executed. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.
[Supplement]	 Overflow occurs when the maximum negative value (8000 0000_H) is divided by -1 (with the quotient = 8000 0000_H) and when the data is divided by 0 with the quotient being undefined. If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during execution of this instruction, the execution is aborted. After exception handling is completed, the execution resumes at the original instruction address when returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction. The smaller the difference in the number of valid bits between reg1 and reg2, the smaller the number of execution cycles. In most cases, the number of instruction cycles is smaller than that of the ordinary division instruction. If data of 16-bit integer type is divided by another 16-bit integer type data, the difference in the number of valid bits is 15 or less, and the operation is completed within 20 cycles.

CAUTIONS

- 1. If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.
- 2. For the accurate number of execution cycles, see the appendix.
- 3. If the number of execution cycles must always be constant to guarantee real-time features, use the ordinary division instruction.



<High-speed divide instructions>

Chigh-speed divide instruct	Divide word unsigned quickly
	Division of (unsigned) word data (variable steps)
[Instruction format]	DIVQU reg1, reg2, reg3
[Operation]	$GR [reg2] \leftarrow GR [reg2] \div GR [reg1]$ $GR [reg3] \leftarrow GR [reg2] \% GR [reg1]$
[Format]	Format XI
[Opcode]	15 031 16 rrrr111111RRRRR wwww0101111110
[Flags]	CY—OV"1" when overflow occurs; otherwise, "0".S"1" when operation result quotient is a negative value; otherwise, "0".Z"1" when operation result quotient is a "0"; otherwise, "0".SAT—
[Description]	Divides the word data in general-purpose register reg2 by the word data in general-purpose register reg1, stores the quotient in reg2, and stores the remainder in general-purpose register reg3. General-purpose register reg1 is not affected. The minimum number of steps required for division is determined from the values in reg1 and reg2, then this operation is executed. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.
[Supplement]	(1) An overflow occurs when there is division by zero (the operation result is undefined). If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during execution of this instruction, the execution is aborted. After exception handling is completed, using the return address as this instruction's start address, the execution resumes when returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.
	(2) The smaller the difference in the number of valid bits between reg1 and reg2, the smaller the number of execution cycles. In most cases, the number of instruction cycles is smaller than that of the ordinary division instruction. If data of 16-bit integer type is divided by another 16-bit integer type data, the difference in the number of valid bits is 15 or less, and the operation is completed within 20 cycles.

CAUTIONS

- 1. If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.
- 2. For the accurate number of execution cycles, see the appendix.
- 3. If the number of execution cycles must always be constant to guarantee real-time features, use the ordinary division instruction.



<Divide instruction>

DIVU	Divide word unsigned			
	Division of (unsigned) word data			
[Instruction format]	DIVU reg1, reg2, reg3			
[Operation]	$GR [reg2] \leftarrow GR [reg2] \div GR [reg1]$ $GR [reg3] \leftarrow GR [reg2] \% GR [reg1]$			
[Format]	Format XI			
[Opcode]	15 031 16 rrrr111111RRRRR wwww01011000010			
[Flags]	CY—OV"1" if overflow occurs; otherwise, "0".S"1" when operation result quotient word data MSB is "1"; otherwise, "0".Z"1" if the operation result quotient is "0"; otherwise, "0".SAT—			
[Description]	Divides the word data of general-purpose register reg2 by the word data of general-purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.			
[Supplement]	When an exception occurs during the DIVU instruction execution, the execution is aborted to process the exception.If reg2 and reg3 are the same register, the remainder is stored in that register.The execution resumes at the original instruction address upon returning from the exception.General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.			
	CAUTION			
	If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.			



<special instruction=""></special>	Enable interrupt
EI	Enable EI level maskable exception
[Instruction format]	EI
[Operation]	PSW.ID $\leftarrow 0$ (enables EI level maskable exception)
[Format]	Format X
[Opcode]	15 031 16 1000011111100000 000000101100000
[Flags]	CY — OV — S — Z — SAT — ID 0
[Description]	Clears the ID flag of the PSW to "0" and enables the acknowledgement of maskable exceptions starting the next instruction.
[Supplement]	If the MCTL.UIC bit has been cleared to 0, this instruction is a supervisor-level instruction. If the MCTL.UIC bit has been set to 1, this instruction can always be executed.



		Return from trap or interrupt
EIRET		Return from EL level exception
[Instruction format]	EIRET	
[Operation]	PC ← EIPC	2
	PSW ← EI	PSW
[Format]	Format X	
[Opcode]	15 00000111	031 16 .11100000 000000101001000
[Flags]	L	
	CY	Value read from EIPSW is set
	OV	Value read from EIPSW is set
	S	Value read from EIPSW is set
	Z	Value read from EIPSW is set
	SAT	Value read from EIPSW is set
[Description]	EIPC and E If EP = 0, it	ecution from an EI level exception. The return PC and PSW are loaded from the EIPSW registers and set in the PC and PSW, and control is passed. means that interrupt (EIINT <i>n</i>) processing has finished, so the corresponding bit of gister is cleared.
[Supplement]	This instruc	ction is a supervisor-level instruction.



<pre><special instruction=""> FERET</special></pre>	Return from trap or interrupt		
	Return from FE level exception		
[Instruction format]	FERET		
[Operation]	$PC \leftarrow FEPC$		
	$PSW \leftarrow FEPSW$		
[Format]	Format X		
[Opcode]	15 031 16 0000011111100000 000000101001010 0		
[Flags]			
	CY Value read from FEPSW is set		
	OV Value read from FEPSW is set		
	S Value read from FEPSW is set		
	ZValue read from FEPSW is setSATValue read from FEPSW is set		
[Description]	Returns execution from an FE level exception. The return PC and PSW are loaded from the FEPC and FEPSW registers and set in the PC and PSW, and control is passed.		
[Supplement]	This instruction is a supervisor-level instruction.		
	CAUTION		
	The FERET instruction can also be used as a hazard barrier instruction when the CPU's operating status (PSW) is changed by a control program such as the OS. Use the FERET instruction to clarify the program blocks on which to effect the hardware function associated with the UM bit in the PSW when these bits are changed to accord with the mounted CPU. The hardware function that operates in accordance with the PSW value updated by the FERET instruction is guaranteed to be effected from the instruction indicated by the return address of the FERET instruction.		



FETRAP			FE-level Trap
			FE level software exception
[Instruction format]	FETRAP ve	ector4	
[Operation]	FEPC ← PC	+ 2 (return PC)	
	$FEPSW \leftarrow F$		
		eption cause code ^{*1}	
	PSW.UM ←		
	$PSW.NP \leftarrow PSW.EP \leftarrow 1$		
	$PSW.ID \leftarrow 1$		
		tion handler address ^{*2}	
	Note 1.	See Table 4.1, Exception Cause List.	
	Note 2.	See Section 4.4, Exception Handler Address.	
[Format]	Format I		
[Opcode]			
	15	0	
	0vvvv0000	0100000	
	Where vvvv	z is vector4	
	Do not set 0 ₁	to vector4 (vvvv \neq 0000).	
[Flags]	CY		
	OV	_	
	S		
	Z	_	
	SAT	_	



[Description]

Saves the contents of the return PC (address of the instruction next to the FETRAP instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores the exception cause code in the FEIC register, and updates the PSW according to the exception causes listed in **Table 4.1**. Execution then branches to the exception handler address and exception handling is started.

Table 7.6 shows the correspondence between vector4 and exception cause codes andexception handler address offset. Exception handler addresses are calculated based on theoffset addresses listed in Table 7.6. For details, see Section 4.4, Exception HandlerAddress.

Table 7.6 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset

vector4	Exception Cause Code	Offset Address	
0 _H		Not specifiable	
1 _H	0000 0031 _H	30 _H	
2 _H	0000 0032 _H		
F _H	0000 003F _H		



<special instruction=""></special>	Hal
HALT	Hal
[Instruction format]	HALT
[Operation]	Places the CPU core in the HALT state.
[Format]	Format X
[Opcode]	15 031 16 0000011111100000 000000100100000 000000000000000000000000000000000000
[Flags]	CY — OV — S — Z — SAT —
[Description]	Places the CPU core that executed the HALT instruction in the HALT state. Occurrence of the HALT state release request will return the system to normal execution status. If an exception is acknowledged while the system is in HALT state, the return PC of that exception is the PC of the instruction that follows the HALT instruction. The HALT state is released under the following condition.
	 A terminating exception occurs Even if the conditions for acknowledging the above exceptions are not satisfied (due to the ID or NP value), as long as a HALT mode release request exists, HALT state is released (for example, even if PSW.ID = 1, HALT state is released when INT0 occurs). Note, however, that the HALT mode will not be released if terminating exceptions are masked by the following mask settings, which are defined individually for each function: Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller*1. Terminating exceptions are masked by a mask setting specified by using the floating-point operation exception enable bit. Terminating exceptions are masked by a mask setting defined by a hardware function other than the above.

Note 1. This does not include masking specified by the ISPR and PMR registers.

[Supplement]

This instruction is a supervisor-level instruction.



<Data manipulation instructions>

нѕн		Halfword swap halfword
		Halfword swap of halfword data
[Instruction format]	HSH reg.	2, reg3
[Operation]	GR [reg3]	$] \leftarrow GR [reg2]$
[Format]	Format X	II
[Opcode]	15 rrrr11	031 16 L111100000 wwww01101000110
[Flags]	CY OV S Z SAT	 "1" if the lower halfword of the operation result is "0"; otherwise, "0". "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the lower halfword of the operation result is "0"; otherwise, "0".
[Description]		content of general-purpose register reg2 in general-purpose register reg3, and stores udgment result in PSW.



<Data manipulation instructions>

нѕѡ		Halfword swap word
		Halfword swap of word data
[Instruction format]	HSW reg	g2, reg3
[Operation]	GR [reg3] \leftarrow GR [reg2] (15:0) GR [reg2] (31:16)
[Format]	Format X	П
[Opcode]	15 rrrr1:	031 16 1111100000 wwww01101000100
[Flags]	CY OV S Z SAT	 "1" when there is at least one halfword of zero in the word data of the operation result; otherwise; "0". 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if operation result word data is "0"; otherwise, "0".
[Description]	Executes	endian swap.



<Branch instruction>

<branch instruction=""></branch>		Jump and register link
JARL		Branch and register link
[Instruction format]	(1) JARL disp22, reg2	
	(2) JARL disp32, reg1	
	(3) JARL [reg1], reg3	
[Operation]	(1) GR [reg2] \leftarrow PC + 4 PC \leftarrow PC + sign-extend (disp22)	
	(2) GR $[reg1] \leftarrow PC + 6$ PC $\leftarrow PC + disp32$	
	(3) $GR[reg3] \leftarrow PC + 4$ $PC \leftarrow GR[reg1]$	
[Format]	(1) Format V	
	(2) Format VI	
	(3) Format XI	
[Opcode]		
	150.3116(1)rrrrr11110ddddddddddddddddddddddddddddddd	
	ddddddddddddddd is the higher 21 bits of disp22.	
	rrrrr \neq 00000 (Do not specify r0 for reg2.)	

	15 () 31 · · ·	16 47 3	\$2
(2)	00000010111RRRRF	ddddddddddddd	סססססססססססססס 0.	D

DDDDDDDDDDDDDDDDddddddddddd is the higher 31 bits of disp32. RRRRR $\neq 00000$ (Do not specify r0 for reg1.)

WWWWW \neq 00000 (Do not specify r0 for reg3.)



[Flags]

CY	_
OV	—
S	_
Z	_
SAT	_

[Description]

- (1) Saves the current PC value + 4 in general-purpose register reg2, adds the 22-bit displacement data, sign-extended to word length, to PC; stores the value in and transfers the control to PC. Bit 0 of the 22-bit displacement is masked to "0".
- (2) Saves the current PC value + 6 in general-purpose register reg1, adds the 32-bit displacement data to PC and stores the value in and transfers the control to PC. Bit 0 of the 32-bit displacement is masked to "0".
- (3) Stores the current PC value + 4 in reg3, specifies the contents of reg1 for the PC value, and then transfers the control.

[Supplement] The current PC value used for calculation is the address of the first byte of this instruction itself. The jump destination is this instruction with the displacement value = 0. JARL instruction corresponds to the call function of the subroutine control instruction, and saves the return PC address in either reg1 or reg2. JMP instruction corresponds to the return function of the subroutine control instruction, and can be used to specify general-purpose register containing the return address as reg1 to the return PC.

CAUTION

Do not specify r0 for the general-purpose register reg2 in the instruction format (1) JARL disp22, reg2.

Do not specify r0 for the general-purpose register reg1 in the instruction format (2) JARL disp32, reg1.

Do not specify r0 for the general-purpose register reg3 in the instruction format (3) JARL [reg1], reg3.



<Branch instruction>

JMP	Jump register
	Unconditional branch (register relative)
[Instruction format]	 (1) JMP [reg1] (2) JMP disp32 [reg1]
[Operation]	 PC ← GR [reg1] PC ← GR [reg1] + disp32
[Format]	 Format I Format VI
[Opcode]	(1) 15 0 15 0 31 16 47 32 (2) 00000110111RRRR ddddddddddddddddddddddd
[Flags]	DDDDDDDDDDDDDDDddddddddddddd is the higher 31 bits of disp32. CY – OV – S – Z – SAT –
[Description]	 Transfers the control to the address specified by general-purpose register reg1. Bit 0 of the address is masked to "0". Adds the 32-bit displacement to general-purpose register reg1, and transfers the control to the resulting address. Bit 0 of the address is masked to "0".
[Supplement]	Using this instruction as the subroutine control instruction requires the return PC to be specified by general-purpose register reg1.



<Branch instruction>

<branch instruction=""></branch>	Jump relative
JR	Unconditional branch (PC relative)
[Instruction format]	(1) JR disp22
	(2) JR disp32
[Operation]	(1) $PC \leftarrow PC + sign-extend (disp22)$
	(2) $PC \leftarrow PC + disp32$
[Format]	(1) Format V
	(2) Format VI
[Opcode]	
	15 0 31 16 (1) 0000011110aadadad dadaadadadadadadadadadadadadadadadada
	ddddddddddddddd is the higher 21 bits of disp22.
	15 0 31 16 47 32
	(2) 0000001011100000 dddddddddddddd DDDDDDDDDD
	DDDDDDDDDDDDDDDddddddddddddd is the higher 31 bits of disp32.
[Flags]	
	CY — OV —
	s —
	Z —
	SAT —
[Description]	(1) Adds the 22-bit displacement data, sign-extended to word length, to the current PC and stores the value in and transfers the control to PC. Bit 0 of the 22-bit displacement is masked to "0".
	(2) Adds the 32-bit displacement data to the current PC and stores the value in PC and transfers the control to PC. Bit 0 of the 32-bit displacement is masked to "0".
[Supplement]	The current PC value used for calculation is the address of the first byte of this instruction itself. The displacement value being "0" signifies that the branch destination is the instruction itself.

RENESAS

<Load instruction>

<load instruction=""></load>	Load byte
LD.B	Load of (signed) byte data
[Instruction format]	 (1) LD.B disp16 [reg1], reg2 (2) LD.B disp23 [reg1], reg3
[Operation]	 (1) adr ← GR [reg1] + sign-extend (disp16)*1 GR [reg2] ← sign-extend (Load-memory (adr, Byte))
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 GR [reg3] ← sign-extend (Load-memory (adr, Byte))
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	 (1) Format VII (2) Format XIV
[Opcode]	15 0 31 16 (1) rrrr111000RRRR dddddddddddddddd
	15 0 31 16 47 32 (2) 00000111100RRRR wwwwwddddddd0101 DDDDDDDDDDDDDDDDDDDDDDD
	Where $RRRR = reg1$, $wwww = reg3$. ddddddd is the lower side bits 6 to 1 of disp23.
	DDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	CY – OV – S – Z – SAT –
[Description]	(1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and stored in general-purpose register reg2.
	(2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign- extended to word length to generate a 32-bit address. Byte data is read from the generated

(2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and stored in general-purpose register reg3.

<Load instruction>

	Load byte unsigned
LD.BU	Load of (unsigned) byte data
[Instruction format]	 (1) LD.BU disp16 [reg1], reg2 (2) LD.BU disp23 [reg1], reg3
[Operation]	 (1) adr ← GR [reg1] + sign-extend (disp16)*1 GR [reg2] ← zero-extend (Load-memory (adr, Byte))
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 GR [reg3] ← zero-extend (Load-memory (adr, Byte))
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	 (1) Format VII (2) Format XIV
[Opcode]	(1) $\begin{array}{c} 15 & 0.31 & 16 \\ rrrrr11110bRRRR & dddddddddddddddddddddddddddddddd$
	15 0 31 16 47 32 (2) 00000111101RRRRR wwwwwddddddd0101 DDDDDDDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. ddddddd is the lower 7 bits of disp23. DDDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	CY — OV —
	S — Z — SAT —



[Description]

- (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored in general-purpose register reg2.
- (2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored in general-purpose register reg3.

CAUTION

Do not specify r0 for reg2.



<Load instruction>

LD.DW							oad Double Word
[Instruction format]	LD.DW di	isp23[reg1], r	eg3				f doubleword data
		т с - <i>О</i> -Ју -	J				
[Operation]	data ← Loa	[reg1] + sign- ad-memory (a - 1] GR [reg	dr, Double-				
	Note 1	. An MAE or M	IDP exception	n might occur d	lepending on	the result of addres	ss calculation.
[Format]	Format XIV	V					
[Opcode]	15	0) 31	16	47	32	
		1101RRRRR				ססססססססס	
	ddddd is	RRRR = reg1, the lower sid	le bits 6 to 1	of disp23.	sp23.		
[Flags]	CY	_					
	OV	_					
	S	_					
	Z SAT	_					
	SAI	_					
[Description]	length to th generated 3	e word data c	of general-pu and the low	urpose registe	er reg1. Dou	alue sign-extend bleword data is r eneral-purpose re	read from the
[Supplement]	reg3 must b	oe an even-nu	mbered regi	ster.			



<Load instruction>

	Load halfword
LD.H	Load of (unsigned) halfword data
[Instruction format]	 (1) LD.H disp16 [reg1], reg2 (2) LD.H disp23 [reg1], reg3
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*1 GR [reg2] ← sign-extend (Load-memory (adr, Halfword))
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 GR [reg3] ← sign-extend (Load-memory (adr, Halfword))
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	 (1) Format VII (2) Format XIV
[Opcode]	(1) Trrrr111001RRRRR dddddddddddddddddddddddddddd
	(2) $15 0.31 16.47 32$ (2) $0000111100RRRR wwwwdddddd00111 DDDDDDDDDDDDDDDDDDDDWhere RRRR = reg1, wwww = reg3.dddddd is the lower side bits 6 to 1 of disp23.DDDDDDDDDDDDDDDDDDD is the higher 16 bits of disp23.$
[Flags]	CY – OV – S – Z – SAT –



[Description]

- Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address, sign-extended to word length, and stored in general-purpose register reg2.
- (2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address, sign-extended to word length, and stored in general-purpose register reg3.



<Load instruction>

LD.HU	Load halfword unsigned
	Load of (signed) halfword data
[Instruction format]	 (1) LD.HU disp16 [reg1], reg2 (2) LD.HU disp23 [reg1], reg3
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*1 GR [reg2] ← zero-extend (Load-memory (adr, Halfword))
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 GR [reg3] ← zero-extend (Load-memory (adr, Halfword))
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	 Format VII Format XIV
[Opcode]	(1) $\frac{15 \qquad 0.31 \qquad 16}{rrrrr111111RRRRR} ddddddddddddddddddddddddddd$
	15 0 31 16 47 32 (2) 00000111101RRRRR wwwwwdddddd00111 DDDDDDDDDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. dddddd is the lower side bits 6 to1 of disp23. DDDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	CY – OV – S – Z – SAT –



[Description]

- Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address, zero-extended to word length, and stored in general-purpose register reg2.
- (2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this address, zero-extended to word length, and stored in general-purpose register reg3.

CAUTION

Do not specify r0 for reg2.



<Load instruction>

LD.W	Load word
	Load of word data
[Instruction format]	 (1) LD.W disp16 [reg1], reg2 (2) LD.W disp23 [reg1], reg3
[Operation]	 (1) adr ← GR [reg1] + sign-extend (disp16)^{*1} GR [reg2] ← Load-memory (adr, Word)
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 GR [reg3] ← Load-memory (adr, Word)
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	 Format VII Format XIV
[Opcode]	<u>15 0.31 16</u>
	(1) rrrr111001RRRRR dddddddddddddddd
	Where dddddddddddd is the higher 15 bits of disp16.
	<u>15 0 31 16 47 32</u>
	(2) 00000111100RRRRR www.wdddddd01001 DDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. dddddd is the lower side bits 6 to 1 of disp23. DDDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	
r0~1	CY —
	ov —
	s —
	SAT —



[Description]

- Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Word data is read from this 32-bit address, and stored in general-purpose register reg2.
- (2) Adds the word data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address. Word data is read from this address, and stored in general-purpose register reg3.



LDL.W	Load Linked
	Load to start atomic word data manipulation
[Instruction format]	LDL.W [reg1], reg3
[Operation]	adr \leftarrow GR[reg1] ^{*1} GR[reg3] \leftarrow Load-memory (adr, Word) LLbit $\leftarrow 1^{*2}$
	Note 1. An MAE, MDP, or DTLBE exception might occur depending on the result of address calculation.
	Note 2. The result of an interrupt or exception, or the execution of a CLL, EIRET, or FERET instruction is LLbit ← 0.
[Format]	Format VII
[Opcode]	15 0 31 16 00000111111RRRRR wwww01101111000
[Flags]	CY – OV – S – Z – SAT –
[Description]	In order to perform an atomic read-modify-write operation, word data is read from the memory and stored in general-purpose register reg3. A link is then generated corresponding to the address range that includes the specified address. Subsequently, if a specific condition is satisfied before an STC.W instruction is executed for this LDL.W instruction, the link will be deleted. If an STC.W instruction is executed after the link has been deleted, STC.W execution will fail. If an STC.W instruction is executed while the link is still available, STC.W execution will succeed. The link is also deleted in this case. The LDL.W and STC.W instructions can be used to accurately update the memory in a multicore system.
[Supplement]	Use the LDL.W and STC.W instructions instead of the CAXI instruction if an atomic guarantee is required when updating the memory in a multi-core system.



LDSR	Load to system registe
LDON	Load to system registe
[Instruction format]	LDSR reg2, regID, selID LDSR reg2, regID
[Operation]	SR [regID, selID] \leftarrow GR [reg2] ^{*1}
	 Note 1. An exception might occur depending on the access permission. For details, see Section 2.5.3, Register Updating.
[Format]	Format IX
[Opcode]	15 0 31 16
	rrrr111111RRRRR ssss00000100000
	rrrr: regID, ssss: selID, RRRR: reg2
[Flags]	CY —
	OV —
	s —
	Z — SAT —
[Description]	Loads the word data of general-purpose register reg2 to the system register specified by the
	system register number and group number (regID, selID). General-purpose register reg2 is no affected. If selID is omitted, it is assumed that selID is 0.
[Supplement]	A PIE or UCPOP exception might occur as a result of executing this instruction, depending on the combination of CPU operating mode and system register to be accessed. For details, see Section 2.5.3, Register Updating .
	CAUTIONS
	 In this instruction, general-purpose register reg2 is used as the source register, but, for mnemonic description convenience, the general-purpose register reg1 field is used in the opcode. The meanings of the register specifications in the mnemonic descriptions and opcode therefore differ from those of other instructions.
	 The system register number or group number is a unique number used to identify each system register. How to access undefined registers is described in Section 2.5.4, Accessing Undefined Registers, but accessing undefined registers is not recommended.



<Loop instruction>

LOOP	Loop
	Loop
[Instruction format]	LOOP reg1,disp16
[Operation]	$GR[reg1] \leftarrow GR[reg1] + (-1)^{*1}$
	if $(GR[reg1] != 0)$ then $PC \leftarrow PC - zero-extend (disp16)$
	Note 1. –1 (0xFFFFFFFF) is added. The carry flag is updated in the same way as when the ADD instruction is executed.
[Format]	Format VII
[Opcode]	15 0 31 16 00000110111RRRRR dddddddddddddd
	Where dddddddddddd is the higher 15 bits of disp16.
[Flags]	CY"1" if a carry occurs from MSB in the reg1 operation; otherwise, "0".OV"1" if an overflow occurs in the reg1 operation; otherwise, "0".S"1" if reg1 is negative; otherwise, "0".Z"1" if reg1 is 0; otherwise, "0".SAT—
[Description]	Updates the general-purpose register reg1 by adding -1 from its contents. If the contents after this update are not 0, the following processing is performed. If the contents are 0, the system continues to the next instruction.
	• The result of logically shifting the 15-bit immediate data 1 bit to the left and zero- extending it to word length is subtracted from the current PC value, and then the control is transferred.
	• -1 (0xFFFFFFF) is added to general-purpose register reg1. The carry flag is updated in the same way as when the ADD instruction, not the SUB instruction, is executed.



[Supplement]

"0" is implicitly used for bit 0 of the 16-bit displacement. Note that, because the current PC value used for calculation is the address of the first byte of this instruction, if the displacement value is 0, the branch destination is this instruction.

CAUTION

Do not specify r0 for reg1.



<Multiply-accumulate instruction>

MAC	Multiply and add word
	Multiply-accumulate for (signed) word data
[Instruction format]	MAC reg1, reg2, reg3, reg4
[Operation]	$GR [reg4+1] \parallel GR [reg4] \leftarrow GR [reg2] \times GR [reg1] + GR [reg3+1] \parallel GR [reg3]$
[Format]	Format XI
[Opcode]	15 0.31 16 rrrrr111111RRRRR wwww0011110mmmm0
[Flags]	CY — OV — S — Z — SAT —
Description]	Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then adds the result (64-bit data) to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose register reg3+1 (for example, this would be "r7" if the reg3 value is r6 and "1" is added) as the higher 32 bits. Of the result (64-bit data), the higher 32 bits are stored in general-purpose register reg4+1 and the lower 32 bits are stored in general-purpose register reg4. The contents of general-purpose registers reg1 and reg2 are handled as 32-bit signed integers. This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.
	CAUTION General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered

General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered register (r0, r2, r4, ..., r30). The result is undefined if an odd-numbered register (r1, r3, ..., r31) is specified.



<Multiply-accumulate instruction>

MACU	Multiply and add word unsigned
	Multiply-accumulate for (unsigned) word data
[Instruction format]	MACU reg1, reg2, reg3, reg4
[Operation]	$GR [reg4+1] \parallel GR [reg4] \leftarrow GR [reg2] \times GR [reg1] + GR [reg3+1] \parallel GR [reg3]$
[Format]	Format XI
[Opcode]	15 0.31 16 rrrrr111111RRRRR wwww0011111mmmm0
[Flags]	CY — OV — S — Z — SAT —
[Description]	Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then adds the result (64-bit data) to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose register reg3+1 (for example, this would be "r7" if the reg3 value is r6 and "1" is added) as the higher 32 bits. Of the result (64-bit data), the higher 32 bits are stored in general-purpose register reg4+1 and the lower 32 bits are stored in general-purpose register reg4. The contents of general-purpose registers reg1 and reg2 are handled as 32-bit signed integers This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.
	CAUTION

General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered register (r0, r2, r4, ..., r30). The result is undefined if an odd-numbered register (r1, r3, ..., r31) is specified.


<Arithmetic instruction>

MOV	Move register/immediate (5-bit) /immediate (32-bit) Data transfe
[Leader of the Course of]	
[Instruction format]	(1) MOV reg1, reg2
	(2) MOV imm5, reg2
	(3) MOV imm32, reg1
[Operation]	(1) GR [reg2] \leftarrow GR [reg1]
	(2) GR [reg2] \leftarrow sign-extend (imm5)
	(3) GR [reg1] \leftarrow imm32
[Format]	(1) Format I
	(2) Format II
	(3) Format VI
[Opcode]	(1) $\frac{15 \qquad 0}{rrrrr \neq 00000 (Do not specify r0 for reg2.)}$ (2) $\frac{15 \qquad 0}{rrrrr \neq 00000 (Do not specify r0 for reg2.)}$ (3) $\frac{15 \qquad 0.31 \qquad 16.47 \qquad 32}{0000110001RRRRR}$
	i (bits 31 to 16) refers to the lower 16 bits of 32-bit immediate data. I (bits 47 to 32) refers to the higher 16 bits of 32-bit immediate data.
[Flags]	CY – OV – S – Z – SAT –

RENESAS

[Description]

- (1) Copies and transfers the word data of general-purpose register reg1 to general-purpose register reg2. General-purpose register reg1 is not affected.
- (2) Copies and transfers the 5-bit immediate data, sign-extended to word length, to generalpurpose register reg2.
- (3) Copies and transfers the 32-bit immediate data to general-purpose register reg1.

CAUTION

Do not specify r0 as reg2 in MOV reg1, reg2 for instruction format (1) or in MOV imm5, reg2 for instruction format (2).



<Arithmetic instruction>

MOVEA	Move effective address
	Effective address transfer
[Instruction format]	MOVEA imm16, reg1, reg2
[Operation]	$GR [reg2] \leftarrow GR [reg1] + sign-extend (imm16)$
[Format]	Format VI
[Opcode]	15 0 31 16 rrrrr110001RRRR iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
[Flags]	CY — OV — S — Z — SAT —
[Description]	Adds the 16-bit immediate data, sign-extended to word length, to the word data of general- purpose register reg1 and stores the result in general-purpose register reg2. Neither general- purpose register reg1 nor the flags is affected.
[Supplement]	This instruction is to execute a 32-bit address calculation with the PSW flag value unchanged.
	CAUTION
	Do not specify r0 for reg2.



<Arithmetic instruction>

моуні	Move high halfword
	Higher halfword transfer
[Instruction format]	MOVHI imm16, reg1, reg2
[Operation]	$GR [reg2] \leftarrow GR [reg1] + (imm16 \parallel 0^{16})$
[Format]	Format VI
[Opcode]	15 031 16 rrrr110010RRRR iiiiiiiiiiiiii
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
[Flags]	CY – OV – S – Z – SAT –
[Description]	Adds the word data with its higher 16 bits specified as the 16-bit immediate data and the lower 16 bits being "0" to the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. Neither general-purpose register reg1 nor the flags is affected.
[Supplement]	This instruction is to generate the higher 16 bits of a 32-bit address.
	CAUTION
	Do not specify r0 for reg2.



<multiply instruction=""></multiply>	
MUL	Multiply word by register/immediate (9-bit)
	Multiplication of (signed) word data
[Instruction format]	(1) MUL reg1, reg2, reg3
	(1) MOL log1, log2(2) MUL imm9, reg2, reg3
	(2) MOL mmi, 16g2, 16g5
[Operation]	(1) GR [reg3] \parallel GR [reg2] \leftarrow GR [reg2] \times GR [reg1]
	(2) GR [reg3] \parallel GR [reg2] \leftarrow GR [reg2] \times sign-extend (imm9)
[Format]	(1) Format XI
	(2) Format XII
[Opcode]	
	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
	(1) rrrr111111RRRRR wwww01000100000
	<u>15 0 31 16</u>
	(2) rrrr111111iiii wwwww01001IIII00
	iiiii are the lower 5 bits of 9-bit immediate data.
	IIII are the higher 4 bits of 9-bit immediate data.
[Flags]	
[l'lags]	Сү —
	ov —
	s —
	Z —
	SAT —
[Description]	(1) Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2. The contents of general-purpose registers reg1 and reg2 are handled as 32-bit signed integers. General-purpose register reg1 is not affected.
	(2) Multiplies the word data in general-purpose register reg2 by 9-bit immediate data, extended to word length, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2.
[Supplement]	When general-purpose register reg2 and general-purpose register reg3 are the same register, only the higher 32 bits of the multiplication result are stored in the register.

RENESAS

RH850G3MH Software

<multiply instruction=""></multiply>	
MULH	Multiply halfword by register/immediate (5-bit)
	Multiplication of (signed) halfword data
[Instruction format]	(1) MULH reg1, reg2
	(1) MULH imm5, reg2
[Operation]	(1) GR [reg2] \leftarrow GR [reg2] (15:0) × GR [reg1] (15:0)
	(2) $GR [reg2] \leftarrow GR [reg2] \times sign-extend (imm5)$
[Format]	(1) Format I
[]	(2) Format II
[Opcode]	15 0
	(1) rrrr000111RRRR
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
	15 0 (2) rrrr010111iiii
	$rrrr \neq 00000$ (Do not specify r0 for reg2.)
[Flags]	
	CY —
	OV — S —
	Z —
	SAT —
[Description]	 Multiplies the lower halfword data of general-purpose register reg2 by the halfword data of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.
	(2) Multiplies the lower halfword data of general-purpose register reg2 by the 5-bit immediate data, sign-extended to halfword length, and stores the result in general-purpose register reg2.



[Supplement]

In the case of a multiplier or a multiplicand, the higher 16 bits of general-purpose registers reg1 and reg2 are ignored.

CAUTION

Do not specify r0 for reg2.



<Multiply instruction>

MULHI	Multiply halfword by immediate (16-bit)
L	Multiplication of (signed) halfword immediate data
[Instruction format]	MULHI imm16, reg1, reg2
[Operation]	$GR [reg2] \leftarrow GR [reg1](15:0) \times imm16$
[Format]	Format VI
[Opcode]	45 0.24 46
	15 0 31 16 rrrr110111RRRRR iiiiiiiiiiiiiii
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
[Flags]	
[8-]	CY —
	ov —
	s —
	Z —
	SAT —
[Description]	Multiplies the lower halfword data of general-purpose register reg1 by the 16-bit immediate data and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.
[Supplement]	In the case of a multiplicand, the higher 16 bits of general-purpose register reg1 are ignored.
	CAUTION
	Do not specify r0 for reg2.



RH850G3MH Software

<Multiply instruction>

Μ	U	L	U	
	U		U	

Multiply word unsigned by register/immediate (9-bit)

Multiplication of (unsigned) word data

[Instruction format]	(1) MULU reg1, reg2, reg3	
	(2) MULU imm9, reg2, reg3	
[Operation]	(1) $GR [reg3] \parallel GR [reg2] \leftarrow GR [reg2] \times GR [reg1]$	
	(2) GR [reg3] \parallel GR [reg2] \leftarrow GR [reg2] \times zero-extend (imm9)	
[Format]	(1) Format XI	
	(2) Format XII	
[0		
[Opcode]	15 0 31 16	
	(1) rrrr111111RRRRR wwww01000100010	
	15 0 31 16	
	(2) rrrr111111iiii wwww01001IIII0	
	iiiii are the lower 5 bits of 9-bit immediate data.	
	IIII are the higher 4 bits of 9-bit immediate data.	
[Flags]		
[1 1485]	CY —	
	ov —	
	s —	
	Z —	
	SAT —	
[Description]	(1) Multiplies the word data in general-purpose register reg2 by the word data in general- purpose register reg1, then stores the higher 32 bits of the result (64-bit data) in general- purpose register reg3 and the lower 32 bits in general-purpose register reg2. General-purpose register reg1 is not affected.	
	(2) Multiplies the word data in general-purpose register reg2 by 9-bit immediate data, zero- extended to word length, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2.	
[Supplement]	When general-purpose register reg2 and general-purpose register reg3 are the same register, only the higher 32 bits of the multiplication result are stored in the register.	

RENESAS

<Special instruction>

NOP		No operation
		No operation
[Instruction format]	NOP	
[Operation]	No operation is performed.	
[Format]	Format I	
[Opcode]	15 0 00000000000000000	
[Flags]	CY – OV – S – Z – SAT –	
[Description]	Performs no processing and executes the next instruction.	
[Supplement]	The opcode is the same as that of MOV r0, r0.	



<Logical instruction>

ΝΟΤ		NOT
		Logical negation (1's complement)
[Instruction format]	NOT reg	1, reg2
[Operation]	GR [reg2]	$] \leftarrow \text{NOT}(\text{GR}[\text{reg1}])$
[Format]	Format I	
[Opcode]	15 rrrrr0	0 00001RRRRR
[Flags]	CY OV S Z SAT	 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0". —
[Description]		negates the word data of general-purpose register reg1 using 1's complement and result in general-purpose register reg2. General-purpose register reg1 is not



<Bit manipulation instruction>

NOT1		NOT bit
		NOT bit
[Instruction format]	(1) NOT1 bit#3, disp16 [reg1]	
	(2) NOT1 reg2, [reg1]	
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*1 token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, bit#3)) token ← not-bit (token, bit#3) Store-memory (adr, token, Byte) 	
	 (2) adr ← GR [reg1]*1 token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, reg2)) token ← not-bit (token, reg2) Store-memory (adr, token, Byte) 	
	Note 1. An MDP exception might occur depending on the result of address calculation.	
[Format]	(1) Format VIII	
	(2) Format IX	
[Opcode]	15 0 31 16	
	(1) 01bbb111110RRRRR ddddddddddddddddd	
	15 0 31 16 (2) rrrr111111RRRR 0000000011100010	
[Flags]	CY—OV—S—Z"1" if bit specified by operand = "0", "0" if bit specified by operand = "1".SAT—	



[Description]	 (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, then the bits indicated by the 3-bit bit number are inverted (0 → 1, 1 → 0) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
	 (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, then the bits specified by lower 3 bits of general-purpose register reg2 are inverted (0 → 1, 1 → 0) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
[Supplement]	The Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is executed and does not indicate the content of the specified bit resulting from the instruction execution.

CAUTION

This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.



<Logical instruction>

OR		OR
		OR
[Instruction format]	OR reg1, r	reg2
[Operation]	GR [reg2]	← GR [reg2] OR GR [reg1]
[Format]	Format I	
[Opcode]	15 rrrrr00	0 1000RRRR
[Flags]	CY OV S Z SAT	 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]		ord data of general-purpose register reg2 with the word data of general-purpose 1 and stores the result in general-purpose register reg2. General-purpose register affected.



<Logical instruction>

ORI		OR immediate (16-bit)
		OR immediate
[Instruction format]	ORI imm	16, reg1, reg2
[Operation]	GR [reg2]	← GR [reg1] OR zero-extend (imm16)
[Format]	Format VI	
[Opcode]	15 rrrrr11	0 31 16 .0100RRRRR iiiiiiiiiiiiii
[Flags]	CY OV S Z SAT	 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]	extended to	ord data of general-purpose register reg1 with the 16-bit immediate data, zero- o word length, and stores the result in general-purpose register reg2. General- gister reg1 is not affected.



<Special instruction>

	Pop registers from Stack
POPSP	POP from the stack
[Instruction format]	POPSP rh-rt
[Operation]	if rh≤rt
	then $\operatorname{cur} \leftarrow \operatorname{rt}$
	$end \leftarrow rh$
	$tmp \leftarrow sp$
	while $(\operatorname{cur} \ge \operatorname{end})$ {
	$adr \leftarrow tmp^{*1, *2}$
	GR[cur] ← Load-memory (adr, Word)
	$cur \leftarrow cur - 1$
	$tmp \leftarrow tmp + 4$
	}
	$sp \leftarrow tmp$
[Format]	Format XI
[Opcode]	
	15 0 31 16
	01100111111RRRRR wwww00101100000
	RRRRR indicates rh.
	wwwww indicates rt.
[Flags]	
	CY —
	OV — S —
	Z —
	SAT —
[Description]	Loads general-purpose register rt to rh from the stack in descending order (rt, rt -1 , rt -2 , rh). After all the registers down to the specified register have been loaded, sp is updated (incremented).

RENESAS

[Supplement]

The lower two bits of the address specified by sp are masked by 0. If an exception is acknowledged before sp is updated, instruction execution is halted and exception handling is executed with the start address of this instruction used as the return address. The POPSP instruction is then executed again. (The sp value from before the exception handling is saved.)

CAUTION

If a register that includes sp(r3) is specified as the restore register (rh = 3 to 31), the value read from the memory is not stored in sp(r3). This allows the POPSP instruction to be correctly reexecuted after execution has been halted.



<Special instruction>

PREPARE			Function prepare
			Create stack frame
[Instruction format]	(1)	PREPARE list12, imm5	
	(2)	PREPARE list12, imm5, sp/imm ^{*1}	
		Note 1. The sp/imm values are specified by bits 19 and 20 of the sub-opcode.	
[Operation]	(1)	$tmp \leftarrow sp$	
		foreach (all regs in list12) {	
		$tmp \leftarrow tmp - 4$	
		$adr \leftarrow tmp^{*1,*2}$	
		Store-memory (adr, GR[reg in list12], Word)	
		}	
		$sp \leftarrow tmp - zero-extend (imm5 logically shift left by 2)$	
	(2)	$tmp \leftarrow sp$	
		foreach (all regs in list12) {	
		$tmp \leftarrow tmp - 4$	
		$adr \leftarrow tmp^{*1,*2}$	
		Store-memory (adr, GR[reg in list12], Word)	
		}	
		$sp \leftarrow tmp - zero-extend (imm5 logically shift left by 2)$	
		case	
		$ff = 00: ep \leftarrow sp$	
		$ff = 01$: ep \leftarrow sign-extend (imm16)	
		ff = 10: ep \leftarrow imm16 logically shift left by 16	
		$ff = 11: ep \leftarrow imm32$	
		Note 1. An MDP exception might occur depending on the result of address ca	lculation.
		Note 2. The lower 2 bits of adr are masked to 0.	
[Format]	For	mat XIII	



[Opcode]

	15 () 31	16
(1)	0000011110iiiiI	LLLLLLLLL000	01
	_15 ()_31	16 Option (47-32 or 63-32)
(2)	0000011110iiiiI	LLLLLLLLLff0	11 imm16/imm32

In the case of 32-bit immediate data (imm32), bits 47 to 32 are the lower 16 bits of imm32 and bits 63 to 48 are the higher 16 bits of imm32.

ff = 00: sp is loaded to ep

ff = 01: Sign-extended 16-bit immediate data (bits 47 to 32) is loaded to ep

ff = 10: 16-bit logical left-shifted 16-bit immediate data (bits 47 to 32) is loaded to ep

ff = 11: 32-bit immediate data (bits 63 to 32) is loaded to ep

The values of LLLLLLLLL are the corresponding bit values shown in register list "list12" (for example, the "L" at bit 21 of the opcode corresponds to the value of bit 21 in list12).

list12 is a 32-bit register list, defined as follows.

31	30	29	28	27	26	25	24	23	22	21	20 1	0
r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31	-	r30

Bits 31 to 21 and bit 0 correspond to general-purpose registers (r20 to r31), so that when any of these bits is set (1), it specifies a corresponding register operation as a processing target. For example, when r20 and r30 are specified, the values in list12 appear as shown below (register bits that do not correspond, i.e., bits 20 to 1 are set as "Don't care").

- When all of the register's non-corresponding bits are "0": $0800\ 0001_{\rm H}$
- When all of the register's non-corresponding bits are "1": 081F FFFF_H

[Flags]

CY	_
OV	—
S	—
Z	—
SAT	_

[Description]	 Saves general-purpose registers specified in list12 (4 is subtracted from the sp value and the data is stored in that address). Next, subtracts 5-bit immediate data, logically left- shifted by 2 bits and zero-extended to word length, from sp.
	 (2) Saves general-purpose registers specified in list12 (4 is subtracted from the sp value and the data is stored in that address). Next, subtracts 5-bit immediate data, logically left-shifted by 2 bits and zero-extended to word length, from sp. Then, loads the data specified by the third operand (sp/imm) to ep.
[Supplement]	list12 general-purpose registers are saved in ascending order (r20, r21,, r31). imm5 is used to create a stack frame that is used for auto variables and temporary data. The lower two bits of the address specified by sp are masked to 0 and aligned to the word boundary.
	CAUTION

If an exception occurs while this instruction is being executed, execution of the instruction might be stopped after the write cycle and the register value write operation are completed, but sp will retain its original value from before the start of execution. The instruction will be executed again later, after a return from the exception.



<Special instruction>

PUSHSP

Push registers to Stack

Push registers to Stack

[Instruction format]	PUSHSP rh-rt						
[Operation]	if rh≤rt						
	then $\operatorname{cur} \leftarrow \operatorname{rh}$						
	end \leftarrow rt						
	tmp ← sp						
	while (cur \leq end) {						
	$tmp \leftarrow tmp - 4$						
	$adr \leftarrow tmp^{*1,*2}$						
	Store-memory (adr, GR[cur], Word)						
	$cur \leftarrow cur + 1$						
	}						
	$sp \leftarrow tmp$						
	Note 1. An MDP exception might occur depending on the result of address calculation.Note 2. The lower 2 bits of adr are masked to 0.						
[Format]	Format XI						
[Opcode]	15 0 31 16 01000111111RRRRR wwww00101100000						
	RRRR indicates rh. wwww indicates rt.						
[Flags]							
	CY — OV —						
	S —						
	Z —						
	SAT —						
[Description]	Stores general-purpose register rh to rt in the stack in ascending order (rh, rh $+1$, rh $+2$,, rt). After all the specified registers have been stored, sp is updated (decremented).						

RENESAS

[Supplement]

The lower two bits of the address specified by sp are masked by 0. If an exception is acknowledged before sp is updated, instruction execution is halted and exception handling is executed with the start address of this instruction used as the return address. The PUSHSP instruction is then executed again. (The sp value from before the exception handling is saved.)



<Special instruction>

	Reserved instruction exception
RIE	Reserved instruction exception
[Instruction format]	 (1) RIE (2) RIE imm5, imm4
[Operation]	FEPC \leftarrow PC (return PC) FEPSW \leftarrow PSW FEIC \leftarrow exception cause code (0000 0060 _H) PSW.UM \leftarrow 0 PSW.NP \leftarrow 1 PSW.EP \leftarrow 1 PSW.ID \leftarrow 1 PC \leftarrow exception handler address (offset address 60 _H)
[Format]	 (1) Format I (2) Format X
[Opcode]	15 0 (1) 0000000000000
	15 031 16 (2) iiiiii1111111111 000000000000000000000000000000000000
[Flags]	CY – OV – S – Z – SAT –
[Description]	Saves the contents of the return PC (address of the RIE instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores the exception cause code in the FEIC register, and updates the PSW according to the exception causes listed in Table 4.1 . Execution then branches to the exception handler address and exception handling is started. Exception handler addresses are calculated based on the offset address 60 _H . For details, see

Section 4.4, Exception Handler Address.

<Data manipulation instruction>

	Rotate Left
ROTL	Rotate
[Instruction format]	(1) ROTL imm5, reg2, reg3
	(2) ROTL reg1, reg2, reg3
[Operation]	(1) $GR[reg3] \leftarrow GR[reg2]$ rotate left by zero-extend (imm5)
	(2) $GR[reg3] \leftarrow GR[reg2]$ rotate left by $GR[reg1]$
[Format]	Format VII
[Opcode]	15 0 31 16
	(1) rrrr111111iiii wwww00011000100
	(2) 15 0 31 16 rrrr111111RRRRR wwww00011000110
[Flags]	
	CY "1" if operation result bit 0 is "1"; otherwise "0", including if the rotate amount is "0".
	OV 0 S "1" if the operation result is negative; otherwise, "0".
	Z "1" if the operation result is "0"; otherwise, "0".
	SAT —
[Description]	(1) Rotates the word data of general-purpose register reg2 to the left by the specified shift amount, which is indicated by a 5-bit immediate value zero-extended to word length. The result is written to general-purpose register reg3. General-purpose register reg2 is not affected.
	(2) Rotates the word data of general-purpose register reg2 to the left by the specified shift amount indicated by the lower 5 bits of general-purpose register reg1. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.



<Data manipulation instruction>

SAR	Shift arithmetic right by register/immediate (5-bit)
	Arithmetic right shift
[Instruction format]	(1) SAR reg1, reg2
	(2) SAR imm5, reg2
	(3) SAR reg1, reg2, reg3
[Operation]	(1) GR [reg2] \leftarrow GR [reg2] arithmetically shift right by GR [reg1]
	(2) GR [reg2] \leftarrow GR [reg2] arithmetically shift right by zero-extend (imm5)
	(3) GR [reg3] \leftarrow GR [reg2] arithmetically shift right by GR [reg1]
[Format]	(1) Format IX
	(2) Format II
	(3) Format XI
[Opcode]	
	15 0.31 16 (1) rrrrr111111RRRRR 000000010100000
	15 0 (2) rrrr010101iiii
	(3) 15 0 31 16
[Flags]	
	CY "1" if the last bit shifted out is "1"; otherwise, "0" including non-shift.
	OV 0 S "1" if the operation result is negative; otherwise, "0".
	Z "1" if the operation result is "0"; otherwise, "0".
	SAT —



[Description] (1) Arithmetically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by copying the pre-shift MSB value to the post-shift MSB. The result is written to general-purpose register reg2. General-purpose register reg1 is not affected.

- (2) Arithmetically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the 5-bit immediate data, zero-extended to word length, by copying the pre-shift MSB value to the post-shift MSB. The result is written to general-purpose register reg2.
- (3) Arithmetically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by copying the pre-shift MSB value to the post-shift MSB. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.



<Data manipulation instruction>

SASF						Shift and set flag condition
JAJI					SI	nift and flag condition settir
Instruction format]	SASF co	ccc, reg2				
[Operation]	then GI		isfied (GR [reg2] Logically sl (GR [reg2] Logically sh			
[Format]	Format I	X				
[Opcode]	15 rrrrr1	.111110c	031 ccc 000000100000	16 0000		
[Flags]	CY OV S Z SAT	 				
[Description]	general- _I condition	ourpose reg n is not me	a specified by condition a gister reg2 by 1 bit, and s t, logically left-shifts dat e condition codes shown	sets (1) the lea ta of reg2 and	ast significated clears the l	nt bit (LSB). If a LSB.
	Condition Code	Name	Condition Formula	Condition Code	Name	Condition Formula
	0000	V	OV = 1	0100	S/N	S = 1
	1000	NV	OV = 0	1100	NS/P	S = 0
	0001	C/L	CY = 1	0101	Т	Always (unconditional
	1001	NC/NL	CY = 0	1101	SA	SAT = 1
	0010	Z	Z = 1	0110	LT	(S xor OV) = 1

[Supplement]

See the SETF instruction.

NZ

NH

Н

Z = 0

(CY or Z) = 1

(CY or Z) = 0

1010

0011

1011

1110

0111

1111

GE

LE

GT

(S xor OV) = 0

((S xor OV) or Z) = 1

((S xor OV) or Z) = 0

<Saturated operation instructions>

CATADD		Saturated add	register/immediate (5-bit)
SATADD			Saturated addition
[Instruction format]	(1) SAT	ADD reg1, reg2	
	(2) SAT.	ADD imm5, reg2	
	(3) SAT.	ADD reg1, reg2, reg3	
[Operation]	(1) GR [$[reg2] \leftarrow saturated (GR [reg2] + GR [reg1])$	
	(2) GR [$[reg2] \leftarrow saturated (GR [reg2] + sign-extend (imm5))$	
	(3) GR [$[reg3] \leftarrow saturated (GR [reg2] + GR [reg1])$	
[Format]	(1) Form	nat I	
	(2) Form	nat II	
	(3) Form	nat XI	
[Opcode]	15	0	
	(1) rr	rrr000110RRRR	
	rrrrr≠	00000 (Do not specify r0 for reg2.)	
	15	0	
	(2) rr	rrr010001iiiii	
	rrrrr≠	00000 (Do not specify r0 for reg2.)	
	15	0 31 16	
	(3) rr	rrr111111RRRRR wwww01110111010	
[Flags]			
L~0~]	CY	"1" if a carry occurs from MSB; otherwise, "0".	
	OV	"1" if overflow occurs; otherwise, "0".	
	S	"1" if saturated operation result is negative; otherwise, "0".	
	Z	"1" if saturated operation result is "0"; otherwise, "0".	
	SAT	"1" if OV = 1; otherwise, does not change.	



[Description]	 Adds the word data of general-purpose register reg1 to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, when the result exceeds the maximum positive value 7FFF FFFF_H, 7FFF FFFF_H is stored in reg2, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg2; then the SAT flag is set (1). General-purpose register reg1 is not affected.
	 (2) Adds the 5-bit immediate data, sign-extended to the word length, to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, when the result exceeds the maximum positive value 7FFF FFFF_H, 7FFF FFFF_H is stored in reg2, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg2; then the SAT flag is set (1).
	(3) Adds the word data of general-purpose register reg1 to the word data of general-purpose register reg2, and stores the result in general-purpose register reg3. However, when the result exceeds the maximum positive value 7FFF FFFF _H , 7FFF FFFF _H is stored in reg3, and when it exceeds the maximum negative value 8000 0000 _H , 8000 0000 _H is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.
[Supplement]	The SAT flag is a cumulative flag. The saturate result sets the flag to "1" and will not be cleared to "0" even if the result of the subsequent operation is not saturated. The saturated operation instruction is executed normally, even with the SAT flag set to "1".
	CAUTIONS

- 1. Use LDSR instruction and load data to the PSW to clear the SAT flag to "0".
- 2. Do not specify r0 as reg2 in instruction format (1) SATADD reg1, reg2 and in instruction format (2) SATADD imm5, reg2.



<Saturated operation instructions>

SATSUB	Saturated subtract
SAISUD	Saturated subtraction
[Instruction format]	 (1) SATSUB reg1, reg2 (2) SATSUB reg1, reg2, reg3
[Operation]	 (1) GR [reg2] ← saturated (GR [reg2] – GR [reg1]) (2) GR [reg3] ← saturated (GR [reg2] – GR [reg1])
[Format]	 Format I Format XI
[Opcode]	$15 0 \\ (1) rrrr000101RRRR \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 031 16 \\ (2) rrrr111111RRRR wwww01110011010 \\ \end{array}$
[Flags]	CY"1" if a borrow occurs from MSB; otherwise, "0".OV"1" if overflow occurs; otherwise, "0".S"1" if saturated operation result is negative; otherwise, "0".Z"1" if saturated operation result is "0"; otherwise, "0".SAT"1" if OV = 1; otherwise, does not change.
[Description]	(1) Subtracts the word data of general-purpose register reg1 from the word data of general-purpose register reg2 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF FFFF _H , 7FFF FFFF _H is stored in reg2; if the result exceeds the maximum negative value 8000 0000 _H , 8000 0000 _H is stored in reg2. The SAT flag is set to "1". General-purpose register reg1 is not affected.
	(2) Subtracts the word data of general-purpose register reg1 from the word data of general-purpose register reg2, and stores the result in general-purpose register reg3. However, when the result exceeds the maximum positive value 7FFF FFFF _H , 7FFF FFFF _H is stored in reg3, and when it exceeds the maximum negative value 8000 0000 _H , 8000 0000 _H is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.

RENESAS

[Supplement] The SAT flag is a cumulative flag. The saturate result sets the flag to "1" and will not be cleared to "0" even if the result of the subsequent operation is not saturated. The saturated operation instruction is executed normally, even with the SAT flag set to "1".

CAUTIONS

- 1. Use LDSR instruction and load data to the PSW to clear the SAT flag to "0".
- 2. Do not specify r0 as reg2 in instruction format (1) SATSUB reg1, reg2.



<Saturated operation instructions>

SATSUBI		Saturated subtract immediate	
SAISUBI		Saturated subtraction	
[Instruction format]	SATSUBI	imm16, reg1, reg2	
[Operation]	GR [reg2]	← saturated (GR [reg1] – sign-extend (imm16))	
[Format]	Format VI		
[Opcode]	15	0 31 16	
	rrrrr11	0011RRRRR iiiiiiiiiiiiiiiiiii	
	rrrrr≠0	00000 (Do not specify r0 for reg2.)	
[Flags]	CY	"1" if a borrow occurs from MSB; otherwise, "0".	
	OV	"1" if overflow occurs; otherwise, "0".	
	S	"1" if saturated operation result is negative; otherwise, "0".	
	Z	"1" if saturated operation result is "0"; otherwise, "0".	
	SAT	"1" if OV = 1; otherwise, does not change.	
[Description]	Subtracts the 16-bit immediate data, sign-extended to word length, from the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF FFFF _H , 7FFF FFFF _H is stored in reg2; if the result exceeds the maximum negative value 8000 0000 _H , 8000 0000 _H is stored in reg2. The SAT flag is set to "1". General-purpose register reg1 is not affected.		
[Supplement]	cleared to '	ag is a cumulative flag. The saturation result sets the flag to "1" and will not be '0" even if the result of the subsequent operation is not saturated. The saturated nstruction is executed normally, even with the SAT flag set to "1".	
	CAUTIONS	8	
		DSR instruction and load data to the PSW to clear the SAT flag to "0".	



<Saturated operation instructions>

SATSUBR)	Saturated subtract reverse			
SAISUBN		Saturated reverse subtraction			
[Instruction format]	SATSUBF	R reg1, reg2			
[Operation]	GR [reg2]	$\leftarrow saturated (GR [reg1] - GR [reg2])$			
[Format]	Format I				
[Opcode]		0 00100RRRR			
Flags	rrrrr≠	00000 (Do not specify r0 for reg2.)			
[Flags]	CY	"1" if a borrow occurs from MSB; otherwise, "0".			
	OV	"1" if overflow occurs; otherwise, "0".			
	S	"1" if saturated operation result is negative; otherwise, "0".			
	Z	"1" if saturated operation result is "0"; otherwise, "0".			
	SAT	"1" if OV = 1; otherwise, does not change.			
[Description]	Subtracts the word data of general-purpose register reg2 from the word data of general- purpose register reg1 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF FFFF _H , 7FFF FFFF _H is stored in reg2; if the result exceeds the maximum negative value 8000 0000 _H , 8000 0000 _H is stored in reg2. The SAT flag is set to "1". General-purpose register reg1 is not affected.				
[Supplement]	The SAT flag is a cumulative flag. The saturation result sets the flag to "1" and will not be cleared to "0" even if the result of the subsequent operation is not saturated. The saturated operation instruction is executed normally, even with the SAT flag set to "1".				
	CAUTION	S			
		LDSR instruction and load data to the PSW to clear the SAT flag to "0". ot specify r0 for reg2.			



-

<Conditional operation instructions>

CDE						Subtract on condition flag	
SBF						Conditional subtraction	
[Instruction format]	SBF co	ccc, reg1, reg	g2, reg3				
[On anotion]	:Coord	:	in Cin J				
[Operation]		itions are sat	GR [reg2] – GR [reg1] –				
			R [reg2] - GR [reg1] - 0				
		-[8-] -	[8-][8-] .				
[Format]	Format	XI					
[Opcode]							
	15		0_31	16			
	rrrri	c1111111RR	RRR wwwww011100cc	:cc0			
[Flags]							
	CY "1" if a borrow occurs from MSB; otherwise, "0".						
	OV "1" if overflow occurs; otherwise, "0".						
	S		operation result is negative;		" <u> </u>		
	Z	"1" if	operation result is "0"; other	wise, "0".			
	SAT	_					
[Description]	Subtrac	rts 1 from the	result of subtracting the	word data	of general-n	rpose register reg1 fror	
	Subtracts 1 from the result of subtracting the word data of general-purpose register reg1 from the word data of general-purpose register reg2, and stores the result of subtraction in general-						
	purpose register reg3, if the condition specified by condition code "cccc" is satisfied.						
	If the condition specified by condition code "cccc" is not satisfied, subtracts the word data of						
	general-purpose register reg1 from the word data of general-purpose register reg2, and stores						
		-	-purpose register reg3.				
	General-purpose registers reg1 and register 2 are not affected.						
	-		e condition codes shown	in the follo	wing table as	s [cccc]. (However, ccc	
	cannot	equal 1101.)					
	Condition	l		Condition	I		
	Code	Name	Condition Formula	Code	Name	Condition Formula	
	0000	V	OV = 1	0100	S/N	S = 1	
	1000	NV	OV = 0	1100	NS/P	S = 0	
	0001	C/L	CY = 1	0101	Т	Always (Unconditional	
	1001	NC/NL	CY = 0	0110	LT	(S xor OV) = 1	
	0010	Z	Z = 1	1110	GE	(S xor OV) = 0	
	1010	NZ	Z = 0	0111	LE	((S xor OV) or Z) = 1	
	0011	NH	(CY or Z) = 1	1111	GT	((S xor OV) or Z) = 0	
	1011	Н	(CY or Z) = 0	(1101)	Setting pr	ohibited	



<Bit search instructions>

SCHOL		Search zero from left
		Bit (0) search from MSB side
[Instruction format]	SCH0L r	eg2, reg3
[Operation]	GR [reg3]	$] \leftarrow$ search zero from left of GR [reg2]
[Format]	Format IX	ζ
[Opcode]	15 rrrr1	031 16 1111100000 wwww01101100100
[Flags]	CY OV S Z SAT	 "1" if bit (0) is found eventually; otherwise, "0". 0 "1" if bit (0) is not found; otherwise, "0". —
[Description]	the number purpose re When bit	word data of general-purpose register reg2 from the left side (MSB side), and writes er of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to general- egister reg3 (e.g., when bit 31 of reg2 is 0, $01_{\rm H}$ is written to reg3). (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the and is the LSB, the CY flag is set (1).



<Bit search instructions>

SCHOR		Search zero from right
		Bit (0) search from LSB side
[Instruction format]	SCH0R r	eg2, reg3
[Operation]	GR [reg3]	$] \leftarrow$ search zero from right of GR [reg2]
[Format]	Format IX	ζ
[Opcode]	15 rrrrr1	031 16 1111100000 wwww01101100000
[Flags]	CY OV S Z SAT	 "1" if bit (0) is found eventually; otherwise, "0". 0 "1" if bit (0) is not found; otherwise, "0". —
[Description]	the number purpose re When bit	word data of general-purpose register reg2 from the right side (LSB side), and writes er of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to general- egister reg3 (e.g., when bit 0 of reg2 is 0, $01_{\rm H}$ is written to reg3). (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the and is the MSB, the CY flag is set (1).


<Bit search instructions>

SCH1L		Search one from left
		Bit (1) search from MSB side
[Instruction format]	SCH1L re	eg2, reg3
[Operation]	GR [reg3]	\leftarrow search one from left of GR [reg2]
[Format]	Format IX	
[Opcode]	15 rrrrr11	031 16 .111100000 wwww01101100110
[Flags]	CY OV S Z SAT	 "1" if bit (0) is found eventually; otherwise, "0". 0 "1" if bit (0) is not found; otherwise, "0".
[Description]	the numbe purpose re When bit (word data of general-purpose register reg2 from the left side (MSB side), and writes r of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to general- gister reg3 (e.g., when bit 31 of reg2 is 1, $01_{\rm H}$ is written to reg3). (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the nd is the LSB, the CY flag is set (1).



<Bit search instructions>

SCH1R		Search one from right
		Bit (1) search from LSB side
[Instruction format]	SCH1R r	reg2, reg3
[Operation]	GR [reg3]] \leftarrow search one from right of GR [reg2]
[Format]	Format IX	ζ
[Opcode]	15	0 31 16
	rrrr1	1111100000 wwwww01101100010
(TT) 1		
[Flags]	CY	"1" if bit (0) is found eventually; otherwise, "0".
	OV	0
	S	0
	Z	"1" if bit (0) is not found; otherwise, "0".
	SAT	_
[Description]	the number purpose re When bit	word data of general-purpose register reg2 from the right side (LSB side), and writes er of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to general- egister reg3 (e.g., when bit 0 of reg2 is 1, $01_{\rm H}$ is written to reg3). (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the and is the MSB, the CY flag is set (1).



<Bit manipulation instruction>

SET1		Set bit
SET1		Bit setting
[Instruction format]	(1) SET1 bit#3, disp16 [reg1]	
	(2) SET1 reg2, [reg1]	
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*1 token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, bit#3)) token ← set-bit (token, bit#3) Store-memory (adr, token, Byte) 	
	 (2) adr ← GR [reg1]*1 token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, reg2)) token ← set-bit (token, reg2) Store-memory (adr, token, Byte) 	
	Note 1. An MDP exception might occur depending on the result of address calculation.	
[Format]	(1) Format VIII	
	(2) Format IX	
[Opcode]	15 0 31 16	
	(1) 00bbb111110RRRRR dddddddddddddddd	
	15 0 31 16 (2) rrrr111111RRRRR 000000011100000	
[Flags]	CY — OV — S — Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1". SAT —	



[Description]	 (1) Adds the word data of general-purpose register reg1 to the16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, the bits indicated by the 3-bit bit number are set (1) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
	(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the lower 3 bits indicated of general-purpose register reg2 are set (1) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
[Supplement]	The Z flag of PSW indicates the initial status of the specified bit (0 or 1) and does not indicate the content of the specified bit resulting from the instruction execution.
	CAUTION
	This instruction provides an atomic guarantee aimed at exclusive control, and during the period

This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.



<Data manipulation instruction>

SETF		Set flag condition
		Flag condition settin
Instruction format]	SETF cccc, reg2	
[Operation]	if conditions are satisfie then GR [reg2] ← 000 else GR [reg2] ← 0000	0 0001 _H
[Format]	Format IX	
[Opcode]		0 31 16 : 00000000000000000
[Flags]	CY – OV – S – Z – SAT –	
[Description]	register reg2 if a conditi	cified by condition code "cccc" is met, stores "1" to general-purpose on is met and stores "0" if a condition is not met. ndition codes shown in the following table as [cccc].

Condition Code	Name	Condition Formula	Condition Code	Name	Condition Formula
0000	V	OV = 1	0100	S/N	S = 1
1000	NV	OV = 0	1100	NS/P	S = 0
0001	C/L	CY = 1	0101	Т	Always (Unconditional)
1001	NC/NL	CY = 0	1101	SA	SAT = 1
0010	Z	Z = 1	0110	LT	(S xor OV) = 1
1010	NZ	Z = 0	1110	GE	(S xor OV) = 0
0011	NH	(CY or Z) = 1	0111	LE	((S xor OV) or Z) = 1
1011	Н	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0



[Supplement]

Examples of SETF instruction:

(1) Translation of multiple condition clauses

If A of statement *if* (*A*) in C language consists of two or greater condition clauses (a₁, a₂, a₃, and so on), it is usually translated to a sequence of *if* (*a*₁) *then*, *if* (*a*₂) *then*. The object code executes "conditional branch" by checking the result of evaluation equivalent to a_n. Because a pipeline operation requires more time to execute "condition judgment" + "branch" than to execute an ordinary operation, the result of evaluating each condition clause *if* (*a_n*) is stored in register Ra. By performing a logical operation to Ra_n after all the condition clauses have been evaluated, the pipeline delay can be prevented.

(2) Double-length operation

To execute a double-length operation, such as "Add with Carry", the result of the CY flag can be stored in general-purpose register reg2. Therefore, a carry from the lower bits can be represented as a numeric value.



<Data manipulation instruction>

еш				Sh	ift logical left by re	gister/immediate (5-bit)
SHL						Logical left shift
[Instruction format]	(1) SHL	reg1, reg2				
		imm5, reg2				
		reg1, reg2, reg3				
	(1) CD		011			
[Operation]		$[reg2] \leftarrow GR [reg$				
		$[reg2] \leftarrow GR [reg$				
	(3) GR	$[reg3] \leftarrow GR [reg$	[2] logically shi	ift left by GR [1	reg1]	
[Format]	(1) Form	nat IX				
	(2) Form	nat II				
	(3) Form	nat XI				
[Opcode]						
[Opcode]	15		0_31	16		
	(1) rr	rrr111111RRI	RRR 000000	011000000		
	15		0			
		rrr010110ii:				
	15		0 31	16		
		rrr111111RRI				
[Flags]	CY	"1" if the last	hit shifted out is "	1": otherwise "O'	' including non-shif	ł
	OV	0		.,		
	S		ration result is ne	gative; otherwise	, "0".	
	Z	"1" if the oper	ration result is "0'	; otherwise, "0".		
	SAT	_				



- [Description] (1) Logically left-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by shifting "0" to LSB. The result is written to general-purpose register reg2. General-purpose register reg1 is not affected.
 (2) Logically left-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the
 - (2) Logically left-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the 5-bit immediate data, zero-extended to word length, by shifting "0" to LSB. The result is written to general-purpose register reg2.
 - (3) Logically left-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by shifting "0" to LSB. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.



<Data manipulation instruction>

SHR				Shift logical rig	ht by register/immediate (5-bit)
					Logical right shift
[Instruction format]	(1) SHR	reg1, reg2			
		imm5, reg2			
		reg1, reg2, re	.g3		
[Operation]	(1) GR [[reg2] ← GR [1	reg2] logically shif	t right by GR [reg1]	
	(2) GR [[reg2] ← GR [ı	reg2] logically shif	t right by zero-extend (imm5)
	(3) GR [[reg3] ← GR [1	reg2] logically shif	t right by GR [reg1]	
[Format]	(1) Form	nat IX			
	(2) Form	nat II			
	(3) Form	nat XI			
[Opcode]					
	(1) 15	rrr111111F	0 31 RRRR 0000000	16 010000000	
	15		0		
	(2) rr	rrr010100i	liii		
	15		0 31	16	
	(3) rr	rrr111111F	RRRRR wwwww00	01000010	
[Flags]					
[1.1.62]	CY	"1" if the la	ast bit shifted out is "1	"; otherwise, "0" including	non-shift.
	OV	0			
	S 7		peration result is neg		
	Z SAT	- ii tiie o	peration result is "0";		



- [Description]
 (1) Logically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by shifting "0" to MSB. The result is written to general-purpose register reg2. General-purpose register reg1 is not affected.
 (2) Logically right shifts the word data of general purpose register reg2 by 'n' (0 to +31), the position specified by the lower for the position specified by the position specified
 - (2) Logically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the 5-bit immediate data, zero-extended to word length, by shifting "0" to MSB. The result is written to general-purpose register reg2.
 - (3) Logically right-shifts the word data of general-purpose register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by shifting "0" to MSB. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.



SLD.B	Short format load byte
	Load of (signed) byte data
[Instruction format]	SLD.B disp7 [ep], reg2
[Operation]	adr ← ep + zero-extend (disp7) ^{*1} GR [reg2] ← sign-extend (Load-memory (adr, Byte))
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	15 0 rrrr0110dddddd
[Flags]	CY – OV – S – Z – SAT –
[Description]	Adds the 7-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word

length, and stored in reg2.



SLD.BU	Short format load byte unsigned
	Load of (unsigned) byte data
[Instruction format]	SLD.BU disp4 [ep], reg2
[Operation]	adr \leftarrow ep + zero-extend (disp4) ^{*1} GR [reg2] \leftarrow zero-extend (Load-memory (adr, Byte))
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	15 0 rrrr0000110dddd
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
[Flags]	CY — OV — S — Z — SAT —
[Description]	Adds the 4-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored in reg2.
	CAUTION
	Do not specify r0 for reg2.



SLD.H		Short format load halfword
		Load of (signed) halfword data
[Instruction format]	SLD.H d	isp8 [ep] , reg2
[Operation]	-	+ zero-extend (disp8) ^{*1} ← sign-extend (Load-memory (adr, Halfword))
	Note	1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV	T
[Opcode]		0 000dddddd d is the higher 7 bits of disp8.
[Flags]	CY OV S Z SAT	
[Description]	generate a	element pointer to the 8-bit displacement data, zero-extended to word length, to a 32-bit address. Halfword data is read from this 32-bit address, sign-extended to th, and stored in general-purpose register reg2.



SLD.HU		Short format load halfword unsigned
		Load of (unsigned) halfword data
[Instruction format]	SLD.HU disp5 [ep], reg2	
[Operation]	adr ← ep + zero-extend (disp5) ^{*1} GR [reg2] ← zero-extend (Load-memory (adr, F	Halfword))
	Note 1. An MAE or MDP exception might occur	depending on the result of address calculation.
[Format]	Format IV	
[Opcode]	15 0 rrrrr0000111dddd rrrrr≠00000 (Do not specify r0 for reg2.) dddd is the higher 4 bits of disp5.	
[Flags]	CY — OV — S — Z — SAT —	
[Description]	Adds the element pointer to the 5-bit displacement generate a 32-bit address. Halfword data is read word length, and stored in general-purpose regis	from this 32-bit address, zero-extended to
	CAUTION	
	Do not specify r0 for reg2.	



SLD.W	Short format load word
	Load of word data
[Instruction format]	SLD.W disp8 [ep], reg2
[Operation]	adr ← ep + zero-extend (disp8) ^{*1} GR [reg2] ← Load-memory (adr, Word)
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	15 0 rrrrr1010ddddd0 ddddd is the higher 6 bits of disp8.
[Flags]	CY — OV — S — Z — SAT —
[Description]	Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address. Word data is read from this 32-bit address, and stored in general-purpose register reg2.



<Special instruction>

SNOOZE	Snooz
	Snooz
[Instruction format]	Snooze
[Operation]	Snooze while hardware-defined period
[Format]	Format X
[Opcode]	15 0 31 16 0000111111100000 0000000100100000
[Flags]	CY – OV – S – Z – SAT –
[Description]	Temporarily halts operation of the CPU core for the period defined by the hardware specifications or when the CPU enters a specific state. When the specified period has elapsed or the CPU exits the specified state, CPU operation automatically resumes and instruction execution begins from the next instruction. The SNOOZE state is released under the following conditions:
	• The predefined period of time passes
	A terminating exception occurs
	Even if the conditions for acknowledging the above exceptions are not satisfied (due to the II or NP value), as long as a SNOOZE mode release request exists, the SNOOZE state is released (for example, even if PSW.ID = 1, the SNOOZE state is released when INT0 occurs Note, however, that the SNOOZE mode will not be released if terminating exceptions are masked by the following mask settings, which are defined individually for each function:
	• Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller* ¹ .
	• Terminating exceptions are masked by a mask setting specified by using the floating- point operation exception enable bit.
	• Terminating exceptions are masked by a mask setting defined by a hardware function other than the above.
	Note 1. This does not include masking specified by the ISPR and PMR registers.

[Supplement]

This instruction is used to prevent the CPU performance from dropping in a multi-core system due to bus band occupancy during a spinlock.

CAUTION

The period of the pause triggered by the SNOOZE instruction is defined according to the hardware specifications of the CPU core. For details, see the hardware manual of the product used.



SST.B	Short format st	-
	Storage of b	yte data
[Instruction format]	SST.B reg2, disp7 [ep]	
[Operation]	adr ← ep + zero-extend (disp7) ^{*1} Store-memory (adr, GR [reg2], Byte)	
	Note 1. An MDP exception might occur depending on the result of address calculation.	
[Format]	Format IV	
[Opcode]	15 0 rrrr0111dddddd	
[Flags]	CY — OV — S — Z — SAT —	
[Description]	Adds the element pointer to the 7-bit displacement data, zero-extended to word length generate a 32-bit address and stores the data of the lowest byte of reg2 to the generated	

address.



	Short format store halfword
SST.H	
	Storage of halfword data
[Instruction format]	SST.H reg2, disp8 [ep]
[Operation]	adr \leftarrow ep + zero-extend (disp8) ^{*1}
	Store-memory (adr, GR [reg2], Halfword)
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	15 0
	rrrr1001dddddd
	dddddd is the higher 7 bits of disp8.
[Flags]	
[1 lugo]	CY —
	ov —
	s —
	Z —
	SAT —
[Description]	Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address, and stores the lower halfword data of reg2 to the generated 32-bit address.



SST.W	Short format store wor
	Storage of word dat
[Instruction format]	SST.W reg2, disp8 [ep]
[Operation]	adr ← ep + zero-extend (disp8) ^{*1} Store-memory (adr, GR [reg2], Word)
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	15 0 rrrrr1010ddddd1 ddddd is the higher 6 bits of disp8.
[Flags]	CY – OV – S – Z – SAT –
[Description]	Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address and stores the word data of reg2 to the generated 32-bit address.



ST.B	Store byte
	Storage of byte data
[Instruction format]	 ST.B reg2, disp16 [reg1] ST.B reg3, disp23 [reg1]
[Operation]	 (1) adr ← GR [reg1] + sign-extend (disp16)*1 Store-memory (adr, GR [reg2], Byte)
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 Store-memory (adr, GR [reg3], Byte)
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	 Format VII Format XIV
[Opcode]	15 0 31 16
	(1) rrrr111010RRRRR ddddddddddddddd
	15 0 31 16 47 32
	(2) 00000111100RRRRR www.wddddddd1101 DDDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. ddddddd is the lower 7 bits of disp23. DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	CY —
	OV —
	S —
	Z — SAT —
[Description]	 Adds the data of general-purpose register reg1 to the 16-bit displacement data, sign- extended to word length, to generate a 32-bit address and stores the lowest byte data of general-purpose register reg2 to the generated address.
	(2) Adds the data of general-purpose register reg1 to the 23-bit displacement data, sign- extended to word length, to generate a 32-bit address and stores the lowest byte data of general-purpose register reg3 to the generated address.

ST.DW							Store Double Word
51.000						Storage	e of doubleword data
[Instruction format]	ST.DW reg3	3, disp23[reg	1]				
[Operation]	data ← GR[ı	reg3+1] GF	extend (disp23) R[reg3] Double-word)	*1			
	Note 1.	An MAE or M	DP exception mig	jht occur d	lepending on the res	sult of add	lress calculation.
[Format]	Format XIV						
[Opcode]	15 000001112	0 101RRRRR		16 101111	47 DDDDDDDDDDDI	32 ססססס	
	dddddd is t	he lower side	wwww = reg3. e bits 6 to 1 of 6 is the higher 16	-	sp23.		
[Flags]	CY OV S Z SAT	 					
[Description]	word length	to generate a a of general-	a 32-bit address purpose registe	Doublev	word data consisti	ng of the	e sign-extended to e lower 32 bits of word data of reg3
[Supplement]	reg3 must be	e an even-nui	nbered register				



ST.H					Store halfword			
					Storage of halfword data			
[Instruction format]		H reg2, disp16 H reg3, disp23						
	(2) 51.1	11 10 <u>6</u> 3, uisp2.	[[081]					
[Operation]			+ sign-extend (dis r, GR [reg2], Half					
	(2) adr \leftarrow GR [reg1] + sign-extend (disp23) ^{*1}							
	Stor	e-memory (ad	r, GR [reg3], Half	word)				
	Note	e 1. An MAE or	MDP exception mig	ht occur depending on the rea	sult of address calculation.			
[Format]	(1) For	mat VII						
	(2) For	mat XIV						
[Opcode]	15		0 31	16				
			RRRRR dddddd					
	Where d	ddddddddd	ldddd is the high	er 15 bits of disp16.				
	15		0 31	1647	32			
	(2) 00	000111101	RRRRR wwwwdc	Idddd01101 DDDDDD	ססססססס			
	Where $RRRR = reg1$, $wwww = reg3$.							
			ide bits 6 to 1 of d	-				
	DDDDDD	וססססססססי	o is the higher 16 l	bits of disp23.				
[Flags]								
-	CY	_						
	OV	—						
	S	_						
	Z SAT	_						
	SAI	—						



[Description]

- (1) Adds the data of general-purpose register reg1 to the 16-bit displacement data, signextended to word length, to generate a 32-bit address and stores the lower halfword data of general-purpose register reg2 to the generated address.
- (2) Adds the data of general-purpose register reg1 to the 23-bit displacement data, signextended to word length, to generate a 32-bit address and stores the lower halfword data of general-purpose register reg3 to the generated address.



ST.W	Store wo
51.00	Storage of word da
[Instruction format]	 (1) ST.W reg2, disp16 [reg1] (2) ST.W reg3, disp23 [reg1]
[Operation]	 (1) adr ← GR [reg1] + sign-extend (disp16)*1 Store-memory (adr, GR [reg2], Word)
	 (2) adr ← GR [reg1] + sign-extend (disp23)*1 Store-memory (adr, GR [reg3], Word)
	Note 1. An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	 Format VII Format XIV
[Opcode]	15 0 31 16
	(1) rrrr111011RRRRR ddddddddddddddd
	Where dddddddddddd is the higher 15 bits of disp16.
	<u>15 031 1647 32</u>
	(2) 00000111100RRRRR www.wdddddd01111 DDDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. dddddd is the lower side bits 6 to 1 of disp23. DDDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	CY —
	CY OV
	s —
	Z —

SAT

_



[Description]

- (1) Adds the data of general-purpose register reg1 to the 16-bit displacement data, signextended to word length, to generate a 32-bit address and stores the word data of generalpurpose register reg2 to the generated 32-bit address.
- (2) Adds the data of general-purpose register reg1 to the 23-bit displacement data, signextended to word length, to generate a 32-bit address and stores the word data of generalpurpose register reg3 to the generated 32-bit address.



<store instruction=""></store>	Otore Conditional
STC.W	Store Conditional
	Conditional storage when atomic word data manipulation is complete
[Instruction format]	STC.W reg3, [reg1]
[Operation]	adr ← GR[reg1] ^{*1} data ← GR[reg3]
	token $\leftarrow \text{LLbit}^{*2}$
	if(token == 1)
	then Store-memory (adr, data, Word)
	$GR[reg3] \leftarrow 1$
	else GR[reg3] $\leftarrow 0$
	endif
	LLbit $\leftarrow 0^{*2}$
	 Note 1. An MAE, MDP exception might occur depending on the result of address calculation. Note 2. For details about the link operation, see Section 5.3.2, Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions.
[Format]	Format VII
[Opcode]	15 031 16 00000111111RRRRR wwww01101111010
[Flags]	CY — OV —
	S —
	Z —
	SAT —



[Description]	This instruction can only be executed successfully if a link exists that corresponds to the specified address. If a corresponding link exists, the word data of general-purpose register reg3 is stored in the memory and an atomic read-modify-write is executed. If the corresponding link has been lost, the data is not stored in the memory and execution of this instruction fails. Whether execution of the STC.W instruction has succeeded or not can be ascertained by checking the contents of general-purpose register reg3 after the instruction has been executed. If execution of the STC.W instruction was successful, general-purpose register reg3 will be set (1). If execution failed, reg3 will be cleared (0). This instruction can be used together with the LDL.W instruction to ensure accurate updating of the memory in a multi-core system.
[Supplement]	Use the LDL.W and STC.W instructions instead of the CAXI instruction if an atomic guarantee is required when updating the memory in a multi-core system.



STSR	Store contents of system register
	Storage of contents of system register
[Instruction format]	STSR regID, reg2, selID STSR regID, reg2
[Operation]	$GR [reg2] \leftarrow SR [regID, selID]^{*1}$
	 Note 1. An exception might occur depending on the access permission. For details, see Section 2.5.3, Register Updating.
[Format]	Format IX
[Opcode]	15 031 16 rrrrr111111RRRRR sssss00001000000 rrrrr: reg2, sssss: selID, RRRRR: regID
[Flags]	CY – OV – S – Z – SAT –
[Description]	Stores the system register contents specified by the system register number and group number (regID, selID) in general-purpose register reg2. The system register is not affected. If selID is omitted, it is assumed that selID is 0.
[Supplement]	A PIE or UCPOP exception might occur as a result of executing this instruction, depending on the combination of CPU operating mode and system register to be accessed. For details, see Section 2.5.3, Register Updating .
	CAUTION
	The system register number or group number is a unique number used to identify each system register. How to access undefined registers is described in Section 2.5.4, Accessing Undefined Registers , but accessing undefined registers is not recommended.

RENESAS

<Arithmetic instruction>

SUB		Subtract
		Subtraction
[Instruction format]	SUB regl	., reg2
[Operation]	GR [reg2]	$\leftarrow GR [reg2] - GR [reg1]$
[Format]	Format I	
[Opcode]	15 rrrr00	0 1101RRRRR
[Flags]	CY OV S Z SAT	 "1" if a borrow occurs from MSB; otherwise, "0". "1" if overflow occurs; otherwise, "0". "1" if the operation result is negative; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]	purpose re	the word data of general-purpose register reg1 from the word data of general- gister reg2 and stores the result in general-purpose register reg2. General-purpose g1 is not affected.



<Arithmetic instruction>

SUBR		Subtract reverse
		Reverse subtraction
[Instruction format]	SUBR reg	1, reg2
[Operation]	GR [reg2] <	\leftarrow GR [reg1] – GR [reg2]
[Format]	Format I	
[Opcode]	15 rrrrr001	0 .100rrrr
[Flags]	CY OV S Z SAT	 "1" if a borrow occurs from MSB; otherwise, "0". "1" if overflow occurs; otherwise, "0". "1" if the operation result is negative; otherwise, "0". "1" if the operation result is "0"; otherwise, "0".
[Description]	purpose reg	he word data of general-purpose register reg2 from the word data of general- gister reg1 and stores the result in general-purpose register reg2. General-purpose 1 is not affected.



<Special instruction>

SWITCH	Jump with table look up
Зинсп	Jump with table look up
[Instruction format]	SWITCH reg1
[Operation]	adr \leftarrow (PC + 2) + (GR [reg1] logically shift left by 1) ^{*1} PC \leftarrow (PC + 2) + (sign-extend (Load-memory (adr, Halfword))) logically shift left by 1
	Note 1. An MDP exception might occur depending on the result of address calculation.
[Format]	Format I
[Opcode]	$\frac{15 \qquad 0}{000000010 RRRR}$ RRRRR $\neq 00000$ (Do not specify r0 for reg1.)
[Flags]	CY – OV – S – Z – SAT –
[Description]	 The following steps are taken. (1) Adds the start address (the one subsequent to the SWITCH instruction) to general-purpose register reg1, logically left-shifted by 1, to generate a 32-bit table entry address (2) Loads the halfword entry data indicated by the address generated in step (1). (3) Adds the table start address after sign-extending the loaded halfword data and logically left-shifting it by 1 (the one subsequent to the SWITCH instruction) to generate a 32-bit target address. (4) Jumps to the target address generated in step (3).
	 CAUTIONS Do not specify r0 for reg1. In the SWITCH instruction memory read operation executed in order to read the table, memory protection is performed.



<Data manipulation instruction>

SXB		Sign extend byte
		Sign-extension of byte data
[Instruction format]	SXB reg	1
[Operation]	GR [reg1]	$] \leftarrow \text{sign-extend} (GR [reg1] (7:0))$
[Format]	Format I	
[Opcode]	15 0000000	0 D0101RRRRR
[Flags]	CY OV S Z SAT	
[Description]	Sign-exter	nds the lowest byte of general-purpose register reg1 to word length.



<Data manipulation instruction>

SXH	Sign extend h	
		Sign-extension of halfword data
[Instruction format]	SXH reg	1
[Operation]	GR [reg1]	\leftarrow sign-extend (GR [reg1] (15:0))
[Format]	Format I	
[Opcode]	15 0000000	0 00111RRRRR
[Flags]	CY OV S Z SAT	
[Description]	Sign-exter	nds the lower halfword of general-purpose register reg1 to word length.



<Special instruction>

SYNCE	Synchronize ex	
		Exception synchronization instruction
[Instruction format]	SYNCE	
[Operation]	No operat	tion is performed.
[Format]	Format I	
[Opcode]	15 0000000	0000011101
[Flags]	CY OV S Z SAT	
[Description]	In this CF	PU, the SYNCE instruction is handled as the NOP instruction.



<Special instruction>

<special instruction=""></special>	Synchronize instruction pipeline
SYNCI	Instruction pipeline synchronization instruction
[Instruction format]	SYNCI
[Operation]	Synchronizes instruction fetches.
[Format]	Format I
[Opcode]	15 0 0000000011100
[Flags]	CY – OV – S – Z – SAT –
[Description]	Discards unexecuted instructions in the pipeline, and re-fetches the subsequent instructions. The SYNCI instruction waits for the completion of execution of the cache instruction and the instruction to update the cache operation function registers. The SYNCI instruction does not wait for the result of the preceding load and store instructions.
[Supplement]	For details about the synchronization function, see Section 5.4, Synchronization Function.
	If the CPU includes an instruction cache, the instruction cache must be disabled to realize self-programming code that alters instructions on the memory.


SYNCM	Synchronize memory
	Memory synchronize instruction
[Instruction format]	SYNCM
[Operation]	Synchronizes memory accesses.
[Format]	Format I
[Opcode]	15 0 0000000011110
[Flags]	CY – OV – S – Z – SAT –
[Description]	Waits for the completion of execution of all preceding instructions and all preceding memory accesses (load and store). By executing the SYNCM instruction, the result of the preceding memory accesses can be referenced by any master device within the system.
[Supplement]	For details about the synchronization function, see Section 5.4, Synchronization Function . The completion of a store instruction may not be guaranteed by the SYNCM instruction depending on the destination of the store instruction. For details, see the hardware manual of the product used.



EVNCD		Synchronize pipeline
SYNCP		Pipeline synchronize instruction
[Instruction format]	SYNCP	
[Operation]	Synchron	izes pipeline.
[Format]	Format I	
[Opcode]	15 0000000	0 0000011111
[Flags]	CY OV S Z SAT	
[Description]	preceding completic	the completion of execution of preceding instructions to reflect the result of the g instructions to subsequent instructions. The SYNCP instruction waits for the on of load instruction (until the loaded data is stored in a register), but does not wait mpletion of store instruction (until the destination memory or register is updated).
[Supplement]	For detail Functio i	s about the synchronization function, see Section 5.4, Synchronization n.



SYSCALL	System call
	System call exception
[Instruction format]	SYSCALL vector8
[Operation]	$\begin{split} & \text{EIPC} \leftarrow \text{PC} + 4 \text{ (return PC)} \\ & \text{EIPSW} \leftarrow \text{PSW} \\ & \text{EIIC} \leftarrow \text{exception cause code}^{*1} \\ & \text{PSW.UM} \leftarrow 0 \\ & \text{PSW.EP} \leftarrow 1 \\ & \text{PSW.ID} \leftarrow 1 \\ & \text{if (vector8} \leq \text{SCCFG.SIZE) is satisfied} \\ & \text{then} \qquad \text{adr} \leftarrow \text{SCBP} + \text{zero-extend (vector8 logically shift left by 2)}^{*2} \\ & \text{else} \qquad \text{adr} \leftarrow \text{SCBP}^{*2} \\ & \text{PC} \leftarrow \text{SCBP} + \text{Load-memory (adr, Word)} \\ \end{split}$
	Note 2. An MDP exception might occur depending on the result of address calculation.
[Format]	Format X
[Opcode]	150311611010111111vvvvv00VVV00101100000Where VVV is the higher 3 bits of vector8 and vvvvv is the lower 5 bits of vector8.
[Flags]	CY – OV – S – Z –
	SAT —



[Description]

- (1) Saves the contents of the return PC (address of the instruction next to the SYSCALL instruction) and PSW to EIPC and EIPSW.
- (2) Stores the exception cause code corresponding to vector8 in the EIIC register. The exception cause code is the value of vector8 plus $8000_{\rm H}$.
- (3) Updates the PSW according to the exception causes listed in **Table 4.1**.
- (4) Generates a 32-bit table entry address by adding the value of the SCBP register and vector8 that is logically shifted 2 bits to the left and zero-extended to a word length. If vector8 is greater than the value specified by the SIZE bit of system register SCCFG; however, vector8 that is used for the generation of a 32-bit table entry address is handled as 0.
- (5) Loads the word of the address generated in (4).
- (6) Generates a 32-bit target address by adding the value of the SCBP register to the data in (5).
- (7) Branches to the target address generated in (6).

CAUTIONS

- 1. In the SYSCALL instruction memory read operation executed in order to read the table, memory protection is performed with the supervisor privilege.
- 2. Be sure not to set the target address of the SYSCALL instruction less than 0000 0100_{H} .



TRAP		Тгар
		Software exception
[Instruction format]	TRAP vector5	
[Operation]	EIPC \leftarrow PC + 4 (return PC)	
	$EIPSW \leftarrow PSW$	
	EIIC \leftarrow exception cause code ^{*1}	
	$PSW.UM \leftarrow 0$	
	$PSW.EP \leftarrow 1$	
	$PSW.ID \leftarrow 1$	
	$PC \leftarrow exception handler address^{*2}$	
	Note 1. See Table 4.1, Exception Cause List.	
	Note 2. See Section 4.4, Exception Handler Address.	
[Format]	Format X	
[Opcode]		
[Opeode]	<u>15 031 16</u>	
	00000111111vvvvv 0000000100000000	
	vvvvv = vector5	
[Flags]		
[Flags]	СҮ —	
[Flags]	CY — OV —	
[Flags]		
[Flags]	OV —	



[Description]

Saves the contents of the return PC (address of the instruction next to the TRAP instruction) and the current contents of the PSW to EIPC and EIPSW, respectively, stores the exception cause code in the EIIC register, and updates the PSW according to the exception causes listed in **Table 4.1**. Execution then branches to the exception handler address and exception handling is started.

The following table shows the correspondence between vector5 and exception cause codes and exception handler address offset. Exception handler addresses are calculated based on the offset addresses listed in the following table. For details, see **Section 4.4, Exception Handler Address**.

vector5	Exception Cause Code	Offset Address
00 _H	0000 0040 _H	40 _H
01 _H	0000 0041 _H	
0F _H	0000 004F _H	
10 _H	0000 0050 _H	50 _H
11 _H	0000 0051 _H	
1F _H	0000 005F _H	



<Logical instruction>

TST		Test
		Test
[Instruction format]	TST regl	, reg2
[Operation]	result \leftarrow	GR [reg2] AND GR [reg1]
[Format]	Format I	
[Opcode]	15 rrrro(0 D1011RRRRR
[Flags]	CY OV S Z SAT	— 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, 0. —
[Description]	register re	e word data of general-purpose register reg2 with the word data of general-purpose eg1. The result is not stored with only the flags being changed. General-purpose reg1 and reg2 are not affected.



<Bit manipulation instruction>

TST1		Test bit
1311		Bit test
[Instruction format]	(1) TST1 bit#3, disp16 [reg1]	
	(2) TST1 reg2, [reg1]	
[Operation]	 adr ← GR [reg1] + sign-extend (disp16)*1 token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, bit#3)) 	
	 (2) adr ← GR [reg1]^{*1} token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, reg2)) 	
	Note 1. An MDP exception might occur depending on the result of address calculation.	
[Format]	 Format VIII Format IX 	
[Opcode]	(1) 15 0 31 16 11bbb111110RRRRR dddddddddddddddd 15 0 31 16 rrrrr111111RRRRR 00000011100110	
[Flags]	CY — OV — S — Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1". SAT —	



- [Description]
 (1) Adds the word data of general-purpose register reg1 to the16-bit displacement data, sign-extended to word length, to generate a 32-bit address; checks the bit specified by the 3-bit bit number at the byte data location referenced by the generated address. If the specified bit is "0", "1" is set to the Z flag of PSW and if the bit is "1", the Z flag is cleared to "0". The byte data, including the specified bit, is not affected.
 - (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address; checks the bit specified by the lower 3 bits of reg2 at the byte data location referenced by the generated address. If the specified bit is "0", "1" is set to the Z flag of PSW and if the bit is "1", the Z flag is cleared to "0". The byte data, including the specified bit, is not affected.



<Logical instruction>

XOR		Exclusive OR
		Exclusive OR
[Instruction format]	XOR reg	1, reg2
[Operation]	GR [reg2]	\leftarrow GR [reg2] XOR GR [reg1]
[Format]	Format I	
[Opcode]	15 rrrr00	0 01001RRRRR
[Flags]	CY OV S Z SAT	 — 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0". —
[Description]	general-p	ly ORs the word data of general-purpose register reg2 with the word data of urpose register reg1 and stores the result in general-purpose register reg2. General- egister reg1 is not affected.



<Logical instruction>

XORI		Exclusive OR immediate (16-bit)
		Exclusive OR immediate
[Instruction format]	XORI im	m16, reg1, reg2
[Operation]	GR [reg2]	← GR [reg1] XOR zero-extend (imm16)
[Format]	Format V	Ι
[Opcode]	15 rrrrrl1	031 16 .0101RRRRR iiiiiiiiiiiiiii
[Flags]	CY OV S Z SAT	 0 "1" if operation result word data MSB is "1"; otherwise, "0". "1" if the operation result is "0"; otherwise, "0". —
[Description]	data, zero	ly ORs the word data of general-purpose register reg1 with the 16-bit immediate -extended to word length, and stores the result in general-purpose register reg2. urpose register reg1 is not affected.



<Data manipulation instruction>

ZXB		Zero extend byte
		Zero-extension of byte data
[Instruction format]	ZXB reg	1
[Operation]	GR [reg1]] \leftarrow zero-extend (GR [reg1] (7:0))
[Format]	Format I	
[Opcode]	15 0000000	0 00100RRRR
[Flags]	CY OV S Z SAT	
[Description]	Zero-exte	ends the lowest byte of general-purpose register reg1 to word length.



<Data manipulation instruction>

ZXH		Zero extend halfword
		Zero-extension of halfword data
[Instruction format]	ZXH reg	gl
[Operation]	GR [reg1]] \leftarrow zero-extend (GR [reg1] (15:0))
[Format]	Format I	
[Opcode]	15 0000000	0 00110RRRRR
[Flags]	CY OV S Z SAT	
[Description]	Zero-exte	ends the lower halfword of general-purpose register reg1 to word length.



7.3 Cache Instructions

7.3.1 Overview of Cache Instructions

This CPU provides the cache instructions to enable efficient manipulation of the cache by the CPU.

The following cache instructions (mnemonics) are available.

- CACHE: Cache
- PREF: Prefetch

7.3.2 Cache Instruction Set

This section details each instruction, dividing each mnemonic (in alphabetical order) into the following items.

- Instruction format: Indicates how the instruction is written and its operand(s).
- Operation: Indicates the function of the instruction.
- Format: Indicates the instruction format.
- Opcode: Indicates the bit field of the instruction opcode.
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.



<Cache instruction>

CACHE	Cache	
	Cache operation	
[Instruction format]	CACHE cacheop, [reg1]	
[Operation]	Manipulates the cache specified by cacheop.	
[Format]	Format X	
[Opcode]	15 031 16 111pp111111RRRRR PPPPP00101100000 ppPPPPP indicates cacheop.	
[Flags]	CY — OV — S — Z — SAT —	
[Description]	Sets the word data of general-purpose register reg1 as a 32-bit address or the cache index and manipulates the cache specified by cacheop. For details about the cache index specification method, see Section 5.2.5, Cache Index Specification Method .	
[Supplement]	Each cache operation has its own instruction execution privilege. For details about the correspondence between cache operations and instruction execution privileges, see Section 5.2.6, Execution Privilege of the CACHE/PREF Instruction .	
	When manipulating the cache by specifying the address, it might become the target of memory protection by the MPU. For details about the relationship between cache manipulation and memory protection, see Section 5.2.7, Memory Protection for CACHE and PREF Instructions.	



cacheop	Target	Processing	Cache Specification	Operation
0000000	Instruction	СНВІІ	Address	(Cache Hit Block Invalidate, Instruction cache) If the specified address hits an address in the instruction cache, the corresponding cache line is disabled. If the specified address does not hit an address in the instruction cache, no processing is performed.
0100000	Instruction	CIBII	Index	(Cache Indexed Block Invalidate, Instruction cache) Disables the instruction cache line of the specified index. This instruction can be used in cases such as when the entire memory cache is initialized by software.
1000000	Instruction	CFALI	Address	(Cache Fetch And Lock, Instruction cache) Loads the data from the specified address and stores it in the instruction cache. At this time, the corresponding cache line is locked. If the data at the specified address is already stored in the instruction cache, this instruction only locks the cache line. If the data at the specified address is already stored in the instruction cache and the corresponding cache line is locked, no processing is performed.
1100000	Instruction	CISTI	Index	(Cache Indexed Store, Instruction cache) Writes (stores) data from a system register to the instruction cache line of the specified index. The specifications of the data to be written and the system register depend on the specifications of the CPU core. For details, see the hardware manual of the product used.
1100001	Instruction	CILDI	Index	(Cache Indexed Load, Instruction cache) Reads (loads) data from the instruction cache line of the specified index to a system register. The specifications of the data to be read and the system register depend on the specifications of the CPU core. For details, see the hardware manual of the product used.
1111110	_	CLL	_	(Clear Load Link-bit) Functions as the CLL instruction.

Table	7.7	Cache	Operation



<Cache instruction>

PREF	F	Prefetch
	F	Prefetch
[Instruction format]	PREF prefop, [reg1]	
[Operation]	Executes the prefetch operation specified by prefop.	
[Format]	Format X	
[Opcode]	15 031 16 1101111111RRRRR PPPPP00101100000	
	PPPPP indicates prefop.	
[Flags]	CY — OV — S — Z — SAT —	
[Description]	Executes the prefetch operation specified by prefop on the word data of general-purpo register reg1 used as a 32-bit address.	se
[Supplement]	The prefetch instruction does not generate a privilege instruction exception in any CPU If the CPU being used does not have a cache, this instruction will not generate a reserv instruction exception and no processing is performed, in the same way as a NOP instru	ved
	Memory protection by the MMU does not generate memory protection exceptions (pri or access permission violation exceptions) during prefetch operations. In such a case, t prefetch operation is implicitly ignored and no processing is performed, in the same we NOP instruction.	the
	CAUTION	
	Be aware that even after the prefetch instruction has finished executing, a prefetch ope might not necessarily have been performed.	ration



prefop	Target	Processing	Cache Specification	Operation
00000	Instruction	PREFI	Address	(Prefetch Instruction cache) Stores the data at the specified address in the instruction cache. If the data at the specified address is already stored in the instruction cache no processing is performed.

CAUTION

The size of the data prefetched in one prefetch operation depends on the specifications of the CPU core.



7.4 Floating-Point Instructions

7.4.1 Instruction formats

All floating-point instructions are in 32-bit format.

When an instruction is actually saved to memory, it is placed as shown below.

- Lower part of instruction format (including bit 0) \rightarrow Lower address side
- Higher part of instruction format (including bit 15 or bit 31) \rightarrow Upper address side

(1) Format F:I

The 32-bit long floating-point instruction format includes a 6-bit opcode field, 4-bit subopcode field, three fields that specify general-purpose registers, a 3-bit category field, and a 2bit type field.

15	11	10		5	4		0	31		27	26				16
r	eg2		opcode			reg1			reg3				sub-	opcode	



7.4.2 Overview of Floating-Point Instructions

Floating-point instructions are divided into single-precision instructions (single) and double-precision instructions (double), and include the following instructions (mnemonics).

(1) Basic operation instructions

- ABSF.D: Floating-point Absolute Value (Double)
- ABSF.S: Floating-point Absolute Value (Single)
- ADDF.D: Floating-point Add (Double)
- ADDF.S: Floating-point Add (Single)
- DIVF.D: Floating-point Divide (Double)
- DIVF.S: Floating-point Divide (Single)
- MAXF.D: Floating-point Maximum (Double)
- MAXF.S: Floating-point Maximum (Single)
- MINF.D: Floating-point Minimum (Double)
- MULF.S: Floating-point Multiply (Single)
- NEGF.D: Floating-point Negate (Double)
- NEGF.S: Floating-point Negate (Single)
- RECIPF.D: Reciprocal of a floating-point value (Double)
- RECIPF.S: Reciprocal of a floating-point value (Single)
- RSQRTF.D: Reciprocal of the square root of a floating-point value (Double)
- RSQRTF.S: Reciprocal of the square root of a floating-point value (Single)
- SQRTF.D: Floating-point Square Root (Double)
- SQRTF.S: Floating-point Square Root (Single)
- SUBF.D: Floating-point Subtract (Double)
- SUBF.S: Floating-point Subtract (Single)

(2) Extended basic operation instructions

- FMAF.S: Floating-point fused-multiply-add (Single)
- FMSF.S: Floating-point fused-multiply-subtract (Single)
- FNMAF.S: Floating-point fused-negate-multiply-add (Single)
- FNMSF.S: Floating-point fused-negate-multiply-subtract (Single)

(3) Conversion instructions

- CEILF.SUW: Floating-point convert single to unsigned-word, round toward positive (Single)
- CEILF.DW: Floating-point Convert Double to Word, round toward positive (Double)
- CEILF.DUL: Floating-point Convert Double to Unsigned-Long, round toward positive (Double)
- CEILF.DUW: Floating-point Convert Double to Unsigned-Word, round toward positive (Double)
- CEILF.SL: Floating-point Convert Single to Long, round toward positive (Single)
- CEILF.SW: Floating-point Convert Single to Word, round toward positive (Single)
- CEILF.SUL: Floating-point Convert Single to Unsigned-Long, round toward positive (Single)
- CEILF.SUW: Floating-point Convert Single to Unsigned-Word, round toward positive (Single)
- CVTF.DL: Floating-point convert double to long (Double)
- CVTF.DS: Floating-point convert double to single (Double)
- CVTF.DUL: Floating-point convert double to unsigned-long (Double)
- CVTF.DUW: Floating-point convert double to unsigned-word (Double)
- CVTF.DW: Floating-point convert double to long (Double)
- CVTF.LD: Floating-point convert long to double (Double)
- CVTF.LS: Floating-point convert long to single (Single)
- CVTF.SD: Floating-point convert single to double (Double)
- CVTF.SL: Floating-point convert single to long (Single)
- CVTF.SUL: Floating-point convert single to unsigned-long (Single)
- CVTF.SUW: Floating-point convert single to unsigned-word (Single)
- CVTF.SW: Floating-point convert single to long (Single)
- CVTF.ULD: Floating-point convert unsigned-long to double (Double)
- CVTF.ULS: Floating-point convert unsigned-long to single (Single)
- CVTF.UWD: Floating-point convert unsigned-word to double (Double)
- CVTF.UWS: Floating-point convert unsigned-word to single (Single)
- CVTF.WD: Floating-point convert word to double (Double)
- CVTF.WS: Floating-point convert word to single (Single)
- FLOORF.DL: Floating-point convert double to long, round toward negative (Double)
- FLOORF.DW: Floating-point convert double to long, round toward negative (Double)
- FLOORF.DUL: Floating-point convert double to unsigned-long, round toward negative (Double)
- FLOORF.DUW: Floating-point convert double to unsigned-word, round toward

negative (Double)

- FLOORF.SL: Floating-point convert single to long, round toward negative (Single)
- FLOORF.SW: Floating-point convert single to long, round toward negative (Single)
- FLOORF.SUL: Floating-point convert single to unsigned-long, round toward negative (Single)
- FLOORF.SUW: Floating-point convert single to unsigned-word, round toward negative (Single)
- TRNCF.DL: Floating-point convert double to long, round toward zero (Double)
- TRNCF.DUL: Floating-point convert double to unsigned-long, round toward zero (Double)
- TRNCF.DUW: Floating-point convert double to unsigned-word, round toward zero (Double)
- TRNCF.DW: Floating-point convert double to long, round toward zero (Double)
- TRNCF.SL: Floating-point convert single to long, round toward zero (Single)
- TRNCF.SUL: Floating-point convert single to unsigned-long, round toward zero (Single)
- TRNCF.SUW: Floating-point convert single to unsigned-word, round toward zero (Single)
- TRNCF.SW: Floating-point convert single to long, round toward zero (Single)
- CVTF.HS: Floating-point convert half to single (Single)
- CVTF.SH: Floating-point convert single to half (Single)

(4) Comparison instructions

- CMPF.S: Compare floating-point values (Single)
- CMPF.D: Compare floating-point values (Double)

(5) Conditional move instructions

- CMOVF.S: Floating-point conditional move (Single)
- CMOVF.D: Floating-point conditional move (Double)

(6) Condition bit transfer instruction

• TRFSR: Transfers specified CC bit to Zero flag in PSW (Single)



7.4.3 Conditions for Comparison Instructions

Floating-point comparison instructions (CMPF.D and CMPF.S) perform two floating-point data compare operations. The result is determined based on the comparison condition contained in the data and code. **Table 7.9** lists the mnemonics for conditions that can be specified by comparison instructions.

The comparison instruction result is transferred by the TRFSR instruction to the Z flag of PSW (program status word), and when performing a conditional branch, the condition logic is inverted and then can be used. **Table 7.10** shows logic inversion based on the true/false status of conditions. In a 4-bit condition code for a floating-point comparison instruction, the condition is specified in the "True" column of the table. The conditional branch instruction BT performs a branch when the comparison result is true, while BF performs a branch when the result is false.

Table 7.9 List of Conditions for Comparison Instructions

Mnemonic	Definition	Inverte	d Logic
F	Always false	(T)	Always true
UN	Unordered	(OR)	Ordered
EQ	Equal	(NEQ)	Not equal
UEQ	Unordered or equal	(OLG)	Ordered and less than or greater than
OLT	Ordered and less than	(UGE)	Unordered or greater than or equal to
ULT	Unordered or less than	(OGE)	Ordered and greater than or equal to
OLE	Ordered and less than or equal to	(UGT)	Unordered or greater than
ULE	Unordered or less than or equal to	(OGT)	Ordered and greater than
SF	Signaling and false	(ST)	Signaling and true
NGLE	Not greater than, not less than, and not equal to	(GLE)	Greater than, less than, or equal to
SEQ	Signaling and equal to	(SNE)	Signaling and not equal to
NGL	Not greater than and not less than	(GL)	Greater than or less than
LT	Less than	(NLT)	Not less than
NGE	Not greater than and not equal to	(GE)	Greater than or equal to
LE	Less than or equal to	(NLE)	Not less than and not equal to
NGT	Not greater than	(GT)	Greater than



	100	le 7.10	Demitions			ogical inversions	
				Bit Definition o	f Condition Code fcond	3:0)	
Mnemonic		on Code ond	Less than	Equal to	Unordered	Invalid operation exception occurs when unordered	Inverted
(True)	Decimal	Binary	fcond(2)	fcond(1)	fcond(0)	fcond(3)	(False)
F	0	0b0000	F	F	F	No	(T)
UN	1	0b0001	F	F	Т	No	(OR)
EQ	2	0b0010	F	Т	F	No	(NEQ)
UEQ	3	0b0011	F	Т	т	No	(OLG)
OLT	4	0b0100	Т	F	F	No	(UGE)
ULT	5	0b0101	Т	F	Т	No	(OGE)
OLE	6	0b0110	Т	Т	F	No	(UGT)
ULE	7	0b0111	Т	Т	Т	No	(OGT)
SF	8	0b1000	F	F	F	Yes	(ST)
NGLE	9	0b1001	F	F	Т	Yes	(GLE)
SEQ	10	0b1010	F	Т	F	Yes	(SNE)
NGL	11	0b1011	F	Т	Т	Yes	(GL)
LT	12	0b1100	Т	F	F	Yes	(NLT)
NGE	13	0b1101	Т	F	Т	Yes	(GE)
LE	14	0b1110	Т	Т	F	Yes	(NLE)
NGT	15	0b1111	Т	Т	Т	Yes	(GT)

Table 7.10 Definitions of Condition Code Bits and Their Logical Inversions



7.4.4 Floating-Point Instruction Set

This section describes the following items in each instruction (based on alphabetical order of instruction mnemonics).

- Instruction format: Indicates how the instruction is written and its operand(s) (symbols are listed in **Table 7.11**).
- Operation: Indicates the function of the instruction. (symbols are listed in **Table 7.12**).
- Format: Indicates the instruction format (see Section 7.4.1, Instruction formats).
- Opcode: Indicates the instruction opcode in bit fields (symbols are listed in **Table 7.13**).
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.

Table 7.11Instruction Format

Symbol	Explanation
reg1	General-purpose register
reg2	General-purpose register
reg3	General-purpose register
reg4	General-purpose register
fcbit	Specifies the bit number of the condition bit that stores the result of a floating- point comparison instruction.
imm ×	× bit immediate data
fcond	Specifies the mnemonic or condition code of the comparison condition of a comparison instruction (for details, see Section 7.4.3, Conditions for Comparison Instructions).



Symbol	Explanation
\leftarrow	Assignment (input for)
GR [a]	Value stored in general-purpose register a
SR [a, b]	Value stored in system register (RegID = a, SeIID = b)
result	Result is reflected in flag.
==	Comparison (true upon a match)
+	Add
_	Subtract
	Bit concatenation
×	Multiply
÷	Divide
abs	Absolute value
ceil	Rounding in +∞ direction
compare	Comparison
cvt	Converts type according to rounding mode
floor	Rounding in –∞ direction
max	Maximum value
min	Minimum value
neg	Sign inversion
round	Rounding to closest value
sqrt	Square root
trunc	Rounding in zero direction
fma(a, b, c)	Result of multiplying a and b and then adding c
fms(a, b, c)	Result of multiplying a and b and then subtracting c

Table 7.12 Operations

Table 7.13 Opcodes

Symbol	Explanation
R	Single bit data of code specifying reg1
r	Single bit data of code specifying reg2
w	Single bit data of code specifying reg3
W	Single bit data of code specifying reg4
I	Single bit data of immediate data (indicates higher bit of immediate data)
i	Single bit data of immediate data
fff	3-bit data that specifies the bit number (fcbit) of the condition bit that stores the result of a floating-point comparison instruction
FFFF	4-bit data corresponding to the mnemonic or condition code (fcond) of the comparison condition of a comparison instruction

ABSF.D																	Absolu				
[Instruction format]		ABSF.D	reg2,	reg3								Floa	ting-j	ooin	t abs	olute	value	(doub	<u>le pr</u>	ecisio	on)
[Operation]		reg3 ← a	abs (reg	g2)																	
[Format]		Format I	F:I																		
[Opcode]																					
	15 r r	1 rr0	1 10 1 1	1 1	1 1	1		0 0		31 w	w	w w		26 1	25 0 (22 21 1 0		1 0	17 0	16 0
		reg2									re	eg3			cate	gory	type	s	ub-ol	p	
[Description]		This inst contents register j	of the	registe	r pair	spe	ecifie	d by	ger	nera	al-pu	irpos	se re								
[Floating-point operation exceptions]]	None																			
[Supplement]		A subno	rmal in	ıput wi	ll not	be	flush	ed ev	/en	if t	he F	'S bi	t of	the	FPS	R re	gister	is 1.			



ABSF.S																	Flo	oatir	וg-p	oint	Abso	ute	Valu	ıe (S	Singl	le)
															Flo	pating	J-po	int a	abso	lute	e value	: (sir	ngle	prec	cisio	n)
[Instruction format]		ABSF.S	S 1	reg2,	reg3																					
[Operation]		reg3 ←	- al	os (re	g2)																					
[Format]		Format	: F:	I																						
[Opcode]	15		11	10			5	4				0	31			27	26	25		23	22 21	1 20			17 ·	16
		rr			1	1	1 1	1	0	0	0			w	w	w w			0		1 0		1			0
		reg2												re	eg3			са	itego	ory	type		sub	o-op		
[Description]		This in content													-	-										
[Floating-point operation exceptions]	None																								
[Supplement]		A subn	orı	nal ir	nput ⁻	will	l not	be f	flus	hed	ev	en	if t	he F	'S t	oit of	the	FF	PSR	reg	gister	is 1				



ADDF.D																Flo	ating-	ooint	Add	l (Dou	ble)
														FI	oating	-poii	nt add	(dou	ble	precis	ion)
[Instruction format]	AD	DF.D	reg1,	reg2, 1	reg3																
[Operation]	reg	$3 \leftarrow r_0$	eg2 + r	egl																	
[Format]	For	mat F	:I																		
[Opcode]																					
	15 r r r		10 1 1	1 1	5 1 1	4 R	RI	R R		31 w	ww	v v		26 1	25 0 0		22 2 1 1		0		0 16
	regi							g1	-		re		-		cate				sub		
[Description]	spe con the	cified tents o regist e of in	uction by gen of the r er pair nfinite	neral-p registe specif	ourpos r pair fied by	e re speo 7 ge	giste cifie nera	er reg d by g l-pur	1 w gen pos	vith era e re	the l-pur	dou pos er re	ble-j e reg 2g3.	pre gist Th	cisior er reg e ope	n flo g2, a ratio	ating and stoon is o	-poir ores exec	nt fo the uteo	ormat resul	t in f it
[Floating-point operation exceptions	5] Inv Ine Ove	alid op kact ex erflow	mented peratio xceptic excep w exce	n exce on (I) tion ((eption D)		-	n (E)													



RH850G3MH Software

[Operation result]

reg2(B) reg1(A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal				-				
–Normal								
+0		A +	в			_∞		
-0								
+ ∞					+∞	Q-NaN [V]		
_ ∞			×		Q-NaN [V]	_∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.



ADDF.S	Floating-point Add (Single)
	Floating-point add (single precision)
[Instruction format]	ADDF.S reg1, reg2, reg3
[Operation]	$reg3 \leftarrow reg2 + reg1$
[Format]	Format F:I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16 r
	reg2 reg1 reg3 category type sub-op
[Description]	This instruction adds the single-precision floating-point format contents of general-purpose register reg1 with the single-precision floating-point format contents of general-purpose register reg2, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.
[Floating-point operation exceptions	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I) Overflow exception (O) Underflow exception (U)



RH850G3MH Software

[Operation result]

reg2(B)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
reg1(A)	TNOrmai	-Normai	ŦŪ	-0	- ~	1	Q-Nan	5-main
+Normal								
–Normal		A +	P			_∞		
+0		A 1	Ъ					
-0								
+∞					+∞	Q-NaN [V]		
_ ∞		-9	×		Q-NaN [V]	-8		
Q-NaN							Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.



CEILF.DL		
[Instruction format]	Conversion to fixed-point format (double precisio CEILF.DL reg2, reg3	<u>n)</u>
[Operation]	$reg3 \leftarrow ceil reg2 (double \rightarrow long-word)$	
[Format]	Format F:I	
[Opcode]	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 7 r	16 0
[Description]	 This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg2 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3. The result is rounded in the +∞ direction regardless of the current rounding mode. When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2⁶³ - 1 to -2⁶³, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources. Source is a positive number or +∞: 2⁶³ - 1 is returned. Source is a negative number, not-a-number, or -∞: -2⁶³ is returned. 	l
[Floating-point operation exceptions	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)	

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CEILF.DL	JL					Flo	batin	g-p	oint	Cor							-		•	round							ŕ
											(Conv	ver	rsior	ı to	uns	igne	ed fi	xed	-point	form	nat ((dou	iple b	oreci	sio	n)
[Instruction format]		CEILF	.DI	JL re	eg2,	reg3																					
[Operation]		reg3 ←	– ce	eil reg	g2 (d	loubl	$e \rightarrow$	• un	nsign	ned	l lo	ng-	w	ord)	1												
[Format]		Forma	t F:	I																							
[Opcode]																											
	15		11	10			5	4				0	3′	1			27	26	25	23	22	21	20		1	7 ′	16
	r :	r r r	0	1 1	1	1 1	1	1	0	0	1	0	W	w	W	W	0	1	0	0 0	1	0	1	0	1 (C	0
		reg2						Ī							reg	13			ca	tegory	ty	/pe		sub-	ор		
[Description]			iste , an sult the is ou ed. lid r is s, a our	er pain d sto is ro source utside opera set a ccord	r spe res t unde ce op e the tion s an ling a pos	ecifie he re ed in beran rang exce inval to dif	d by sult the d is e of eptic lid o ffere	in $+\infty$ inf 2^6 ons open	ener the dir init init are ratic es a er o	ral-j reg rect e, r 1 to nor 2 nor 2	pur gist ion not- o 0 t er and ong ide	pos er p reg a-n , an nabl no sou the	se : pai gai ur llec ez urc e ra	regi r sp rdle mbe EEE 1, th xcep xes. ang	ste eci ss r, c 75 e p otic e o	fied of the or no 4-d oreson on o	g2 1 l by he c egat efin erva	to u ge curr tive ned atic urs. 1 t	insi ener ent e nu inv on b Th	gned round mber ralid c it (bit e retu	64- rpos ding , or oper 4) rn v	bit se r g m wh catio of t	fixe regist node nen on e the	ed-poster i e. the r excej FPS	roun roun ptio R rs as	3. nde n i	ed s
[Floating-point operation exceptions]	Unimp Invalid Inexac	l op	eratio	on ez	xcept				on ((E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CEILF.DU	W	Floating	g-poin	t Conv						rd, round					,
					Conv	/ersio	n to uns	igne	d fix	ed-point t	format	(dou	ble pre	cisi	on)
[Instruction format]	CEILF.DUW reg2, 1	reg3													
[Operation]	reg3 ← ceil reg2 (do	uble \rightarrow	unsi	gned [,]	word)									
[Format]	Format F:I														
[Opcode]															
· 	5 11 10	5	4		0	31		27	26	25 23	22 21	20		17	16
	rrr 0 1 1 1 1	1 1	1 (0 0	1 0	W W	w w	W	1	0 0 0	1 0	1	0 0	0	0
	reg2						reg3			category	type		sub-op)	
[Description]	This instruction arith the register pair spec format, and stores th The result is rounded When the source ope result is outside the r detected. If invalid operation e register is set as an in follows, according to • Source is a posi • Source is a nega	ified by e result l in the erand is ange of exception valid of different tive num	y gene in ge $+\infty$ d infin 2^{32} ons ar operatences mber	eral-p neral- irectio ite, no - 1 to e not ion an amon outsio	urpos -purp on reg ot-a-n 0, an enabl nd no ig sou de the	e reg ose re gardle umbe IEEI led, th exce urces. e rang	ister re egister ess of th er, or no E754-d he prese ption o ge of 2 ⁶	g2 1 reg2 he c egat efin erva occu	to un 3. eurre tive ed i ntior rs. 7	nsigned i ent round number, nvalid o n bit (bit The retur 0, or +o	32-bit ling m or wh perati 4) of m valu	fixe node nen 1 on e the 1 ue d	ed-poi the rot xcept FPSR	nt und ion as	ed is
[Floating-point operation exceptions]	Unimplemented oper Invalid operation exe Inexact exception (I)	eption	-	ion (I	E)										

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CEILF.DV	V									Float	ing	-poi	nt (Con							round							
															C	onv	ers	on	to fi	xed	-point	for	mat	(dol	ible p	oreci	SIO	n)
[Instruction format]		CE	EILI	F.D'	W re	g2, r	eg3																					
[Operation]		reg	<u>3</u> 3 ↔	- c	eil reg	g2 (d	ouble	$e \rightarrow$	W	ord)																		
[Format]		Fo	rma	at F:	:I																							
[Opcode]																												
	15				10			5	4					31				27		25			2 21			1	7 ^	
	rı	: r	r	0	1 1	1	1 1	1	0	0	0	1	0	W	W	W	W	W	1	0	0 0) -	1 0	1	0	0 0)	0
		reg	<u>j</u> 2													reg	3			ca	tegory	/	type		sub-	∙ор		
[Description]		the sto Th Wl ran If i reg	e reg pres ne re hen nge invæ giste llow	gista the esuli the of 2 alid er is vs, a	uction result t is ro 2^{31} - opera s set a accord rce is rce is	r spe t in g unde ce op 1 to ation s an ling a pos	ecified gener ed in perand -2^{31} , exce inval to dif	d by ral-p the d is i an l eptio lid o ffere	y ge urj + ∞ inf IEI ons pe: enc nb	enera pose dire inite EE75 are 1 ratio es ar er or	1-p regetion or 4-c not not not	ourp gist on not defi ena nd ng s ∞: 2	er 1 reg t-a- ine abl no sou 2 ³¹	e reg reg garconu d in ed, ex irce - 1	egii 3. dle: mb nva nva , th cep es.	ster ss c er, ilid e p otio	of the or voor voor voor voor voor voor voor	g2 the construction of the second sec	to 3 curr en t tior atio	32-t rent he i n ex on b Th	roun round cepti it (bi e retu	din led on t 4	-poi ng m resi is d) of	nt f node ult i leteo the	forma e. s out cted. FPS	at, a tside R	nd e th	
[Floating-point operation exceptions]	Inv	vali	d op	nente peratie xcepti	on ez	kcept			-	n (E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.
CEILF.SL	•												F	loati	ng-	ooir	nt	Co	onv	er	t Sir	ngle	to	Lo	ong	, rou	ind	tov	varo	d pc	sitiv	/e (Sinę	gle)
																			C	Co	nve	rsio	n t	o fi	xe	d-po	int	for	nat	(sir	igle	pre	cisi	on)
[Instruction format]		(CEI	LF.	SI	_ re	eg2	, re	g3	;																								
[Operation]		r	eg3	3 ←	· CI	eil 1	regi	2 (s	sin	gle	\rightarrow	lor	ng-v	wor	d)																			
[Format]		ł	Fori	mat	F	:I																												
[Opcode]	15				11	10					5	1				0	2	1				2	7 0	26	25		03	22	21	20			17	16
	r	r	r			1	1	1	1	1	1	4	0	0	1		Т		w	v	7 W		1		25 0	0		1		1		1		0
			eg2													-					g3					tego			pe		sub			
[Description]		۶ ۲ ۲ ۲ ۲ ۲	geno pair The Who rang If in regi	eral spe res en t ge o nval ster ows Sc	-p eci he f 2 id ; a	urp fiec t is sou 2 ⁶³ ope set accc	ose d by rou urce - 1 erat t as ordi is a	y ge und e op to ion an ng po	gis ene ed per -2 in zin to sit	tter eral in t cance 63 vali dif	reg -pu he l is an ptic id c fere	2 to rpc $+\infty$ inf IEI ons ope enc mb	o 6 ose o di init EE7 are rati es :	4-bi reg rect te of 754- e no con amc or +	t fi iste ion no def t er and ong ∞ :	xec r re re t-a ine iabl no sou 2 ⁶³	l-j ga -n ed le ur	po 3. ard iui d, ir d, exc ce - 1	lles mb the cep s. is	t fo ss er ilio e j otio	orm of f of or ores on etur	hat, the wh pera serv occ	ar cu ler ati vat ur:	nd urre n th on iou s. '	sto ent ne 1 ex n b Th	ng-j rou rour cour it (b e ret turn	the nde tic oit tur	e re ling ed r on i 4) rn v	g m esu s de of t	t in ode ilt i eteo the	the e. s ou cted	e re utsi I. SR	egis de t	ter
[Floating-point operation exceptions]	Ι	Unii Inva Inex	alid	oŗ	pera	itio	n e	xc				-	on	(E)																			

[Operation result]

reg2 (A)	+Normal	-Normal	+0	-0	+∞	8	Q-NaN	S-NaN
Operation result [exception]	A (int	æger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CEILF.SU	JL					Floati	ing	-poir	nt C	onv	rert	Si	ngle	e to	Uns	igne	ed-L	ong	, roun	d to	war	d p	osit	ive (Sing	gle)
											Cor	nve	ersic	on '	to un	sigr	ed	fixe	d-poin	t for	ma	t (si	ngle	e pre	ecisi	on)
[Instruction format]		CEILF.S	UL 1	reg2,	reg3																					
[Operation]		reg3 ← c	eil re	eg2 (single	$e \rightarrow 1$	uns	sign	ed	lor	ıg-v	wC	ord)													
[Format]		Format F	:I																							
[Opcode]																										
	15		10			5	4				0	1					26			3 22		1				16
	r	rrrr	1	1 1	1 :	1 1	1	0	0	1	0	v	/ W		w w	0	1	0	0 0	1	. 0	0	0) 1	0	0
		reg2												re	g3			са	itegory	/ t	уре		su	ıb-op)	
		the result The result When the result is o detected. If invalid register is follows, a	t is r sou outsio oper s set	round rce o de the ration as an	led in operar e rang n exco n inva	the nd is ge of eptio llid o	$+\infty$ inf 2^6 ons	o dir finit ⁵⁴ – are ratio	rect e, r 1 to no on a	ion not- o 0 t er and	reg a-n , an nabl	ga nui n II leo	rdle mbe EEF d, th xcej	ess er, E7	of t or n 54-d	he o ega lefir	tive tive	ent nu inv	t roun umber valid o pit (bi	idin r, or ope t 4)	r wl rati of	hen ion the	exc exc FF	cept PSR	ion	
		• Sour	rce is	s a po	ositiv	e nur	nb	er o	uts	ide	the	e r	ang	e (of 2 ⁶	54 _	1 t	0 0	, or +	∞: 2́	2 ⁶⁴	- 1	is	retu	ırne	ed.
		• Sour	rce is	s a ne	egativ	e nu	mb	ber,	not	-a-	nur	nt	ber,	or	-∞:	0 i	s re	tur	ned.							
[Floating-point operation exceptions	5]	Unimpler Invalid oj Inexact e	perat	tion e	excep			-	on ((E)																
[Operation result]																										

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CEILF.SU	JN	V				F	loati	ng-	poir	nt C	onv	ert	Sir	igle	to	Uns	sign	ed-	Wo	rd,	round	d tov	ма	rd p	osi	tive	(Sin	gle)
												Cor	ive	ersio	n	to u	nsig	ne	d fix	ed	l-point	for	ma	ıt (si	ng	le pi	recis	ion)
[Instruction format]		CEI	LF.SU	JW	reg2,	reg3																						
[Operation]		reg3	- ← c	eil re	g2 (s	ingle	\rightarrow	un	sign	ned	wo	rd)																
[Format]		Form	nat F	I																								
[Opcode]	45		44	10			-	4				0	0.0				0.	7 0	~ ~		00	20		1 00			4-	10
	15			10	1 1	1 1	1	4	0	0	1		31						62 L		0 0		0	1 20		0 0		16
	r	r r	r r	1.		1 1	1		0	0	Ţ	0	W	W		w v	7 W	<u> </u>								0 0	0 0	0
		reg2													re	eg3			(cat	egory	ty	pe		S	ub-o	р	
[Description]		the r form The Whe resu detec If in regis follo	valid ster is ows, a Sour	er pa nd sto t is ro sour utsid oper set a ccor cce is	ir spe pres t punde ree op le the ation as an ding a pos	ecified he resed in berand rang exce inval to dif	d by sult the d is e of ptio id o ffere	y ge in $+\infty$ inf 2^3 2^3 ons ope enc mb	ener gen dir finit ² _ are ratio es a er o	ral- nera rect e, 1 1 t no on umc	pur al-p tion not- o 0 t er and ong	pos urp reg a-n , an nabl no sou the	se gan nur lec e ra	regi e re rdle mbe EEF l, th xcej xes. ang	ist egi ess er, E7 ne pt	er r ister of or 1 54- pre- ion	eg2 the nega defi serv occ	to g3. cu ati ne vat urs	un rre ve i d ii ion s. T to	nt nu nva bi The 0,	gned round mber, alid o it (bit e retur	32- ding or per 4)	g n w cati	t fix nod hen ion	e. th ex	l-po ne ro cep PSF fers	int ound tion as	led is
[Floating-point operation exceptions]	Inva	mpler ilid op act ex	berati	ion ex	kcepti			-	on	(E)																	

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CEILF.SV	V									Floa	atin	ıg-po	oin	t Co	onv	ert S	ingl	e to	W	ord	, rour	d to	wa	rd p	ositi\	/e (S	Sing	le)
															(Conv	rersi	on	o f	ixec	d-poir	t fo	rma	ıt (si	ngle	pre	cisio	on)
[Instruction format]		CE	EILI	F.SV	V reg	g2, 1	reg3																					
[Operation]		reg	<u></u> 3 ←	— c	eil re	g2 (:	single	$e \rightarrow 1$	wo	ord)																		
[Format]		For	rma	at F:	I																							
[Opcode]																												
	15				10			5	4				-	31			2	27	26	25	2	3 22	22	1 20	1		17	16
	гı	: r	r	r	1 1	. 1	1 1	1 1	0	0	0	1	0	W	W	W	W	w	1	0	0 0	1	. 0) 0	0	0	0	0
		reg	J2													reg3				cat	tegory	/ t	ype	:	sub	о-ор		
[Description]		ger pur Th Wh ran If i reg	nera rpos he re hen nge inva giste llow S	al-p se r esult the of 2 alid er is vs, a	urpos egiste t is ro sourc 2 ³¹ – opera set a ccoro ce is	er re ound ce of 1 to ation is an ding a po	ithme gister g3. ed in peran -2^{31} n exce i inva to di ositive egativ	r reg the d is , an leptio llid o ffere e nur	2 to +∞ inf IEF per enco mbo	o 32- o dire inite EE75 are 1 ration res an	bit ecti or 4-c not n a not	t fix on : not defi ena nd : ng s 2	reg -a- abl no sou	-pc garc num d in ed, ince - 1	dle: mb nva th cep es.	t for ss o er, o er, o ilid e pr otion retu	mat f the or w ope eser n oc	, a e cu he rat va cui	nd urro n th on tion rs.	sto ent he r ex n b The	roun counc cepti it (bi e retu	he r din led ion t 4) urn	resung r res is (ult in mod sult i dete `the	n ge e. is ou cted FPS	enera utsic 1. SR	al- le t	
[Floating-point operation exceptions]	Inv	valio	d op		on e	oerati excep (I)			-	n (.	E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CMOVF.[כ											Flo				move (
[Instruction format]		CMOVF.	D fcbit, reg1, re	eg2	, re	eg3														,
[Operation]		reg3 < else	.CCn == 1) ther - reg1 - reg2	1																
		Note	1. n = fcbit																	
[Format]		Format F	:I																	
[Opcode]																				
	15	11	10		4				31			27		25	23	22 21	20		17	16
	r :	rrr O	1 1 1 1 1	1	R	R R	R	0	W		W	0	1	0 (0 (0 0	1	f f	f	0
		reg2				reg	g1			reg3	3* ¹			cate	gory	type		sub-op		
		Caution 1.	reg3: wwww! = 0 wwww ≠ 0000 (d iit: fff		ot s	et reg	3 to r	0)												
[Description]		from the these bits	CC(7:0) bits of register pair spe are false (0), da fied by reg3.	cifi	ed	by re	g1 is	sto	ore	ed in tl	ne re	egis	ter	pair	spec	cified l	oy re	eg3. V	Vhe	n
[Floating-point operation exceptions	5]	None																		
[Supplement]		A subnor	mal input will n	ot t	be f	lushe	d ev	en	if	the FS	bit	oft	the	FPS	R re	gister	is 1.			
		CAUTION	1																	
		Do not se	t reg3 to r0.																	



<Floating-point condition instruction >

CMOVF.S	2								Float	ing-po	int C	onditio	nal N	love (S	ingle	?)
										Condit	tiona	l move	(sing	gle prec	isio	1)
[Instruction format]	CMOVF.	S fcbit, reg1, r	eg2,	reg3												
[Operation]	reg3 else	.CCn == 1) the ← reg1 ← reg2	en													
	Note	1. n = fcbit														
[Format]	Format F:	Ι														
[Opcode]	15 11	10	5	4	0	31		~	7 26	25	22	22.24	20		17 1	6
	15 11 r r r r r	1 1 1 1 1		4 R R R		w	w w		7 26 v 1			22 21 0 0			17 1 f)
	reg2			reç	J1		reg3	*1		cate	gory	type	s	ub-op		
	Caution 1.	reg3: wwwww! = wwwww ≠ 0000 it: fff		not set r	eg3 to r	0)										
[Description]		CC(7:0) bits c is stored in re			-	-		-			-				dat	a
[Floating-point operation exceptions	None]															
[Supplement]	A subnorn	nal input will	not b	e flushe	d even	ift	he FS	bit o	f the	e FPSI	R reg	gister	is 1.			
		t reg3 to r0.														-
		-														



CMPF.D

Compare floating-point values (Double)

loating-point comparison (double precision)

[Instruction format]	CMPF.D	fcond, reg2, reg	,1, fo	cbit													
	CMPF.D	fcond, reg2, reg	<u>,</u> 1														
[Operation]	result.le result.e result.u if fconc	eg1) or isNaN(r ess $\leftarrow 0$ qual $\leftarrow 0$ nordered $\leftarrow 1$ l[3] == 1 then operation except			ected.												
	result.	less \leftarrow reg2 < re equal \leftarrow reg2 == unordered \leftarrow 0	-	g1													
	endif																
		n ← (fcond[2] & [0] & result.unor			(fco	ond	[1] &	resul	t.eq	ual)						
	Note	1. n: fcbit															
[Format]	Format F:	I															
[Opcode]																	
	15 11		5 4) 3					25	23	22 21				7 16
	rrrr O	1 1 1 1 1	1 1	RRR	R () () F	FF	F	1	0	0 0	0 1	1	fi	f	0
	reg2			reg	1						cat	tegory	type		sub-c	р	
	Note: fcoi fcbi	nd: FFFF t: fff															



[Description]

This instruction compares the double-precision floating-point format contents of the register pair specified by general-purpose register reg2 with the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, based on the condition "fcond", and sets the result (1 if true, 0 if false) to the condition bits (the CC(7:0) bits: bits 31 to 24) in the FPSR register specified by fcbit in the opcode. If fcbit is omitted, the result is set to the CC0 bit (bit 24).

For description of the comparison condition "fcond" code, see **Table 7.14, Comparison Conditions**.

If one of the values is not-a-number, and the MSB of the comparison condition "fcond" has been set, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are enabled, the comparison result is not set and processing is passed to the exception.

If the enable bits are not set, no exception occurs, and the preservation bit (bit 4) of the FPSR register is set, then the comparison result is set to the CC(7:0) bits of the FPSR register. When SignalingNaN (S-NaN) is acknowledged as an operand value in a floating-point instruction (including a comparison), it is regarded as an invalid operation condition. When using only S-NaN but also QuietNaN (Q-NaN) for a comparison that is an invalid operation, it is simpler to use a program in which any NaN results in an error. In other words, there is no need to insert code that checks for Q-NaN that would result in an unordered result. Instead, the exception handling system should perform error processing when an exception occurs after detecting an invalid operation. The following shows a comparison that checks for equivalence of two numerical values and triggers an error when an unordered result is detected.

Compar Conditio				Detection of invalid operation
	fcond	Definition	Description	exception by unordered
F	0	FALSE	Always false	No
UN	1	Unordered	One of reg1 and reg2 is not-a-number	No
EQ	2	reg2 = reg1	Ordered (both reg1 and reg2 is not not-a-number) and equal	No
UEQ	3	reg2 ? = reg1	Unordered (at least, one of reg1 and reg2 is not-a-number) or equal	No
OLT	4	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	No
ULT	5	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
OLE	6	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	No
ULE	7	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
SF	8	FALSE	Always false	Yes
NGLE	9	Unordered	One of reg1 and reg2 is not-a-number	Yes
SEQ	10	reg2 = reg1	Ordered (both reg1 and reg2 are not not-a-number) and equal	Yes
NGL	11	reg2 ? = reg1	Unordered (one of reg1 and reg2 is not-a-number) or equal	Yes
LT	12	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	Yes
NGE	13	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than	Yes
LE	14	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	Yes
NGT	15	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	Yes

Table 7.14 Comparison Conditions

Note: ?: Unordered (invalid comparison)



```
# When explicitly testing Q-NaN
                              CMPF.D
                                             OLT, r12, r14, 0 # Check if r12 < r14
                                             UN, r12, r14, 1 # Check if unordered
                              CMPF.D
                              TRFSR
                                             0
                              ΒT
                                                                   # If true, go to L2
                                             T<sub>1</sub>2
                              TRFSR
                                             1
                              BT
                                             ERROR
                                                                   # If true, go to error processing
                        # Enter code for processing when neither unordered nor r12 < r14
                              L2:
                        # Enter code for processing when r12 < r14
                              ...
                        # When using a comparison to detect Q-NaN
                                            LT, r12, r14, 0
                              CMPF.D
                                                                   # Check if r12 ?< r14
                              TRFSR
                                             0
                              ΒT
                                             L2
                                                                   # If true, go to L2
                        # Enter code for processing when not r12 < r14
                              L2:
                        # Enter code for processing when r12 < r14
                              ...
[Floating-point
                        Invalid operation exception (V)
operation exceptions]
[Supplement]
                        A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.
```



Section 7 INSTRUCTION

RH850G3MH Software

[Operation result]

[Condition code (fcond) = 0 to 7]

reg1(B) reg2(A)	+Normal	-Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN								
±Normal	ci			.												
±0		Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)														
±∞																
Q-NaN				Unordered												
S-NaN				Unorde	ered [V]											

[Condition code (fcond) = 8 to 15]

reg1(B) reg2(A)	+Normal	-Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN
±Normal	ä		. "	.				
±0		s result of co parison cond						
±∞			()					
Q-NaN				Linorda	ered [V]			
S-NaN				UNDIDE				

Note: [] indicates an exception that must occur.



CMPF.S

Compare floating-point values (Single)

Floating-point comparison (single precision)

[Instruction format]	CMPF.S fcond, reg2, reg1, fcbit CMPF.S fcond, reg2, reg1
[Operation]	if isNaN(reg1) or isNaN(reg2) then result.less $\leftarrow 0$ result.equal $\leftarrow 0$ result.unordered $\leftarrow 1$ if fcond[3] == 1 then Invalid operation exception is detected. endif else result.less \leftarrow reg2 < reg1 result.equal \leftarrow reg2 == reg1 result.unordered $\leftarrow 0$ endif FPSR.CCn \leftarrow (fcond[2] & result.less) (fcond[1] & result.equal) (fcond[0] & result.unordered)
	Note 1. n: fcbit
[Format]	Format F:I

[Opcode]



[Description]

This instruction compares the single-precision floating-point format contents of generalpurpose register reg2 with the single-precision floating-point format contents of generalpurpose register reg1, based on the comparison condition "fcond", then sets the result (1 if true, 0 if false) to the condition bits (the CC(7:0) bits: bits 31 to 24) in the FPSR register specified by fcbit in the opcode. If fcbit is omitted, the result is set to the CC0 bit (bit 24). For description of the comparison condition "fcond" code, see **Table 7.15, Comparison Conditions**.

If one of the values is not-a-number, and the MSB of the comparison condition "fcond" has been set, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are enabled, the comparison result is not set and processing is passed to the exception.

If the enable bits are not set, no exception occurs, and the preservation bit (bit 4) of the FPSR register is set, then the comparison result is set to the CC(7:0) bits of the FPSR register. When SignalingNaN (S-NaN) is acknowledged as an operand value in a floating-point instruction (including a comparison), it is regarded as an invalid operation condition. When using only S-NaN but also QuietNaN (Q-NaN) for a comparison that is an invalid operation, it is simpler to use a program in which any NaN results in an error. In other words, there is no need to insert code that explicitly checks for Q-NaN that would result in an unordered result. Instead, the exception handling system should perform error processing when an exception occurs after detecting an invalid operation. The following shows a comparison that checks for equivalence of two numerical values and triggers an error when an unordered result is detected.

Comparis Condition	fcond I 0 F 0 F 1 I 2 r 3 r 4 r 5 r 6 r 7 r 8 F 9 I 10 r 11 r 12 r			Detection of invalid
	fcond	Definition	Description	operation exception by unordered
F	0	FALSE	Always false	No
UN	1	Unordered	One of reg1 and reg2 is not-a-number	No
EQ	2	reg2 = reg1	Ordered (both reg1 and reg2 is not not-a-number) and equal	No
UEQ	3	reg2 ? = reg1	Unordered (at least, one of reg1 and reg2 is not-a-number) or equal	No
OLT	4	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	No
ULT	5	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
OLE	6	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	No
ULE	7	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
SF	8	FALSE	Always false	Yes
NGLE	9	Unordered	One of reg1 and reg2 is not-a-number	Yes
SEQ	10	reg2 = reg1	Ordered (both reg1 and reg2 are not not-a-number) and equal	Yes
NGL	11	reg2 ? = reg1	Unordered (one of reg1 and reg2 is not-a-number) or equal	Yes
LT	12	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	Yes
NGE	13	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than	Yes
LE	14	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	Yes
NGT	15	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	Yes

Table 7.15	Comparison	Conditions
------------	------------	------------

Note: ?: Unordered (invalid comparison)

When explicitly testing Q-NaN OLT, r12, r13, 0 CMPF.S # Check if r12 < r14 UN, r12, r13, 1 CMPF.S # Check if unordered TRFSR 0 ΒT L2 # If true, go to L2 TRFSR 1 ΒT ERROR # If true, go to error processing # Enter code for processing when neither unordered nor r12 < r14L2: # Enter code for processing when r12 < r14... # When using a comparison to detect Q-NaN LT, r12, r13, 0 # Check if r12 ?< r14 CMPF.S TRFSR 0 # If true, go to L2 ΒT L2 # Enter code for processing when not r12 < r14L2: # Enter code for processing when r12 < r14••• [Floating-point Invalid operation exception (V) operation exceptions] [Supplement] A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.



RH850G3MH Software

[Operation result]

[Condition code (fcond) = 0 to 7]

reg1(B) reg2(A)		-Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN							
±Normal	ci														
±0		Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)													
±∞			,	,	, ,	,									
Q-NaN				Unordered			-								
S-NaN				Unorde	ered [V]			-							

[Condition code (fcond) = 8 to 15]

reg1(B) reg2(A)		–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
±Normal	C,							
±0		res result of on mparison cor						
±∞								
Q-NaN				Upord	ered [V]		-	
S-NaN				UNDID				

Note: [] indicates an exception that must occur.



CVTF.DL																				FI	oat	ing-	poi	nt C	Соі	nver	t Do	out	ole	to l	on	g (C	out	ole)
																			С	onv	ers	ion	to	fixe	d-p	point	for	ma	at (d	dou	ıble	pre	cisi	on)
[Instruction format]		1	CVI	ΓF.C)L	reg	2, r	eg	3																									
[Operation]			reg3	←	cv	rt reg	g2 (do	ubl	e —	→]	on	g-v	vor	d)																			
[Format]		-	Forn	nat	F:1	I																												
[Opcode]	15			1	1	10					_	4				0		54				07	20	2.01	-	2			01	20			17	16
	15 r	r	r		- 1	10 1 1	. 1	L :	1	1 1	L	0	0	1	0	0	-		w	W	w		1	0		0 (1	21 0		0	1	17 0	0
			reg2																I	reg	3			С	ate	egor	/	typ	e		sub	o-op		
[Description]			the r acco by g Whe rang If in regis follo	regis ordan gener en th ge of vali- ster ows, Sou	ste nc ral ie 22 d o is ao	action r pai e with -pur source ⁶³ _ opera set a ccore ce is	r sj th t pos ce c 1 to atio s a ding a p	pec he se 1 ope on c on c in g to	cific cur regi erar 2 ⁶³ exc nva o di itiv	ed b rren ister nd is , an epti ilid iffer e nu	y tr r i i i i i i i i i i i i i i i i i	ge rou eg nfi EE ns per nce	ner ndi 3. nite E7 are atic es a	ral-j ing e or 54- no on a mo	pur mc nc de: t er and ong ∞ :	pos ode t-a fine nab nc sou 2 ⁶³	se r_{r} ; r_{r} r_{r} r_{r} r_{r}	e re and nur l in exc cce - 1	egis d s mb iva the cep s. is	er, llid e protio	res or op res n c	eg2 the whoera erv occu	to res en tio ati- urs	64- the n e on . Th	-bi : in : rc exc bit he	it fix n the counc cept: t (bi retu	ted re led on t 4 urn	l-po gis l re is) o	oin ster esul de of th	t for part tec ne	orm air s ou cted FPS	nat, spe utsi I. SR	in cifi de t	ed
[Floating-point operation exceptions]		Inva	lid o	эp	erati cept	on	ex	cep				-	on ((E)																			

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CVTF.DS										C							ouble format				
[Instruction format]		CVTF.DS	S reg2,	, reg3																	
[Operation]		$reg3 \leftarrow c$	vt reg2	(doub)	$le \rightarrow$	sing	(le)														
[Format]		Format F	:I																		
[Opcode]	45	44	10		F	4			0	04			07	00	05	00	22.24	20		47	40
	15 r r	11 rrr0 reg2	10	1 1	5	4	0 0	1		<u>31</u> ₩	w v	v w g3		1	0 C	0	22 21 1 0 type	1	0 0 ub-op		0
[Description]		This instr the registe format, an accordance	er pair nd stor	specifi es the r	ed by esult	ger in g	ener	-purj al-pı	pos 1rp	e re ose	egist	er re	eg2	to s	ingle	-pre	cision	floa	ting-j		
[Floating-point operation exceptions]]	Unimpler Invalid op Inexact ex Overflow Underflow	peratio xceptic excep	n excep on (I) tion (O	otion	-	otion	(E)													

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Operation result [exception]	A (Si	ngle)	+0	-0	+8	_8	Q-NaN	Q-NaN [V]

Note: [] indicates an exception that must occur.



CVTF.DU	L															(-											-		ong			
[Instruction format]		(CV	TF	F.DI	UL	re	g2,	re	g3																										
[Operation]		r	eg?	3 ←	— c	vt	reg	2 (0	dou	ıble	\rightarrow	· u	nsig	gn(ed	lo	ng-	w	or	d)																
[Format]		F	For	ma	at F	:I																														
[Opcode]																																				
[opeoue]	15				11	10)				5		4				0	3	31				:	27	26	25	5	2	32	22.2	21 2	20			17	16
	r	r	r	r	0	1	1	1	1	. 1	1		1 ()	1	0	0		w	W	1	w v	N	0	1	0	(0 () :	1	0	1	0	1	0	0
	L	r	eg2	2		Ì						1						Ī			re	g3			1	са	ate	gor	/	typ	e		sub-	-op		
[Description]		t f s V r d I r	he form spec Wh resu dete lf in regi	reg mat ciff nen ult ect nva iste low	gist t, in ied the is o ed. alid er is vs, a Sour	er j n ac by e so outs op s se acc	pain cco ge ourc side era et a: ord is a	r sp rda: ner ce c e th tion s ar ing a po	nce al- ope e r n e n in g to osit	ified e wi pur rand ang xce nval o dif	d by ith t pos d is e of fid c ffere	y § the se in f 2 on op en	gene e cu regi nfin 2 ⁶⁴ _ s ar erat ces	era rre iste ite - 1 re 1 tio ar	al-] ent er e, r 1 to no on a mo uts	pui t ro reg not o 0 t ei anc ong ide	rpo yun g3. -a-1 , ai nab l no so e th	se di nu n l ole o e ur	e ro ng im IE ed, ex rce rce	egi g m lbe EE th cep es. nge	e o	er r de, or 1 54- pre fon	eg an neg de sei oc	2 t d s gat fin cu	to ustor tive ed ttic rs. 1 t	ins ces e nu inv on t Th	th un va bit	ned nbe lid (bi retu	64 esu r, c ope t 4 urn	4-b lt i or v era	it f n th vhe tion f th llue	ixe ne i en t n e le l e di	t con ed-p regi the : exce FPS iffen is re	ooir iste rou epti SR rs a	nt r pa und on us	air ed is
[Floating-point operation exceptions]	Ι	nva	alio		per	atio	on e	exc	ept			cept V)	io	on ((E)																				

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CVTF.DU	W															•						ible to		-	-					
													Co	onv	ers	ion	to	JNSI	igne	ed fi	xed	-poin	t fc	orma	it (d	oul	ole p	recis	sion)
[Instruction format]		C	VTI	F.DI	JW	reg.	2, re	g3																						
[Operation]		re	g3 •	← c	vt re	g2 (dou	ble -	\rightarrow v	WOI	rd)																			
[Format]		Fc	orm	at F	:I																									
[Opcode]																														
	15				10				5	4					31				27		25		Ĩ	22 2				17	7 16	
	rı	r r	r r	0	1	1 1	1 1	1	1	1	0	1	0	0	W	W	W	W	W	1	0	0	0	1	0	1	0	0 0	0	
		reę	g2													I	reg	3			ca	tegor	у	typ	е	5	sub-o	эр		
[Description]		the for W res de If	e re rma /hen sult etect inva gist llov	gist at, an the is c ted. alid er is vs, a	oper oper s source oper s set accor	ir sjores rce de th catic as a rding	pecia s the oper he ra on ex n in g to	fied resu rand inge kcep valic diffe	by alt i is i of tion d op	gen in g infi 2 ³² ns a pera	nera gene inite $2^2 - 1$ are r ation es an	l-p ral , no to not n a	ena nd s	oos rpo i-n an abl no sou	e re ose um IE ed, exo	egi re; bei EE tho cep es.	ster gist r, o 75 ⁴ e pr otio	r reş r ne 1-de n o	g2 reg ega efir erva	to 1 3. tive ned atic	insi e nu inv on b Th	igned umbe valid oit (b e ret	d 3 er, op it 2 uri	32-b or v pera 4) o n va	it fi vhe tior f th lue	ixe en t n e: e F e di	ed-po he r xcer FPSI ffers	oint ound otior R s as	ded 1 is	l
						-										-							w	'. <i>L</i>	_	11	15 10	tuin	cu.	
[Floating-point operation exceptions]	In	nim vali	pler id oj	nent	ed c ion	opera exce	ation	n ex	ccej	ptio			un	nbe	r, c	or –	∞:	0 is	s re	tur	ned.								
		In	exa	ct e	xcep	tion	i (1)																							

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CVTF.DW	/																C													ord (C e pre		ŗ
[Instruction format]		CV	VTF	F.DV	W re	g2,	re	g3											,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,				ixee			0111		<u>(uot</u>			.0101	
[Operation]		reş	g3 +	— c	vt reg	g2 ((do	ouble	; →	W	orc	d)																				
[Format]		Fo	orma	at F	:I																											
[Opcode]	15			11	10				5	4	L				0	3	1				27	26	25		23	22	21	1 20)		17	16
	r	r r	r r	0	10	1 :	1	1 1	1	0	,) (0 1	L (0	0	w	r w	v	w	w	w	1	0	0	0	1	0	1	0	0	0	0
		reç	g2															re	ega	3			са	ateg	jory	ty	'npe	<u> </u>	su	ıb-op)	
[Description]		the sto W ran If	e reg ores hen nge inva giste llow	gist the the of 2 alid er is vs, a	uctio er pa resu sour 2 ³¹ – oper s set a accor rce is	ir s lt in ce l t ation as a din a p	n g ope on o on o an i g te	cifie ener eran -2 ³¹ , exce nval o dif itive	d by ral-p d is an ptic lid c ffero	y g pur int IE ons ope enc mb	en po fin EE an era ces	eral se 1 ite o 2754 re n tion am	$\frac{1-p}{regi}$ or r $\frac{4-d}{1-d}$ of and $\frac{1}{r}$ $\frac{1}{r}$	urp iste not efi ena nd 1 g s	nos -a- abl no ou 31	reg -nu ed lec urc	reg g3. uml inv l, tl xce xes. 1 i	gis be val he pt	ter er, id pitio	or op ceso n o	g2 wh era erv ccu	to en tio atio urs	32- the n ex on t Th	bit rou kce bit (fixe inde ptic (bit etui	ed - ed r on i 4) m v	poi rest is d of	nt f ult i lete the	forn is o cte FP	mat, outsi ed. PSR	, an de t	d
[Floating-point operation exceptions]	In	vali	d oj	nente perat xcept	ion	ex	cept			-	tion	ı (E	5)																		

[Operation result]

reg2 (A)	+Normal	-Normal	+0	-0	+∞	- ∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CVTF.HS																Flo	atir	ng-p	oint	Cor	١ve	rt H	alf	to S	ingle	e (Sin	gle)
														Cor	nver	sio	n to	flo	ating	g-po	int	forn	nat	(sin	gle p	orecis	ion)
[Instruction format]		CVT	F.HS	S reg2	2, reg3	3																					
[Operation]		reg3	← c	vt reg	2 (hal	f→	sin	gle)																		
[Format]		Form	at F	:I																							
[Opcode]	45			10			_					0	24				07	00	05		00	20	04	20		47	10
	15 r :	rrı		10 1 1	. 1 1	L 1	5 1	4	0	0	1		31 w	w	w	w	27 w	1	25 0		0	22 1			0		0 0
		reg2												I	rega	3			ca	tego	ry	typ	ре		sub-	ор	
[Description]		the lo	wer ling	16 bi the re	n arith its of g esult ir e regis	genei 1 acc	ral- corc	pur lanc	pos	e re	egis	tei	r re	g2	to s	sing	gle-	pre	cisi	ion	flo	oatir	ng-j	poi	nt fo	rma	
[Floating-point operation exceptions]	Inval	id oj	peratio	on exc	cepti	on	(V)																			
[Supplement]		be ac	cura	tely c	tion of onvert hed ev	ted in	nto	sin	gle-	-pre	ecisi	io	n fl	oat	ing	-po	int	for	ma	-	-						

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	A (ŀ	Half)	+0	-0	+∞	_∞	Q-NaN	Q-NaN [V]

Note: [] indicates an exception that must occur.

CVTF.LD														Fl	oati	ng-	poir	nt Co	onve	ert L	-oné	g to	Do	uble	(Doi	uble)
												(Con	vers	sion	to	floa	ting	-poi	nt fe	orm	at (dou	ble p	oreci	sion)
[Instruction format]		CVTF.LD	reg	2, reg	3																						
[Operation]		$reg3 \leftarrow cv$	vt reg	;2 (loı	ng-w	ord	\rightarrow	doı	uble	e)																	
[Format]		Format F:	I																								
[Opcode]	15	11	10			5	4				0	31				27	26	25		23	22	21	20		1	7 1(3
	r r	r r 0	1 1	. 1	1 1	1	0	0	0	0	1	w	w	W	W	0	1		0	0	1			0	0 1	C]
		reg2												rega	3			ca	tego	ory	typ	ре		sub-	ор	ĺ	
[Description]	-	This instrupair speci accordance by genera	fied l e wi	by ger th the	neral- curre	-pu ent	rpo: rou	se r ndi	egi	ster	re	g2	to	dou	ble	e-pr	reci	sio	n fl	oat	ing	-po	oint	form	nat i	n	
[Floating-point operation exceptions		Inexact ex	cept	ion (I)																						
[Operation result]								_							_					_							
		reg2 (A)		+Inte	ger					tege	ər						iteg	er)									
		Operation result [exception					A	(NO	orma	1)					+	-0											



													F	loa	tinę	g-po	oint (Conv	ert	Lon	ıg to) Sir	ngle	(Sin	gle)
												Co	nver	sior	n to	flo	ating	g-poir	nt fo	orma	<u>at (s</u>	sing	le pr	ecis	ion)
[Instruction format]		CVTF.LS	reg2	2, reg3																					
[Operation]		$reg3 \leftarrow cr$	vt reg	2 (long	g-woi	:d -	→ siı	ngle	e)																
[Format]		Format F:	I																						
[Opcode]																									
	15 r 1	<u>11</u>	10	1 1			4 0 0	0	0		31 w		W	747		26 1	25 0			22 2 1 (21 2		0 0		16 0
		reg2		± ±	± .			0	0	-			reg3					egor					ub-o		
[Description]		This instr pair speci stores the current ro	fied b resul	y gene t in gei	eral-p neral-	urp	ose	reg	iste	r re	eg2	to	sing	gle-	pre	ecis	ion	floa	tin	g-p	oin	t fo	orma	it, a	nd
[Floating-point operation exceptions	5]	Inexact ex	ccepti	on (I)																					
[Operation result]								1						-						_					
		reg2 (A)		+Integ	jer			-1	nteg	ger				0	(ir	iteg	er)								
		Operation result [exception					A (N	orm	al)					+	0										



CVTF.SD															Εŀ	oatır	ıg-p	oint	Cor	nve	rt S	Ingl	e to	Do	uble	e (Do	bub	e)
														Co	nve	rsio	n to	floa	ting	-poi	int f	orm	at (dou	ble	prec	isic	n)
[Instruction format]		CVTF	.SE) reg.	2, re	eg3																						
[Operation]		reg3 ←	- c	vt reg	2 (s	ingl	le →	∙ do	ub	le)																		
[Format]		Forma	t F	:I																								
[Opcode]	15		11	10				5 4	4			0	3	1			27	26	25		23	22	21	20			17	16
	r	rrr	r	1 1	1	1	1	1 (0	0 (0 1	. 0	W	7 V	νw	v w	0	1	0	0	0	1	0	1	0	0	1	0
	L	reg2													reę	g3			cat	tego	ory	ty	pe		sub	-op		
[Description]		This in genera the cur purpos	ıl-p rrer	urpos nt rou	e re ndir	gist 1g m	er re	eg2	to	dou	ıble	-pre	ecis	ior	n flo	oati	ng-p	poir	nt fo	orm	nat,	in	acc	ord	lanc	ce w	rith	
[Floating-point operation exceptions]	Unimp Invalio Inexac	d op	perati	on e	exce			-	tior	n (E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Do	ouble)	+0	-0	+∞	-8	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.

CVTF.SL												Floa	atinę	g-po	oint	Conve	rt Sir	ngle	to	Long	(Siną	gle)
CVIF.SL											С	conver	sion	to	fixe	d-point	forn	nat (sin	gle pi	recisi	ion)
[Instruction format]		CVTF.SL	reg2	, reg3																		
[Operation]		reg3 ← c	vt reg2	2 (singl	$e \rightarrow l$	ong	-wor	d)														
[Format]		Format F	:I																			
[Opcode]																						
	15	11 r r r	10	1 1	1 1	4	0 1	0		31		ww		26		23 0 0	22 1			0 1		16
	r r	reg2		I I	I I	0	0 1	0	0	w		eg3	0			tegory	1			sub-o		
[Description]			urpose mode, source $2^{63} - 1$ operate s set as according rce is a	e registo and sto e opera to -2^6 tion exo an inv	er reg ores tl nd is : ³ , an l ceptio alid o liffere ve nur	2 to he r infini IEE ns a pera nce	o 64-b esult nite o E754 are no ation es am- er or -	bit fi in the or not-defined on the the the on the	xed he i t-a- fine nabl no sou 2 ⁶³	-nur ed in led, exco urce - 1	int ster mbo iva the cep s. is	form r pair er, or lid op e prese tion o	at, i spe whe era erv occu ned.	in a cifi en t tior atio urs.	he he n ez on b Th	ordanc by gen roundo cceptic oit (bit e retur	e wi nera ed re on is 4) c rn va	ith 1 l-pu esul s de of th	the arp It is tec ne]	curr ose r s outs ted. FPSF	ent egis side 1	ter
[Floating-point operation exceptions]	Unimpler Invalid oj Inexact ez	peratio	n exce				(E)														

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

CVTF.SH																•				ert Sinç	-			
[Instruction format]		CVTF.S	SH	reg2,	, reg3				Co	nve	rsio	n to	hali	f-pre	cisic	n fl	batir	ig-po	<u>oint</u>	forma	<u>: (sin</u>	<u>gle</u> p	precis	sion)
[Operation]		reg3 ←	zer	o-ext	tend (c	evt re	•g2	2 (si	ngle	\rightarrow	hal	f))												
[Format]		Format	F:I																					
[Opcode]																								
	15 r 1	r r r	11 1 r :		1 1		1	4	0 0	1		31 w	W	W	2 w v		0	0	23 0	22 21 1 0		0		7 16 0
		reg2											r	eg3			Ca	ateg	ory	type		sub-	ор	
[Description]		This ins general- accorda stored in	-pur nce	pose with	regist the cu	er re urrer	eg2 nt r	2 to 1 roun	half- ding	pre g mo	cisi	on	floa	ting	g-po	int	fori	nat,	ro	undin	g th	e res	sult i	n
[Floating-point operation exceptions]	Unimpl Invalid Inexact Overflo Underfl	ope exc w e	ration eptio xcept	n exce on (I) tion ((ptio D)		-	tion	(E)														

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (ŀ	lalf)	+0	-0	+∞	_8	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



CVTF.SU	L														Flc	oatir	ng-	poir	t Co	nve	ert S	Sing	le to	Un	sig	ned	-Lo	ng (Sing	gle)
														Cor	nve	rsic	on	to ui	nsigr	ned	fixe	ed-p	oint	form	mat	: (sir	ngle	e pre	cisi	on)
[Instruction format]		CV	VTF.	.SU	Lre	eg2,	reg	3																						
[Operation]		reg	<u>3</u> 3 ←	- cv	vt reg	g2 (sing	le –	→ u	nsi	igne	ed	lon	g-w	vor	d)														
[Format]		Fo	orma	ıt F:	I																									
[Opcode]	4-				10				_													_								4.0
	15 r	r r	- r	11 r	10	1 1	. 1	1	5	4	0	1	0		31 w		, ,	wv		26			23		21 0	20		1		16 0
		reg		-	± .			-	-		0	-	0	0				eg3					gory		pe			⊥ b-op		
[Description]		ger cur reg Wl res det If i	nera rren giste hen sult i tecte inva giste llow	t ro er re the is o ed. hlid er is vs, a	sour utsic oper set a ccor ce is	se ro ng r rce c le th ratio as an ding a p	egist mod oper- ne ra on ex n inv g to ositi	ter r e, ar and nge ccep valic diffe	eg2 nd s is i of tion d op eren	2 to sto inf 2 ⁶⁴ ns per nce	o ur res init ⁴ _ are catic es a er o	nsig the e, 1 1 t no on a mo	gne e re not- o 0 t er anc ong	d 6 sul a-r , an nab nc sou the	4-t t ir nur n II lec o ez urc e ra	n th n be EEH l, th xce ang	fix er, E7 ne pt	or 1 54-0 presion	poir ster nega defin serv occu	nt fo pai ttiv ned atio urs.	orm ir s e n in in on l Th Th	um val bit), o	in a cifie ber, id o (bit retur	or per 4)	ord y g wh atio	lanc gene nen on c the ue d	ce veral the exc FP liff	with l-pui e rou cepti SR čers a	the rpo and on	e se ed is
		•	S	our	ce is	a n	egat	ive	nur	nb	er,	not	t-a-	nur	nb	er,	or	. –∞	: 0 i	s re	etui	me	d.							
[Floating-point operation exceptions	;]	Inv	valic	d op	nente perati ccept	ion	exce				-	on	(E)																	
[Operation result]																														

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CVTF.SU	W	I															Flo	ati	ing	-po	oint	Co	nve	rt Si	ngle	e to	Un	sigi	ned	-W	ord	(Sin	igle)
																Co	nve	rsi	ion	to	o un	sigr	ned	fixe	d-po	oint	for	mat	: (si	ngl	le pr	ecis	sion)
[Instruction format]		C	CVT	ΓF.S	SU	W	reg	g2,	re	g3																							
[Operation]		r	eg3	, ←	- c'	vt re	g2	(s	ing	gle -		uns	igı	ned	wo	rd)																	
[Format]		F	Form	nat	: F:	Ι																											
[Opcode]						4.0					_											07		~-		~~~							
	15 r	r	r	r	11 r	10	1	1	1	1	1	4	C) 1	0		31 w		w	w	W	27 w		25	0			21 <u>21</u> 0	20		0 0		7 16 0
			eg2		-	-	-	-	-	-	-									eg					iteg		<u> </u>	/pe	<u> </u>		ub-o		
[Description]		g g V r d I I r	gene gene Whe esul lete f in regis	eral eral en t lt is cteo val ster	l-p he s o d. id s, a	uctio urpo sou utsio oper set ccon	ose ose rce de rat as rdi	reg reg e of the ion an	gis gis per e ra n ez in to	ter ter rand ang xce val dif	reg reg d is e of ptic id c	2 to 3. inf $f 2^3$ ons open	in: 2_ arv rat	insig ite, - 1 t e no ion amo	gne not to 0 ot en anc ong	d 3 -a-1 , ar nab l nc sor	2-t nur n IH led o ey urc	nb EE l, t	the ep	xe ;, 0 75, e p tic	d-r or n 4-d res	ega lefii erv	nt fo ned atic	orm e nu inv on t Th	at, a umb valie oit (ber, d o	d sto , or oper 4) rn v	ore wł rati of valu	es then on the	ne th ex FI diff	resu ne ro cep PSR fers	ult i ound tion	n ded 1 is
			•	So	our	ce is	s a	po	sit	ive	nu	mbe	er	outs	side	th	e ra	an	ge	0	f 2 ⁶	o4 _	11	0 0	, or	+c	o: 2	,32	- 1	is	ret	urn	ed.
			•	So	our	ce is	s a	ne	ga	tive	e nu	ımb	oer	, no	t-a-	nui	nb	er	; o	r –	-∞:	0 i	s re	etur	ned								
[Floating-point operation exceptions]	I	nva	alid	oŗ	nent berat kcep	io	n e	xc				-	ion	(E)																		

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

CVTF.SW	/														0					Conve							
															Co	onve	rsio	n to	fixe	d-poin	tor	mat	(SII	igle	prec	SISIC	on)
[Instruction format]		CV	TF.SV	V reg	2, re	eg3																					
[Operation]		reg3	3 ← c	vt reg	2 (si	ingle	\rightarrow i	woi	rd)																		
[Format]		For	mat F	Ι																							
[Opcode]				10			_																				
	15		11	10			5	4				0	3	1			21	26	25	23	3 22	21	20			17	16
	r	r r	11 r r	1 1	1	1 1	1	0	0	1	0	0	Ŵ	V V	J	W V	v w	1	0	0 0	1	0	0	0	0	0	0
		reg2	2												re	eg3		1	са	tegory	ty	/pe		sub	-op		
[Description]		geno purp Who rang If in regi follo	s instr eral-p pose r en the ge of 2 ivalid ster is ows, a Sour Sour	urpos egiste sourc 2^{31} – opera set a ccord ce is a	e reg r reg e op l to tion s an ing a po	gister g3. -2^{31} , exce inval to dif	reg d is an I ptio id o fere nui	2 to inf EE ons oper enc mb	o 32 init EE7 are rationes a er c	2-bi e or 54-o not on a amo	t find no defit en and ong ∞ :	t-a- ine- abl no sou 2 ³¹	d-p -n leo e uro -	um inv d, tl xce ces.	nt be ali he pt	forn r, or d op pres ion	nat, wh bera serv occ	en tior ation	l sto the n ex on b . Th	round ceptio	e r ed : on i : 4) rn v	esul resu s de of t	lt in ult i etec the	s ou s ou ted. FPS	nera itsid SR	ıl- le ti	
[Floating-point operation exceptions]	Inva	mpler alid op cact ez	peratio	on e	xcept			-	on ((E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-8	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

		Operation					Α	(Nor	mal)					+0									
[Operation result]		reg2 (A)		+Inte	ger				-Inte	ger				0 (i	nteg	jer)							
[Floating-point operation exception	s]	Inexact ex	kcept	ion (I))																		
[Description]		This instr register p format in specified	air sp acco	ecifie rdance	d by e wit	ge h tł	nera ne c	al-pu urre	irpos nt ro	e re und	egis	ster	reg2	to c	lou	ble-j	precis	sion	floa	atin	g-po	int	
		reg2										I	eg3			cat	egory	ty	ре	s	ub-o	D	
	r :	rrr O	1 1	1 1	1 1	1	1	0	0 0	1	w	W	w v	w 0	1	0	0 0	1	0	1	0 0	1	0
[Opcode]	15	11	10			5	4			0	31			27	7 26	25	23	3 22	21	20		17	16
[Format]		Format F	I																				
[Operation]		reg3 ← c	vt reg	g2 (un	signe	ed l	ong	g-wo	ord –	→ do	oub	le)											
[Instruction format]		CVTF.UI	D re	eg2, re	eg3																		
												Con	versio	on to	floa	ting	point	form	at (d	doub	ole pr	ecisio	on)
CVTF.UL	.D									F	loat	ing-	point	Con	vert	Uns	igned	-Lon	g to	Dou	ıble (I	Doub	le)

R01US0143EJ0130	Rev.1.30
Dec 22, 2016	

result [exception]



CVTF.UL	S									Floa	ating	g-poi	nt C	on	ver	t Ur	isign	ed-l	Lon	ig to :	Sing	le (S	Sing	e)
											Cor	ivers	ion	to	floa	ating	I-poir	nt fo	orma	at (si	ngle	pre	cisic	n)
[Instruction format]		CVTF.U	LS re	g2, reg3	3																			
[Operation]		$reg3 \leftarrow c$	vt reg	g2 (unsig	gned	long	-wor	d→	• siı	ıgle	e)													
[Format]		Format F	:I																					
[Opcode]	45		40		_				0	0.4					~~	05	0			1.00			47	40
	15 r 1		10	L 1 1	1 1	4	0 0	0		31 w	W	w			26 1	25 0	0 (1 (2 <u>1 20</u> 0 0) 0		17 1	0
		reg2										eg3				cat	egor	y 1	type	ə	sub	o-op		
[Description]		This instr register p format, a accordan	air sp nd sto	becified bres the	by ge result	nera	ıl-puı gener	rpose al-p	e re urp	egis ose	ter	reg2	to	siı	ngl	le-p	recis	sior	n fl	oatir	ıg-p	oin		10
[Floating-point operation exceptions	5]	Inexact e	xcept	ion (I)																				
[Operation result]		reg2 (A)		+Intege	ər			Integ	ner				0 ('inf	teg	er)								
		Operation	1	lincoge		A	(Norm		,				+0			.,								

result [exception]



CVTF.UV	٧D								Flo	oatir	ng-p	oint C	onv	ert	Unsi	gned-	Word	to Do	ouble	(Doub	le)
										C	Conv	versio	n to	floa	ting-	point	forma	<u>t (do</u>	uble p	recisio	on)
[Instruction format]		CVTF.U	WD r	eg2, reg	3																
[Operation]		$reg3 \leftarrow c$	evt reg	2 (unsig	ned v	wor	$d \rightarrow $	dout	ole)												
[Format]		Format F	7:I																		
[Opcode]	45				_				•	0.4			07		05	0.0				47	40
	15 r :	r r r r	1 10 1 1	. 1 1	5 1 1		0 0	0 (31 w	W	w w		26 1	0		1		0	17 0 1	16 0
		reg2									r	eg3			cat	egory	type	ə	sub-o	op	
[Description]		This inst general-p the curre purpose p This con	ourpos nt rou registe	e registe nding m er reg3.	er reg ode,	2 to and	dou store	ble-p es the	orec e re	cisio esul	on f t in	loating the r	ng-p egis	poir ster	nt fo pair	ormat, r spec	, in a cified	ccoro l by g	dance gener	e with	1
[Floating-point operation exceptions	5]	None																			
[Operation result]		reg2 (A)		+Intege	r		-	-Inte	ger				0 (iı	nteg	jer)						

Operation

result [exception]



A (Normal)

+0

CVTF.UW	/S											F	=loa	ting	-poi	int (Cor	ver	t Ur	nsig	nec	1-W	ord	to S	Sing	gle (Sing	gle)
														Con	ver	sior	ı to	floa	ating	g-pc	oint	forr	mat	(sir	ıgle	e pre	cisi	on)
[Instruction format]		CVTF.	UV	VS re	eg2, r	eg3																						
[Operation]		reg3 ←	- cv	vt reg	2 (un	sign	ed v	vor	d →	→ si	ngle	e)																
[Format]		Format	F:	I																								
[Opcode]	45			10			-	4				0	24				07	00	05		00	00	04	20			47	40
	15 r 1	rrr		10 1 1	1	1 1		4 1	0	0	0		31 w	w	w			1	25 0		0	1	0			0	17 1	0
		reg2												r	eg3	;			ca	tego	ory	ty	ре	<u> </u> 	suł	b-op)	
[Description]		This in general general mode.	-pı	urpos	e regi	ister	reg2	$\frac{1}{2}$ to	sin	gle	-pre	eci	sio	n fl	oati	ing	-pc	oint	for	ma	it, a	and	sto	res	the	e res	sult	
[Floating-point operation exceptions]	Inexact	ex	cepti	ion (I))																						
[Operation result]								_					_	_	_					_			1					
		reg2 (A Operat result			+Inte	eger		A	(No		n teg al)	er					(in 0	nteg	jer)									

[exception]



CVTF.WE)										С	on												(Dou orecis	ŕ
[Instruction format]		CVTF.W	D reg	g2, reg3														<u> </u>			<u></u>		<u></u>		<u></u> ,
[Operation]		reg3 ← c	vt reg	2 (word	\rightarrow d	out	ole)																		
[Format]		Format F	I																						
[Opcode]																									
	15 r	11 r r r r	10 1 1	1 1	5 1 1	T	0	0	0 0	1	31 w	w	w		27 0	26 1	25 0		23 0	22 1		20 1	0	17 0 1	/ 16 0
		reg2											eg3				ca	tego	ory			5	sub-o		
[Description]		This instr purpose r current ro register re This conv	egiste undir 23.	r reg2 to 1g mode	o dou , and	ble sto	-pre res	the t	on f resu	lo lt	atiı in	1g- the	poii reg	nt f iste	ori er j	mat pair	t, in r sp	ac eci	cor fie	dan d by	nce y ge	wit ener	h th	e	ose
[Floating-point operation exceptions	5]	None																							
[Operation result]								_					_							_					

reç	g2 (A)	+Integer	-Integer	0 (integer)
res	peration sult (ception]	A (No	rmal)	+0



$Conversion to floating-point format (single precision)$ [Instruction format] CVTF.WS reg2, reg3 [Operation] reg3 \leftarrow cvt reg2 (word \rightarrow single) [Format] Format F:I [Opcode] $\frac{15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16}{ $	CVTF.WS	S												Fl	oatir	ng-p	oint	Co	nver	rt Wo	ord to) Sii	ngle	(sing	gle)
$[Operation] \qquad reg3 \leftarrow cvt reg2 (word \rightarrow single)$ $[Format] \qquad Format F:I$ $[Opcode] \qquad 15 \qquad 11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31 \qquad 27 \ 26 \ 25 \qquad 23 \ 22 \ 21 \ 20 \qquad 17 \ 16 \qquad 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$													Cor	iversi	on t	o flo	batin	g-p	oint	form	iat (s	sing	le pre	ecisio	on)
[Format] Format F:I [Opcode] 15 11	[Instruction format]		CVTF.W	'S reg	g2, reg.	3																			
[Opcode] 15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16 x x x x x 1 1 1 1 1 1 1 0 0 0 0 0 w w w w w 1 0 0 0 1 0 0 0 0	[Operation]		$reg3 \leftarrow c$	evt reg	g2 (wo	$rd \rightarrow$	→ si	ngle	e)																
$\begin{bmatrix} 15 & 11 & 10 & 5 & 4 & 0 & 31 & 27 & 26 & 25 & 23 & 22 & 21 & 20 & 17 & 16 \\ \hline x & x & x & x & 1 & 1 & 1 & 1 & 1 & 1 &$	[Format]		Format I	F:I																					
r r	[Opcode]																								
[Description] This instruction arithmetically converts the 32-bit fixed-point format contents of general-purpose register reg2 to single-precision floating-point format, and stores the result in general-purpose register reg3. The result is rounded in accordance with the current rounding mode. [Floating-point operation exceptions] Inexact exception (I) [Operation result] reg2 (A) +Integer -Integer 0 (integer)				1	1 1 1	1	1	4	0	0 0		Ĩ	1.7	T.7 T									0 0		
purpose register reg2 to single-precision floating-point format, and stores the result in general-purpose register reg3. The result is rounded in accordance with the current rounding mode. [Floating-point operation exceptions] [Operation result] reg2 (A) +Integer 0 (integer)				 		. 1	T		0	0 0	0	w			v w		<u> </u>								0
operation exceptions] [Operation result] reg2 (A) +Integer 0 (integer)	[Description]		purpose general-j	regist	er reg2	to si	ing	le-p	oreci	ision	flo	atir	ng-p	ooint	for	ma	t, an	nd s	tore	es th	e re	sult	t in		ıg
reg2 (A) +Integer –Integer 0 (integer)		5]	Inexact e	except	ion (I)																				
	[Operation result]				•																				
					+Integ	ger		^			ger					inte	ger)								

reg2 (A)	+Integer	–Integer	0 (integer)
Operation result [exception]	A (No	rmal)	+0



																Float	ng-poi	nt Di	vide	(Dou	ıble)
DIVF.D													Fl	oati	ing-p	oint d	vision	(dou	ıble p	recis	sion)
[Instruction format]		DIVF.D 1	eg1,	reg2,	reg3																
[Operation]		reg3 ← re	eg2 ÷	regl																	
[Format]		Format F:	I																		
[Opcode]	45	44	10			~			0		24		07	00	05	00	00.04	- 20		4-	7 40
	15 r	rrr 0	10 1 1	L 1 1		T	4 R R	RR	0 0		w w	w w	0	1	25 0	0 0	22 21 1 1		1 3		7 16 0
		reg2						reg1		Ì	l	reg3			cat	egory	type		sub-c	р	
[Description]			by ge of the er pai ifinite e ope no re n the	eneral- regist ir spec e accu eration emaine	-purpo er pai ified l racy, a resul der. Th ct ope	ose r sj by and t m he	regi pecif gene l the nay b effec	ster reg ied by ral-pu result e judg	g2 b ger rpos is ro ed a as f	oy ne se ou as o is	the oral-p ral-p regis undec an ir llows enat	double urpose ster re d in ac nexact s: bled (F	e-pr e reg g3. cor ope	ecia gist Th dar erat	sion ter r le op nce tion XE.I	float eg1, a peration with t even (1 = 1),	ing-po and sto on is e he cur when the in	oint ores exec trent the	form the r uted t rou oper	nat resul as i ndin ratio	lt in f it ng n
		prese	n the ervati	ion bit	is set	(F	PSR	excepti .XP.I = vhen th	= 1)	, i	and tl	he quo	otier	nt n	nay	have			-		
[Floating-point operation exceptions]	Unimplen Invalid op Inexact ex Division b Overflow Underflow	erati cepti by zei exce	on exc ion (I) ro exc eption	ceptio eption (O)	n (V)	ion (E))												



[Operation result]

reg2(B) reg1(A)	Normal	-Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal		B =	÷ A		+∞	_∞		
-Normal					_∞	+∞		
+0		[7]		N D/7	+∞	_∞		
-0	±∞	[Z]	Q-Na	ווא [V]	_∞	+∞		
+∞	+0	-0	+0	-0		NI [\/]		
_∞	-0	+0	-0	+0	Q-INC	aN [V]		
Q-NaN		•	•	•	•		Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.


DIVF.S					Floa	ting-poir	nt Divide (Sing	gle)
				F	loating-point o	livision (single precisi	ion)
[Instruction format]	DIVF.S reg1, reg2, r	eg3						
[Operation]	reg3 ← reg2 ÷ reg1							
[Format]	Format F:I							
[Opcode]								
	<u>15 11 10</u> r r r r r r 1 1 1 1	5 1 1			26 25 23 1 0 0 0	22 21 2 1 1	<u>20 17</u> 0111	0 16
	reg2		reg1	reg3	category	type	sub-op	
[Description]	This instruction divid register reg2 by the si reg1, and stores the r were of infinite accur mode. The operation result has no remaind • When the inexad	ngle-pr esult in cacy, an result r ler. The	recision floating a general-purpose ad the result is re- may be judged a e effects are as for	-point format co e register reg3. ' ounded in accord s an inexact ope bllows:	ntents of ge The operation dance with t eration even	neral-po on is ex he curro when t	urpose regis ecuted as if ent rounding he operation	ster it g n
	cause bit is set (occurs.	-	-				-	
	• When the inexact preservation bit in the last place	is set (l	FPSR.XP.I = 1),	and the quotien	t may have		-	
[Floating-point operation exceptions]	Unimplemented oper Invalid operation exc Inexact exception (I) Division by zero exce Overflow exception (Underflow exception	eption eption ((O)	(V)					



[Operation result]

reg2(B)								
reg1(A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal		В -	÷Α		+∞	-∞		
-Normal					_∞	+∞		
+0	±∞	[Z]	Q-Na	N [V]	+∞	_∞		
-0					-∞	+∞		
+∞	+0	-0	+0	-0		aN [V]		
-∞	-0	+0	-0	+0	Q-142	מושנטן		
Q-NaN					•		Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.



FLOORF.	D	L						F	loati	ng-	poir	nt C	Con	ve	rt C	Dout	le to	o Lo	ng,	round	toward	neg	ative	(Dou	uble	;)
														(Coi	nver	sion	to f	ixed	l-point	format	(do	uble j	precis	sior	1)
[Instruction format]		FLO	ORF	DL 1	eg2, 1	reg3																				
[Operation]		reg3	← fl	loor re	eg2 (d	loubl	e –	→ lo	ong-	wo	ord)															
[Format]		Form	nat F	:I																						
[Opcode]	45			10			-	4				0	24				0-		. 05	00		4 00		4	7 4	~
	15 r	rri		10 1 1	1 1	1 1	5 1	4	0	0	1	1	31 w	W	<i>,</i> ,	w v		1	0		22 2 1 0		0		7 1 0 C	
		reg2													re	g3			са	itegory	type		sub-	юр		
[Description]		the restore. The r When range If inv regis follo	egisto s the cesul n the e of 2 valid ter is ws, a Sour	er pain result is ro source $2^{63} - \frac{1}{2}$ opera s set a accord rce is a	spect in the undected open to -2 tion ession in ing to a posi	ified le reg l in the rand 2 ⁶³ , a except nvalio diff	by gist he is is n btio d o ere nur	ge er p -∞ infi IEE ns per nce	nera oair dire nite EE75 are 1 ratio es ar er or	al- p spection or 54-c not n a moteory $\cdot +c$	purprecified on, not defined the first one of the first	oos ied reg t-a- ine abl no sou 263	e r l by gar -nu d i ed ed ex urco	reg y g dlo im nv , tl ace es. 1 i	gist gen ess be vali he epti	er r era of r, or d o pres ion	eg2 -pu the wh pera serv occ	to rpc cur en atio ratio urs	64-1 se 1 rent the n ez on t Th	ing-po bit fix registe t round cception bit (bit re retu	ed-po r reg3 ding r ed res on is c 4) of rn val	int f 3. nod ult i dete	forma e. is out cted. FPS	at, a tside SR	nd	
[Floating-point operation exceptions]	Inval	id op	nente peratio xcepti	on exc	ceptio			-	n (E)															

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

FLOORF	DUL	Floating	g-poi	int Co				to Unsi on to un	-		-				-		
[Instruction format]	FLOORF.DUL reg	2, reg3															
[Operation]	$reg3 \leftarrow floor reg2 (e$	double -	→ uı	nsign	ed l	ong	-woi	rd)									
[Format]	Format F:I																
[Opcode]																	
	15 11 10	5	4				31				25			21 2			71
	r r r r 0 1 1 1 reg2		T	0 0	1	T	WV	v w v	v O	1		0 0 tegory	<u> </u>		0 sub		(
[Description]	This instruction arith the register pair spec format, and stores th The result is rounde When the source op result is outside the detected. If invalid operation register is set as an follows, according t • Source is a pos • Source is a neg	cified by ne result d in the erand is range of exception invalid of o different itive num	y gen in t $-\infty$ infi infi 2 ⁶⁴ ons a opera-	neral- he re direc inite, $4^4 - 1^{-1}$ are no ation es amor r out	-pui gist tior not to 0 ot en anc ong side	rpos er p , re -a-n , an nabl 1 no ; sou e the	e reg air sj gardl umb IEE ed, t exce irces	pecific less of er, or 1 E754-0 he pre- eption ge of 2	eg2 ed by the nega defin serva	to u ge cur tive ned atic urs. 1 t	unsi ener rent e nu inv on b The	gned al-pun t roun mber, alid o it (bit e retur	64-t pos ding or pera 4) c	bit fix e reg g mod wher ation of the alue	ked-p ister de. the exce FPS diffe	round round ptior SR rs as	deo n is
[Floating-point operation exceptions	Unimplemented ope] Invalid operation ex Inexact exception (I	ception			(E)	I											

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

FLOORF.	DUW	Floating-point Co			Unsigned						·		,
[Instruction format]	FLOORF.DUW reg	g2, reg3											
[Operation]	$reg3 \leftarrow floor reg2 ($	double \rightarrow unsign	ned wor	d)									
[Format]	Format F:I												
[Opcode]													
	15 11 10 r r r r 0 1 1 1	5 4		31		26			22 21			17	
	rrrr0111 reg2			1	w w w	1	0 0 categ	0	1 0 type		0 0 sub-op	0	0
[Description]	This instruction arith the register pair spec format, and stores th The result is rounde When the source op result is outside the detected. If invalid operation register is set as an follows, according t • Source is a pos • Source is a neg	cified by general ne result in general d in the $-\infty$ direct erand is infinite, range of $2^{32} - 1$ exceptions are n invalid operation o differences are	l-purpos ral-purp tion, re not-a-r to 0, ar ot enab a and no hong sou	se regis ose reg gardles number i IEEE led, the excep arces. e range	ster reg2 gister reg ss of the o r, or nega 754-defin e preserva- otion occu	to u 3. curr tive ned atio urs. 1 to	rent ro e numb invali on bit (The r	ed $\frac{1}{2}$ ound ber, $\frac{1}{2}$ id of (bit etur r + α	32-bit ling m or wh peration 4) of t n valu	fixe ode en on e he	ed-poir e. the rou exception FPSR iffers a	nde on i s	ed is
[Floating-point operation exceptions	Unimplemented ope] Invalid operation ex Inexact exception (I	ception (V)	n (E)										

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

FLOORF.	D	W						F	loat	ing-	poii	nt C	on	ver	t D	oub	le to	Wo	ord, I	round	tov	ward i	neg	ative	(Do	uble)
														C	Cor	ver	sion	to f	ixed	-point	for	rmat (dou	ıble p	rec	ision)
[Instruction format]		FLO	OORF	5.DW	reg2	l, reg3	3																				
[Operation]		reg?	3 ← f	loor re	eg2 (doubl	le –	→ v	vorc	ł)																	
[Format]		For	mat F	:I																							
[Opcode]	45			10			-					0	0.4				07		05			0.04	00			7 40	
	15 r	r r	11 r r	10	1	1 1	1	4	0	0	1	1	31	747	T	7 74	21	20	25			<u>2 21</u> 1 0	1	0 (0		1
					1	<u> </u>	1		0	0	Ŧ	1	vv														ļ
		reg2	2												re	g3			ca	tegory	' t	type		sub-c	эр		
[Description]		the stor The Wh rang If ir regi foll	s instr regist res the e resul en the ge of 2 nvalid ister is ows, a Sour	er pai e resul t is ro e sourc 2^{31} – opera s set a accorc rce is	r spe t in g unde ce op 1 to - ation s an ling 1 a pos	ecified generated in the erand -2^{31} , where exceptions invaliantly the exception of th	l by al-p he l is an l otio id o fere nur	y ge purp $-\infty$ infi IEF ons open enco mbo	ener oose dire inite EE7: are ratic es a er o	al- _I e reș e cti e or 54- not on a mo r +c	purj gist ion, no def t en and ng ∞ : 2	er i regimenter t-a- ine abl no sou 2 ³¹	e r reg gai d i led ex irc	regi g3. rdle imt inva , th acep es. 1 is	iste ess per alie pti	er re of , or d op pres on o	eg2 the who bera eerv boccu	to 3 cur en t tion atic urs.	32-t rent he i n ex on b The	bit fix t round cound ccepti it (bi e retu	ed- ndin led on t 4) urn	ng m resu is de) of t	nt fo lode lt is etec he l	orma e. s outs cted. FPSI	ıt, a side R	nd e the	
		•	Sou	rce is	a neg	gative	nu	mb	er, 1	not	-a-r	nun	nb	er, e	or		: -2 ⁻	³¹ i	s re	turne	d.						
[Floating-point operation exceptions]	Inva	implei alid o xact e	perati	on ex	ccepti			-	on ((E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	8	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.



FLOORF.	S	L								Flo	oatin	ng-po	oint	С	onv	ert	t Do	oub	le t	o Lo	ong,	round	to	ware	d ne	gat	ive (Sinę	gle)
																С	on	vers	sior	to	fixe	d-poin	t fo	rma	ıt (si	ngle	e pre	ecisi	on)
[Instruction format]		FL/	OORI	F.SL	, re	eg2,	reg	3																					
[Operation]		reg	3 ← f	100	r re	g2 (sing	le –	→ lo	ong	g-w(ord)																	
[Format]		Foi	rmat F	F:I																									
[Opcode]																													
	15 r	r r	11	1 10	1	1	1 -	5 1 1	4	0	0	1	0	3	1 v v	A7	747	TA 7	27 0	26 1	25	23	3 22	<u>2 2</u>	1 20) 0	1	17 0	16 0
			11 r r 2	-	-	-	± .				. 0	-	-			r	-03		0			tegon	- -			<u> </u>	+ h-on	Ŭ	Ű
[Description]		gen pai The Wh ran If i reg foll	is inst neral-p r spec e resu nen the ge of nvalid ister i lows, Sou Sou	burp iffied lt is e soo 2 ⁶³ l op s se acco rce	rose rou	e reg y ge unde e op to - tion an ing i pos	gister enerated in berand -2^{63} exco invato di sitivo	r reg ll-pu the d is , an eptic lid o ffer e nu	s_{2}^{2} t r_{1} r_{2}^{2} r_{2}	o 6 ose o di îni EE are rat es	64-b reg frect te o 754 e nc ion amo or +	it fi: iste tion r no -def ot en and ong -∞:	xec r re , re t-a- îne abl no sou 2 ⁶³	d-p gg -n ed leo e	ooii 3. ardl um inv d, t xce ces	nt les lbe val he ept	for ss c er, o lid pr tion	rma of th or v op rese n o	nt, a he whe era erva erva ecu	und cur en t tion atic rs.	sto rent he i n ex on b Tho	t rour round cepti it (bi	ne r ndir led on t 4) rn	ng r res is c of	lt in nod ult i lete the	n th le. is o cte FP	ne re outsi d. PSR	gis de t	ter
[Floating-point operation exceptions]	Inv	imple alid o xact e	pera	atio	n ez	хсер			-	ion	(E)																	

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.



FLOORF.	Floating-point Convert Single to Unsigned-Long, round toward negative (Single)
	Conversion to unsigned fixed-point format (single precision)
[Instruction format]	FLOORF.SUL reg2, reg3
[Operation]	reg3 \leftarrow floor reg2 (single \rightarrow unsigned long-word)
[Format]	Format F:I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16
	r r r r r 1 1 1 1 1 1 1 1 w w w w 0 1 0 0 0 1 0 0 0 1 0 0
	reg2 reg3 category type sub-op
[Description]	 This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3. The result is rounded in the -∞ direction, regardless of the current rounding mode. When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2⁶⁴ - 1 to 0, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources. Source is a positive number outside the range of 2⁶⁴ - 1 to 0, or +∞: 2⁶⁴ - 1 is returned. Source is a negative number, not-a-number, or -∞: 0 is returned.
[Floating-point operation exceptions]	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

FLOORF	SUW		FI	loatin	ig-po	int	Con	vert :	Sir	ngle	to U	Insig	nec	l-W	ord,	rou	nd	tow	ard	neę	gativ	e (Sir	ngle	э)
								Со	٥n	/ersid	on to	o un	sigr	ed	fixe	d-pc	oint	forr	nat	(sir	ngle	precis	sio	า)
[Instruction format]	FLOORF	SUW	reg2, re	eg3																				
[Operation]	reg3 ← fl	oor reg	2 (singl	le →	• uns	igr	ned	wore	d)															
[Format]	Format F	Ι																						
[Opcode]	15 11	10		F	4			0		0.4			07	26	25		^ 2	22	01	20		4-	7 1	6
	15 11 r r r r r	10	1 1 1	5 . 1	4	0	0 3		3	w w	w	w	27 W	1	25 0	0		22 1			0		7 1	0
	reg2					-	-		1		reg			 		tego					sub-			
[Description]	This instr general-p general-p The result When the result is o detected. If invalid register is follows, a • Sour • Sour	urpose urpose is roun source utside operati set as ccordin ce is a	register register nded in operan the rang on exce an inval ng to dif positive	reg. reg. the - d is ge of lid o ffere	2 to $\frac{1}{3}$. $-\infty$ d infir 2^{32} ms and pera ncess mber	lire nite - 1 re 1 tio s ar	sign ectio e, no l to n an non utsid	ed 3 n, re t-a-1 0, an enab d no g so e th	eg nu n l ole ur	-bit t ardl- imbe IEEI ied, th exce cces. rang	fixe ess er, c E75 me p ptic	ed-p of t or no 4-d oreso on o	oin he ega efir erva ccu	t fo cur tive ned atio urs. 1 te	orma ren e nu inv on b Th	at, a t ro umb calic it (l e re	and uncer, er, d op tit tur $+\alpha$	l sto ding or pera 4) c n v	g m wh atic of t alu	s th odd en on e he e d	e res e. the r exce FPS iffer	sult i round ptior R rs as	in deo n is	d S
[Floating-point operation exceptions	Unimpler] Invalid op Inexact ex	eratior	n except		-	tio	n (E)																

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (integer)	0 [V]	0 (int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

FLOORF.	SW		Floating-point	Convert Single to	Word, round	toward n	egative (Sing	gle)
				Conversion	to fixed-point	t format (s	single precisi	ion)
[Instruction format]	FLOORF.SW reg	2, reg3						
[Operation]	reg3 ← floor reg2	(single \rightarrow w	vord)					
[Format]	Format F:I							
[Opcode]								40
	15 11 10 r r r r r r 1 1 1 reg2	<u>54</u> 1110	0 0 0 0 1 1	31 27 wwwww	<u>26 25 23</u> 1 0 0 0	1 0	<u>0 17</u> 0 0 0 0	16 0
	reg2			rea3	category	type	sub-op	
[Description]	This instruction and general-purpose register re- purpose register re- The result is round When the source of range of $2^{31} - 1$ to If invalid operation register is set as and follows, according	egister reg2 t eg3. ded in the $-\infty$ perand is inf $p - 2^{31}$, an IEI n exceptions n invalid ope to difference	to 32-bit fixed direction, reg finite or not-a- EE754-defined s are not enable eration and no ces among sou	-point format, a ardless of the c number, or whe d invalid operat ed, the preserva exception occu rces.	nd stores th urrent round in the round ion exception tion bit (bit	ding model ding model ding model ed result on is det 4) of the	in general- de. is outside t ected. e FPSR	
				-1 is returned.	1:	ı		
[Floating-point operation exceptions]	• Source is a number of the source o	peration exce exception (V	eption (E)	wher, or $-\infty$: -2^3	¹ is returned	d.		

[Operation result]

reg2 (A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	+Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

.

FMAF.S													F	loat	tina	-noi	int f										-		cisic	
[Instruction format]		FM	AF.S	S	regl	, re	g2,	reg	3						ung			<u>usc</u>	<u>u-11</u>		<u>Jiy-</u>	100		crat			igic	pre		<u>,,,,</u>
[Operation]		reg	3 ←	fr	na (r	eg2	2, re	:g1,	reg	(3)																				
[Format]		For	mat	F:	Ι																									
[Opcode]																														
	15 r :	r r			10 1	1 1	1 1	1		4 R		R	R		31 w		w	w			25 0		23 1	1			0	0	17 0	16 0
		reg	2								I	reg1	1				reg	3		ļ	ca	teg	ory	ty	ре		sub	o-op		
[Description]		pur pur pur mu	s ins pose pose pose ltiply ndec	e re e re e re y c	egist egist egist opera	er r er r er r atio	reg2 reg1 reg3 n is	2 wi , ad , an , an	th th lds t nd st t rou	he s the tore und	sing sing es the	gle-p gle- ne ro duri	preo -preo esu ing	cisi ecis lt i thi	on ion n g s o	flo flo ene per	atir Dati Pral Pratic	ng-j ng- -pu on,	poii -poi rpo but	nt f int ose	orn fori reg	nat mat iste	coi t co er re	nter onte eg3	nts i ents . Tl	in g in he 1	gene gene gen	eral nera ult c	l- 1l- of th	
[Floating-point operation exceptions]	Inv Ine Ove	impl alid xact erflo derfl	op ex w	erat kcept exce	ion tion epti	exc (I) on (cept (O)	ion		-	on ((E)																	



RH850G3MH Software

[Operation result]

	reg2(B)								
reg3(C)	reg1(A)	+ Normal	– Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
	+Normal					+∞	_∞		
	-Normal		FMA (/	A, B, C)		_∞	+∞		
±Normal	±0					Q-N	aN[V]		
	+∞	+∞	_∞			+∞	_∞		
	_∞	-∞	+∞	Q-Ni	aN[V]	_∞	+∞		
	+Normal					+∞	-∞		
	–Normal		FMA (/	A, B, C)		-∞	+∞		
±0	±0					Q-N	aN[V]		
	+∞	+∞	_∞			+∞	_∞		
	_∞	_∞	+∞	Q-N	aN[V]	_∞	+∞		
	+Normal			•		+∞	Q-NaN[V]		
	–Normal		+	-∞		Q-NaN[V]	+∞		
+∞	±0					Q-N	aN[V]		
	+∞	+∞	Q-NaN[V]		aN[V]	+∞	Q-NaN[V]		
	_∞	Q-NaN[V]	+8	Q-N	antv	Q-NaN[V]	+∞		
	+Normal					Q-NaN[V]	_∞		
	-Normal		-	-∞		_∞	Q-NaN[V]		
_8	±0					Q-N	aN[V]		
	+∞	Q-NaN[V]	_∞		aN[V]	Q-NaN[V]	_∞		
	_∞	-∞	Q-NaN[V]	Q-N	antol	_∞	Q-NaN[V]		
	±Normal								
Q-NaN	±0			Q-I	NaN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								
S-NaN	Don't care								Q-NaN[V]

Note 1. [] indicates an exception that must occur.

Note 2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **Section 6.1.8**, **Flushing Subnormal Numbers**.

[Supplement]

The result of the multiply operation is not rounded during this operation, but the result of the add operation is rounded, in accordance with the current rounding mode. The result therefore differs from the result obtained when using a combination of the ADDF and MULF instructions.

FMSF.S																F	loat	ing	-po	nt F	use	ed-l	Multi	iply-	-sut	otrac	ct (S	ingl	e)
											Flo	oatir	ng-	poir	nt fu	sed	-mu	ıltip	ly-s	ubti	ract	ор	erati	on	(sin	igle	prec	cisio	n)
[Instruction format]		FM	SF.S	regl	l, re	eg2,	reg3	3																					
[Operation]		reg	3 ← 1	fms (reg	2, re	eg1,	reg.	3)																				
[Format]		For	mat F	F:I																									
[Opcode]																													
	15 r	r r		1 10 1 1	1	1 :	1 1		4 R	R	R	R		31 w	W	w			26 1			23 1	1			0		17 ' 1	1 <u>6</u> 0
		reg2	2	Ì						r	reg1				r	eg3				ca	tegr	ory	typ	be		sub-	-op		
[Description]		purj purj purj mul	s inst pose pose pose tiply punde	regis regis regis oper	ter ter ter atic	reg2 reg1 reg3 on is	2 wit 1, sul 3, and 5 not	h th btra d st rou	ne s octs ore nde	the thes the ed d	le-p sin ne re lurii	orec ngle esul	cisi e-pr lt in this	on reci n g s op	floa isio ene oera	atin n fl ral- tior	g-p oat pui n, b	ing po ut	nt f g-po se i	orn oint regi	nat t fo iste	coi rm er re	nten at co eg3.	its i ont . Th	in g ent ne r	gene s in resu	eral- gei ilt o	nera f th	e
[Floating-point operation exceptions]	Inva Inez Ove	mple alid o kact e erflov lerflo	opera excep v exc	tion otion cept	n exo n (I) ion	cepti) (O)	on		-	on (E)																	



RH850G3MH Software

[Operation result]

$\overline{\ }$	reg2(B)								
reg3(C)	reg1(A)	+Normal	–Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN
	+Normal					+∞	-∞		
	–Normal		FMS (A	A, B, C)		-∞	+∞		
±Normal	±0					Q-N	aN[V]		
	+∞	+∞	_∞	Q-N	aN[V]	+∞	_∞		
	_∞	_∞	+∞			_∞	+∞		
	+Normal		L			+∞	_∞		
	–Normal		FMS (/	A, B, C)		_∞	+∞		
±0	±0					Q-N	aN[V]		
	+∞	+∞	_∞	Q-N	aN[V]	+∞	_∞		
	_∞	_∞	+∞			_∞	+∞		
	+Normal		1			Q-NaN[V]	_∞		
	–Normal		-	-∞		_∞	Q-NaN[V]		
+∞	±0					Q-N	aN[V]		
	+∞	Q-NaN[V]	_∞	Q-N	aN[V]	Q-NaN[V]	_∞		
	_∞	_∞	Q-NaN[V]			_∞	Q-NaN[V]		
	+Normal		I			+∞	Q-NaN[V]		
	–Normal		+	-∞		Q-NaN[V]	+∞		
_∞	±0					Q-N	aN[V]		
	+∞	+∞	Q-NaN[V]	Q-N	aN[V]	+∞	Q-NaN[V]		
	_∞	Q-NaN[V]	+∞	-		Q-NaN[V]	+∞		
	±Normal								
Q-NaN	±0			Q-I	NaN				
	±∞	1							
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								
S-NaN	Don't care	1							Q-NaN[V

Note 1. [] indicates an exception that must occur.

Note 2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **Section 6.1.8**, **Flushing Subnormal Numbers**.

[Supplement]

The result of the multiply operation is not rounded during this operation, but the result of the subtract operation is rounded, in accordance with the current rounding mode. The result therefore differs from the result obtained when using a combination of the SUBF and MULF instructions.

FNMAF.S	5										Fl	oating	g-poi	nt F	use	d-Neg	ate-M	ultipl	y-add	(Sing	gle)
								F	loat	ting	-poir	nt fuse	ed-m	ultip	oly-a	idd op	eratio	n (sir	ngle p	recisi	on)
[Instruction format]		FNMAF.	S reg	l, reg2	2, reg	3															
[Operation]		reg3 ← n	leg (fn	na (reg	g2, re	g1,	reg3))													
[Format]		Format F	:I																		
[Opcode]																					
	15	11	10			5 4	ļ		0	31			27	26	25	23	22 2	1 20		17	16
	r i	r r r r	1 1	1 1	. 1	1 F	RR	R R	R	w	W	w w	W	1	0	0 1	1 3	L O	0 3	1 0	0
		reg2					re	eg1			re	eg3			cat	egory	type	;	sub-c	р	
[Description]		This instr purpose r purpose r purpose r The resul add opera reversed	registe registe registe t of th ation is	r reg2 r reg1 r reg3 e mul s roun	with , adds , inve tiply ded,	the s the erts t open	singl sing the si ration	e-pre le-pro gn, an	cisi ecis nd s ot re	on ion stor	floa floa es th ded	ting- ating he res durin	poii -poi sult ng ti	nt f int f in g his	orm form gene ope	at co nat co eral-p ration	ntents onten ourpos n, but	s in g ts in se re the	gener gene gister resul	al- ral- r reg3 t of t	
[Floating-point operation exceptions	;]	Unimplet Invalid o Inexact e Overflow Underflo	peratio xcepti / excep	on exc on (I) otion (eptio		-	n (E)													



RH850G3MH Software

[Operation result]

$\overline{\ }$	reg2(B)								
reg3(C)	reg1(A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
	+Normal					_∞	+∞		
	–Normal		FNMA	(A, B, C)		+∞	_∞		
±Normal	±0					Q-N	aN[V]		
	+∞	_∞	+∞			_∞	+∞		
	_∞	+∞	_∞	Q-N	aN[V]	+∞	_∞		
	+Normal					_∞	+∞		
	–Normal		FNMA	(A, B, C)		+∞	_∞		
±0	±0					Q-N	aN[V]		
	+∞	_∞	+∞		aN[V]	_∞	+∞		
	_∞	+∞	_∞	Q-N	antvj	+∞	_∞		
	+Normal					_∞	Q-NaN[V]		
	–Normal		-	-∞		Q-NaN[V]	_∞		
+∞	±0					Q-N	aN[V]		
	+∞	-∞	Q-NaN[V]		aN[V]	_∞	Q-NaN[V]		
	_∞	Q-NaN[V]	_∞	Q-N	antvj	Q-NaN[V]	_∞		
	+Normal					Q-NaN[V]	+∞		
	–Normal		+	-∞		+∞	Q-NaN[V]		
_8	±0					Q-N	aN[V]		
	+∞	Q-NaN[V]	+∞	Q-N	aN[V]	Q-NaN[V]	+∞		
	_∞	+∞	Q-NaN[V]			+∞	Q-NaN[V]		
	±Normal		-	•			•		
Q-NaN	±0			Q-I	NaN				
	±∞	1							
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								
S-NaN	Don't care								Q-NaN[\

Note 1. [] indicates an exception that must occur.

Note 2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **Section 6.1.8**, **Flushing Subnormal Numbers**.

[Supplement]

The result of the multiply operation is not rounded during this operation, but the result of the add operation is rounded, in accordance with the current rounding mode. The result therefore differs from the result obtained when using a combination of the ADDF, MULF, and NEGF instructions.



FNMSF.S										Flo	batin	ıg-poi	nt F	use	d-N€	egate-	Mult	iply-:	sub	otract	(Sing	gle)
								Float	ing-j	ooin	t fus	ed-m	ultip	ly-s	ubtr	act op	erat	ion (sin	gle pr	ecisi	on)
[Instruction format]	FNI	MSF.S	reg1	, reg2	, reg	3																
[Operation]	reg	$3 \leftarrow n$	eg (fm	s (reg	g2, re	g1,	reg3))														
[Format]	For	mat F:	I																			
[Opcode]			10			_							~-						~~			
	15 r r r	11 r r	10 1 1	1 1	1	5 4 1 H	l RR	R R		31 w	w	w w	27 w	26 1	0			21 2 1		0 1	17 . 1	16 0
	reg2	2					re	eg1			re	eg3			cat	egory	ty	pe	ŝ	sub-o	р	
[Description]	purj purj purj The sub	s instru- pose re pose re result tract o ersed a	egister egister egister t of the perati	reg2 reg1 reg3 e mul on is	with , sub , invo tiply rouno	the tract erts ope	singl ts the the si ration	e-pre singl gn, an	cisi e-pi nd s ot re	on f recis tore	floa sior es th led	ting- n floa ne res durii	pointing sult ng t	nt fo g-po in g his	orm oint gene ope	at co form eral-p ration	nter at c ourp n, bo	nts in onte ose : ut th	n g ents reg ie r	enera s in g gister esult	al- gener reg3 t of tl	3. he
[Floating-point operation exceptions] Inva Inez Ove	mplen alid op xact ex erflow derflow	oeratio kceptio excep	n exc on (I) otion (eptic		-	n (E)														



RH850G3MH Software

[Operation result]

	reg2(B)								
reg3(C)	reg1(A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
	+Normal					-∞	+∞		
	-Normal		FNMS ((A, B, C)		+∞	_∞		
±Normal	±0					Q-N	aN[V]		
	+∞	_∞	+∞		- N I() /]	_∞	+∞		
	_∞	+∞	_∞	Q-N	aN[V]	+∞	_∞		
	+Normal		L			_∞	+∞		
	-Normal		FNMS ((A, B, C)		+∞	_∞		
±0	±0					Q-N	aN[V]		
	+∞	_∞	+∞		- N IT) /1	_∞	+∞		
	_∞	+∞	_∞	Q-Ni	aN[V]	+∞	_∞		
	+Normal		1	1		Q-NaN[V]	+∞		
	-Normal		+	~~~~		+∞	Q-NaN[V]		
+∞	±0					Q-N	aN[V]		
	+∞	Q-NaN[V]	+∞		10.0	Q-NaN[V]	+∞		
	_∞	+∞	Q-NaN[V]	Q-Ni	aN[V]	+∞	Q-NaN[V]		
	+Normal					_∞	Q-NaN[V]		
	-Normal		-	.∞		Q-NaN[V]	_∞		
_∞	±0					Q-N	aN[V]		
	+∞	_∞	Q-NaN[V]			_∞	Q-NaN[V]		
	_∞	Q-NaN[V]	_∞	Q-N	aN[V]	Q-NaN[V]	_∞		
	±Normal								
Q-NaN	±0			Q-I	NaN				
	±∞	1							
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								
S-NaN	Don't care								Q-NaN[V

Note 1. [] indicates an exception that must occur.

Note 2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **Section 6.1.8**, **Flushing Subnormal Numbers**.

[Supplement]

The result of the multiply operation is not rounded during this operation, but the result of the subtract operation is rounded, in accordance with the current rounding mode. The result therefore differs from the result obtained when using a combination of the SUBF, MULF, and NEGF instructions.

MAXF.D																		Flo	pating	g-p	oin	t Ma	axir	num	ı (Do	bub	e)
														Floa	iting	g-po	oint	ma	kimu	m١	/alı) e (dou	ble j	prec	isic	n)
[Instruction format]		MAXF.D	reg]	l, reg	2, reg	g3																					
[Operation]		reg3 ← r	nax (1	eg2,	reg1))																					
[Format]		Format F	:I																								
[Opcode]	45		10			_					•					~ -	~~~	0-			~~		~~				
	15 r :	11 r r r 0	10	L 1	1 1	1	4 R	R	R	R		31 w	W	w		<u>27</u> 0	26 1	25 0	0		<u>22</u> 1	21 1		1		0	16 0
		reg2						r	eg1				I	reg3				cat	egor	у	typ	ре		sub-	-op		
[Description]		This insta data in the the regist If one of detected. occurs.	e reg er pai the so	ister p ir spe ource	pair s cified oper	pec d by and	ifie / ge ls is	ed by eners S-1	y ge al-p NaN	ene urț [, a	ral oos n I	-pu se ro EE	rpc egis E7:	ose 1 ster 54-0	regi reg lefi	iste g3.	ers i d ii	reg	l an lid o	d r pe	eg2 rat	2, a	nd ex	stoi cep	res tior	it in 1 is	1
[Floating-point operation exceptions	;]	Invalid o	perati	on ex	cept	ion	(V))																			
[Supplement]		When bo	-		-																				in 1	eg.	3.
		A subnor	mal i	nput v	will r	not	be f	flusl	hed	ev	en	if t	he	FS I	oit	of	the	FP	SR 1	eg	ist	er i	s 1				



RH850G3MH Software

[Operation result]

reg2(B)								
reg1(A)	+Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0			MAN				no n1 (A)	
-0			MAX	(А, В)			reg1 (A)	
+∞								
-∞								
Q-NaN			reg2	2 (B)			Q-NaN	
S-NaN								Q-NaN [V]

Note: [] indicates an exception that must occur.



MAXF.S																				F	oating	g-po	oint	Max	timi	um (S	Sing	le)
																Floa	atin	g-p	oin	t ma	ximur	n v	alue	e (si	ngle	e pre	cisio	on)
[Instruction format]	1	MAXF	S.S	regl	, reg	g2, r	eg3																					
[Operation]	I	reg3 ←	- m	nax (1	eg2	, reg	g1)																					
[Format]	I	Forma	tF:	Ι																								
[Opcode]	15		11	10				5	1				0	31				27	26	25	22	2.00	, ,	1 20	'n		17	16
	r r	r r	r		L 1	1	1	5 1	4 R	R	R	R		Ī	w	W		w	1	0			. 1	1 20 - 0		0	0	0
	r	eg2								r	eg1				r	eg3				cat	egory	t	уре	•	su	b-op		
[Description]		This in lata in f one letecte occurs	ge of t d.	neral he so	-pui ourc	rpos e op	e re erai	gis nds	ters is	s re S-N	gl a NaN	anc I, a	d re in I	eg2 EE	, an E7:	d st 54-d	ore lefi	s it ne	t in d ii	gei ival	neral- lid op	-pu bera	rpo atio	ose i on e:	egi xce	ister ptio	reg n is	<u>3</u> .
[Floating-point operation exceptions		nvalid	op	oerati	on e	exce	ptio	on (V)																			
[Supplement]		When I		-			-																			ed in	reg	g3.

[Operation result]

reg2(B)								
reg1(A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0			MAX				no n1 (A)	
-0			MAX	(А, Б)			reg1 (A)	
+∞								
-8								
Q-NaN			reg2	2 (A)			Q-NaN	
S-NaN								Q-NaN [V]

Note: [] indicates an exception that must occur.



MINF.D															Flo	oating	-poi	int N	1inin	านท	(Doul	ble)
												Float	ing-p	ooin	mir	nimum	ı val	ue (dou	ble p	recisi	ion)
[Instruction format]	М	INF.D	reg1, r	reg2, re	eg3																	
[Operation]	re	g3 ← m	in (reg	g2, reg	1)																	
[Format]	Fc	ormat F:	I																			
[Opcode]	45	44	10		F				0	24			07	00	05	00		04	20		47	10
	15 r r r	11 r r 0		1 1	5 1 1	1	R F	r r		31 w	w	ww		26 1	0		Ĩ	21 1		1 (16 0
	re	g2					re	g1			r	eg3			cat	egory	ty	pe	:	sub-c	ор	
[Description]	da the If de	nis instru ta in the e registe one of t etected. I ecurs.	e regis er pair he sou	ter pai specif irce op	r spec ied by erand	ifie / ge ls is	d by meral S-N	gen l-pur aN, s	eral pos an I	-pu se re EE	rpo egis E75	se re ter ro 54-de	giste eg3. efine	ers i	reg1 nval	and id op	reg era	2, a tion	nd ex	store	es it i	in s
[Floating-point operation exceptions		valid op	eratio	n exce	ption	(V))															
[Supplement]	W	hen bot	h reg1	and re	eg2 is	eitl	her +	0 or	-0,	wh	eth	er +() or	-0	is st	ored	in r	eg3	is	unde	efine	d.
	А	subnorr	nal inj	out wil	ll not	be f	flushe	ed ev	ven	if tl	he I	FS bi	t of	the	FPS	SR re	gist	ter i	s 1.			



RH850G3MH Software

[Operation result]

reg2(B) reg1(A)	Normal	-Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal				-				
-Normal								
+0			N di N d					
-0			MIN ((А, В)			reg1 (A)	
+∞								
-∞								
Q-NaN			reg2	2 (B)			Q-NaN	
S-NaN								Q-NaN [V]

Note: [] indicates an exception that must occur.



MINF.S															Floa	ating	-poi			-					m (S prec		
[Instruction format]		MINF.S	5 1	reg1,	reg2	2, re	g3										-										
[Operation]		reg3 ←	m	nin (re	eg2,	reg	1)																				
[Format]		Format	F:	Ι																							
[Opcode]																											
	15 r 1		11 r	10 1 1	. 1	1	5 1 1	1	R	R	R		31 w	W	w	27 w w	7 26 1		0	<u>23</u> 0	22 1	21 1		1		17 · 1	16 0
	 	reg2							r	reg1				r	eg3			ca	tego	ory	ty	/pe		sub	-op		
[Description]		This ins data in g If one o detected occurs.	ge of t	neral he sc	-pur	pose e ope	e reg eranc	istei 1s is	rs re S-l	eg1 : NaN	anc J, a	l re n I	eg2 EE	, an E7:	d sto 54-d	ores efin	it ir ed i	n ge nva	ner lid	al-p ope	pur era	pos tion	se re	egis cep	ster : otior	reg 1 is	3.
[Floating-point operation exceptions]	Invalid	oŗ	oerati	on e	xcep	otion	(V))																		
[Supplement]		When b A subno		-			-															-			lefir	ned	



RH850G3MH Software

[Operation result]

reg2(B) reg1(A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
	Normai	-Normai	ŦŪ	-0	+~		Q-INAIN	3-Ivalv
+Normal								
-Normal								
+0			MIN ($rog1(\Lambda)$	
-0				(А, Б)			reg1 (A)	
+∞								
-∞								
Q-NaN			reg2	2 (B)			Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.



MULF.D																F	loati	ng-	-point	Mult	iply (Doub	ole)
													Flo	ating	g-po	int n	nultip	lica	ation (dout	ole pr	ecisio	on)
[Instruction format]		MULF	.D	reg1,	reg2, 1	reg3																	
[Operation]		reg3 ←	- re	eg2 × r	eg1																		
[Format]		Format	tF:	Ι																			
[Opcode]				10		_								~-		0.5						47	4.0
	15			10		5					31				26				22 21		0 1	17	
	r r	rr	0		1 1	1 1	R	R R	R	0	W	W	w w	0	1	0	0 (1 1		0 1	0	0
		reg2						reg	1			r	eg3			cat	egor	y	type	s	sub-o	р	
[Description]		This in specific content general	ed i ts c	by ger of the r	neral-p register	urpos r pair	se re spe	egister	reg	2 b	y tl	he d	oubl	e-pr	eci	sion	floa	ntin	ng-po	int f	form	at	
[Floating-point operation exceptions]]	Unimp Invalid Inexact Overfle Underf	l op t ex	eratio ceptic excep	n exce on (I) tion (C	ption D)		-	(E)														

[Operation result]

reg2(B)								
reg1(A)	Normal	-Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal					+∞	_∞		
-Normal		Δ.	× B		_∞	+∞		
+0		A ·	* D					
-0					Q-Na	aN [V]		
+∞	+∞	_∞	Q-Na	N [V]	+∞	_∞		
_ ∞	8	+∞			_∞	+∞		
Q-NaN					•	•	Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.

MULF.S																		ing-poi				-	
													FI	oatin	g-p	oint r	nultip	licatio	n (sir	gle	prec	sior	<u>ו</u>)
[Instruction format]		MULF	.S	reg1,	reg2,	reg3																	
[Operation]		reg3 ←	- re	$g_2 \times g_2$	reg1																		
[Format]		Forma	tF:	I																			
[Opcode]																							
	15		11	10		5	4			0	31			27	26	25	23	3 22 2	1 20			17 1	6
	rı	rr	r	1 1	1 1	1 1	R	RI	R R	R	w	W	w w	7 W	1	0	0 0	1 1	0	0	1	0 (0
		reg2						re	g1			re	eg3		l	cate	egory	type		sub	-ор		
[Description]		This in purpos purpos	e re	egiste	r reg2	by the	sir	ngle-p	oreci	sio	n fl	oatir	ıg-p	oint	for	mat	cont	ents o		-		-	
[Floating-point operation exceptions]	Unimp Invalid Inexac Overfle Underf	l op t ex ow	eratic ception except	on exc on (I) otion (eption O)		-	n (E)														

[Operation result]

reg2(B) reg1(A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
+Normal					+∞	_∞		
-Normal		A			_∞	+∞		
+0		Α >	¢В		0.11			
-0					Q-Na	aN [V]		
+∞	+∞	_∞	0.14		+∞	_∞		
-∞	-8	+∞	Q-Na	N [V]	_∞	+∞		
Q-NaN		1			1	1	Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.

NEGF.D	Floating-point Negate (Double)
	Floating-point sign inversion (double precision)
[Instruction format]	NEGF.D reg2, reg3
[Operation]	$reg3 \leftarrow neg reg2$
[Format]	Format F:I
[Opcode]	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16
	r r r r 0 1 1 1 1 1 1 0 0 0 0 1 w w w w 0 1 0 0 0 1 0 1
	reg2 reg3 category type sub-op
[Description]	This instruction inverts the sign of double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in general-purpose register reg3.
[Floating-point operation exceptions	None]
[Supplement]	A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.



NEGF.S														Flo	pating	q-pc			ing-poi versior		-		-	
[Instruction format]		NEGF.	S	reg2,	reg3											<u> </u>		0			0	<u>.</u>		
[Operation]		reg3 ←	- n	eg reg	<u>,</u> 2																			
[Format]		Forma	t F:	Ι																				
[Opcode]																								
	15 r 1	rr		10	1 1				0			31 w	w	ia7 Ta		26 1		<u>23</u> 0 0	22 2 ²			0	17 1 0	16 0
		reg2	-							0	÷			g3				gory				o-op		
[Description]		This in genera					-				-	-										s of		
[Floating-point operation exceptions]	None																						
[Supplement]		A subr	nori	nal in	iput w	vill not	be	flus	shed	l eve	en	ift	he F	S bi	it of	the	FPS	R re	gister	is 1				



RECIPF.C	Reciprocal of a Floating-point Value (Double)
	Reciprocal (double precision)
[Instruction format]	RECIPF.D reg2, reg3
[Operation]	$reg3 \leftarrow 1 \div reg2$
[Format]	Format F:I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16 r r r r 1
[Description]	This instruction approximates the reciprocal of the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in the register pair specified by general-purpose register reg3. The result differs from the result obtained by using the DIVF instruction.
[Floating-point operation exceptions]	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I) Division by zero exception (Z) Underflow exception (U)

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	1//	N [I]	–∞ [Z]	–∞ [Z]	+0	-0	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.

RECIPF.S	5											Re	ecipi	OCa	al of a	Floa	ating-po	oint Va	lue (S	Singl	e)
																Red	iprocal	(single	e prec	cisio	n)
[Instruction format]		RECIPF	S reg2	2, reg3																	
[Operation]		reg3 ← 1	l ÷ reg	2																	
[Format]		Format I	F:I																		
[Opcode]																					
	15	1	1 10		5	4			0	31			27				22 21			17 1	16
	rı	rrr	1 1	1 1	1 1	0	0 C	0	1	W	W W	W	W	1	0 0	0 0	1 0	0 1	. 1	1	0
		reg2									reg	3			cate	gory	type	su	ıb-op		
[Description]		This inst contents reg3. Th	of gen	eral-pu	rpose	regis	ster 1	reg2,	, ar	nd s	stores	the	res	ult	in ge	nera	al-purp	ose r			
[Floating-point operation exceptions]	Unimple Invalid o Inexact o Division Underflo	peratio excepti by zer	on exce on (I) o excej	ption	(V)	tion	(E)													

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	1/A	A [I]	+∞ [Z]	–∞ [Z]	+0	-0	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



RSQRTF	D							Red	cipro	cal o	of the	e Squ	are l	Roo	t of a	a Floa	ting-po	int V	alue (Doul	ble)
												F	Recip	oroc	al of	squa	re root	(dou	ble pr	ecisi	ion)
[Instruction format]	RSC	RTF	D reg	2, reg3																	
[Operation]	reg3	← 1	÷ (sqr	t reg2)																	
[Format]	For	nat F	:I																		
[Opcode]	15	11	10		5	5 4			0	31			27	26	25	23	22 21	20		17	16
	rrr			1 1				0			w	w w		1		0 0	1 0	1	1 1		0
	reg2										re	eg3			cate	egory	type		sub-o	þ	
[Description]	poin appi gene The	t fori oxim eral-p	mat con nates th purpose It diffen	obtains ntents c le recip e registe rs from	of the roca er reg	e reg l of g3.	giste this	r pai resu	r spo lt an	ecifi d st	ied l	by go s the	ener rest	al-j ilt i	purp in th	ose r e reg	egiste ister p	r reg air s	2, the pecif	en ied	by
[Floating-point operation exceptions	5] Inva Inex	lid o act e	peratio xceptio	l operat n excep on (I) o excep	otion	(V)	on (E)												

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN
Operation result [exception]	1/√-A [I]	Q-NaN [V]	+∞ [Z]	–∞[Z]	+0	Q-NaN [V]	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



RSQRTF.											Rec	ipro	cal	of t	he	Squ	lare	Ro	ot o	faF	-loa	ating-po	oint \	/alu	ıe (S	ingl	e)
																	Rec	ipro	cal	of se	qua	are root	(sin	gle	prec	isio	n)
[Instruction format]		RSQRT	F.S	reg	2, re	g3																					
[Operation]		reg3 ←	1 ÷	(sqr	t reg	g2)																					
[Format]		Format I	F:I																								
[Opcode]																											
	15		1 10		_		5	4			_		31					1	25			22 21	1			7 1	
	r	rrrı	. 1	1	Ţ	1.	1 1	0	0	0	1	0	W	W	W	W	W	1	0	0	0	1 0	0	T	1	1	0
		reg2												I	reg	3			ca	tego	ory	type	5	sub	-ор		
[Description]		This inst point for this resu obtained	rma lt ai	t con nd s	nten tores	ts o s it i	f gen in ge	iera ner	ıl-p al-p	urp pur	ose pos	reg e re	gist egis	er i ster	reg reg	2, t g3.	her Tł	n ap ne r	pro esu	oxin lt d	nate iffe	es the ers fro	reci	pro	ocal	of	
[Floating-point operation exceptions	5]	Unimple Invalid o Inexact o Division	oper exce	atio eptic	on ex on (I	cep	otion	(V)	-	on	(E)																

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Operation result [exception]	1/√-A [I]	Q-NaN [V]	+∞ [Z]	–∞[Z]	+0	Q-NaN [V]	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



SQRTF.D														F	loatin	ј-рој	int Squ	are Ro	oot (C	oub	le)
															S	qua	re root	(doub	e pre	cisio	on)
[Instruction format]		SQRTF.D	reg2,	reg3																	
[Operation]		reg3 ← so	qrt reg2																		
[Format]		Format F:	I																		
[Opcode]																					
	15	11	10		5	4			0	31			27	26	25	23	22 21	20		17	16
	r	rrr O	1 1	1 1 1	1	0	0 0	0 0	0	w	w w	W	0	1	0 0	0	1 0	1 1	. 1	1	0
		reg2									reg	3			cate	jory	type	รเ	ıb-op		
[Description]		This instruction obtains the arithmetic positive square root of the double-precision floating- point format contents of the register pair specified by general-purpose register reg2, and stores the result in the register pair specified by general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode. When the source operand value is –0, the result becomes –0.													n						
[Floating-point operation exceptions]	Unimplen Invalid op Inexact ex	eration	excepti			-	(E)													

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Operation result [exception]	√A	Q-NaN [V]	+0	-0	+∞	Q-NaN [V]	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



SQRTF.S														F	loati	ng-po	oint Sq	uare	Root	: (Sin	gle)
																Squa	are roo	t (sin	gle p	recisi	ion)
[Instruction format]		SQRTF.S	reg2,	reg3																	
[Operation]		reg3 ← so	qrt reg2	2																	
[Format]		Format F	I																		
[Opcode]	45	44	10		_	4			•	0.1			07	00	05	00				47	. 40
	15 r 1		10	1 1	5 1 1	4 0	0 0	0		31 w	w	w w		26 1	0		1 0		1 :		0 0
		reg2									r	eg3			cate	egory	type		sub-c	эр	
[Description]		This instr point forr register re rounded i the result	nat con eg3. Th n accor	tents o e opera dance	of gen ation	eral is e	-purp xecut	pose ted a	reg as i	gist f it	er r wer	eg2, re of	and infi	sto nite	res i acc	t in g uracy	genera y, and	l-pu the	rpos resul	e lt is	
[Floating-point operation exceptions]]	Unimpler Invalid of Inexact ex	peration	n excep		-	otion	(E)													

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	√A	Q-NaN [V]	+0	-0	+∞	Q-NaN [V]	Q-NaN	Q-NaN [V]

Note 1. [] indicates an exception that must occur.



SUBF.D																Floa	ating	J-poin	t Sul	otrac	t (Do	uble
													Flo	ating	-poii	nt si	ubtr	action	(do	uble	prec	sion
[Instruction format]	SU	BF.D	reg1, reg2	2, reg3	;																	
[Operation]	reg	3 ← r	reg2 – reg1																			
[Format]	Fo	mat F	F:I																			
[Opcode]																						
	15 r r r		1 10 1 1 1 1	1 1		4 D	D	ם ם		31		w		27 20 0 1		0		22 2 1 1		0		7 10
	reg			1 I	Ŧ	R		eg1	. 0	w		reg3				ateg		type			-op	
[Description]	pai for res as	r spec mat co ult in t if it wo	ruction sul ified by ge ontents of the registe ere of infin mode.	eneral- the reg r pair s	·pu gist spee	rpos er p cifie	se ro air ed b	egist spec by ge	er ro ifieo nera	eg1 d by al-p	fro y ge urp	m tl mera ose	ne d al-p reg	loub urpo ister	le-p ose i reg	rec regi 3. T	isio iste The	n flo r reg2 opera	atin 2, ar atio	g-po nd si n is	oint tores exec	the
[Floating-point operation exceptions] Inv Ine Ov	alid o xact e erflow	mented op peration e exception (v exception ow exception	xcepti I) 1 (O)	on	-		on (E)													


[Operation result]

reg2(B)								
reg1(A)	Normal	–Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN
Normal								
-Normal		Р	- A		+∞			
+0		D -	- A		+∞	_∞		
-0								
+∞		-	.∞		Q-NaN [V]			
_∞			+∞		[*]	Q-NaN		
						[V]		
Q-NaN						•	Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.



SUBF.S																		Fle	batin	g-po	oint	Subt	rac	t (Sir	ngle)
															F	oati	ng-	point	subt	ract	ion	(sing	le	oreci	sion)
[Instruction format]		SUBF	S	reg1,	reg2, r	reg3																			
[Operation]		reg3 ←	– re	eg2 –	regl																				
[Format]		Forma	t F	:I																					
[Opcode]				10			_											~-							
	15 r	rrr		10 1 1	1 1	1		4 R	R	RI		31 w		w	W	27 2 w	26 1	25 0 0			21 1		0	0 1	7 16 0
		reg2							r	eg1				reg	3			cate	jory	ty	pe	s	ub-	ор	
[Description]		This in reg1 fro the resu and the	om ult i	the sin n gene	gle-pro eral-pu	ecisio rpose	on fl e reg	loat gist	ting- er re	poin g3. 7	t forr The c	nat per	cor ratio	ntent on is	s of exe	gen cute	era ed a	ll-purj as if it	oose we	reg	ister	regi	2, a	nd st	ores
[Floating-point operation exceptions	5]	Unimp Invalio Inexac Overfl Under	d op et e: ow	peratio xcepti exce	on exc ion (I) ption (ception (O)	on		•	n (E)														

[Operation result]

reg2(B)								
reg1(A)	Normal	–Normal	+0	-0	+∞	-	Q-NaN	S-NaN
Normal								
-Normal		В -	٨		+∞			
+0		В-	- A		+∞	_∞		
-0								
+∞		_	×		Q-NaN [V]			
_∞			+∞			Q-NaN [V]		
Q-NaN							Q-NaN	
S-NaN								Q-NaN [V]

Note 1. [] indicates an exception that must occur.

TRFSR	-										Tı	rans	sfers	spe	ecifie	ed (CCI	bit t	o Z	ero f	lag	in P	SW (\$	Sing	e)
																						F	lag ti	rans	er
[Instruction format]		TRFSR TRFSR	fcbit																						
[Operation]	I	PSW.Z ←	– fcbi	t																					
[Format]	I	Format F	:I																						
[Opcode]						_									~-	~~	~-		~~						
	15 0 0	0 0 0	1 10 1 1	. 1 1	L 1	5 1	4 0	0	0	0 0	31 0	0	0		27 0	26 1		0	23 0	22 2 0			f	17 f	16 0
	L												reg3	3	ĺ		cat	ego	ory	typ	e	รเ	ıb-op		
	Ν	Note: fcl	bit: ffi	Ē																					
[Description]	r	This instr egister s he CC0	pecifi	ed by							· ·														rs
[Floating-point operation exceptions		None																							



TRNCF.D	L						Floa	ating-p	poi	nt C					-			zero (E Jble pre		
[Instruction format]	TR	NCF.E	DL reg	2, reg3													<u></u>			<u> </u>
[Operation]	reg3	5 ← tr	unc re	g2 (doı	ıble –	→ lc	ong-w	ord)												
[Format]	For	nat F	:I																	
[Opcode]	15	11	10		5	4			0 :	31		27	7 26	3 25	23	22 21	20		17	16
	r r r reg2	r O	1 1	1 1	1 1	0	0 0	0	1	W	w w reg3	w 0	1	0 ca	0 0 tegory	1 0 type	1	0 1 sub-op	0	0
[Description]	the stor The Who rang If in regi follo	registores the resulten the ge of 2 avalid ster is pows, a Sour	er pair result t is rou source $2^{63} - 1$ operat s set as accordi cce is a	arithmo specifi in the r inded in e operation exc an inva ng to d positiv negativ	ed by register n the z nd is i 3, an I ception alid op iffere re num	ger per p zerco nfin EE: ns a pera nce	neral-j air sp o direct nite of E754- are no ation a s amo	purpo ecific etion, r not- defin t ena and r ong so $\infty: 2^{0}$	ose ed , re a-1 nec ble no our	e reg by egar num d in ed, 1 exc rces - 1	gister genera dless bber, o valid o the pro eption	reg2 al-pu of th or wh opera eserv occ	to rpo e cu en atio ratio urs.	64-t ose r urre the n n ex on b . Th	oit fixe registe nt rou rounde cceptic oit (bit e retur	ed-poi r reg3 nding ed resu on is d 4) of rn valu	nt f mo ult is letec the	òrmat, de. s outsi cted. FPSR	, and de t	d
[Floating-point operation exceptions] Inva	alid op		operat n excep on (I)		-	otion	(E)												

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_∞	Q-NaN	S-NaN
Operation result [exception]	A (int	eger)	0 (int	eger)	Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

.

<Floating-point instruction>

TRNCF.D	UL	Floating-p	Cor		-		I-Long, roi xed-point				
[Instruction format]	TRNCF.DUL reg2,	reg3									
[Operation]	$reg3 \leftarrow trunc reg2 (or a state of the state$	double \rightarrow uns	signed lon	ig-word)							
[Format]	Format F:I										
[Opcode]											
	15 11 10 r r r r 0 1 1 1	5 4		31		26 1	25 23 0 0 0	22 21 1 0	20 1 0		7 16
			001								
	reg2			I	eg3	ļ	category	1 1		b-op	
[Description]	This instruction arith the register pair spec format, and stores th The result is rounded When the source op- result is outside the detected. If invalid operation of register is set as an i follows, according to	cified by generative result in the zero of the result in the zero of the rand is infinite range of 2^{64} - exceptions are nvalid operative.	eral-purpo e register direction, ite, not-a- - 1 to 0, and e not enab- ion and no	se regist pair spec regardle number, n IEEE7 bled, the p except	er reg2 cified by ess of the or nega 54-defir preserva	to u y ge e cu tive ned	insigned ineral-pui irrent rou e number invalid c	64-bit rpose re nding r , or who operation	fixed- egiste mode. en the on exc he FP	-point er reg3 e round ception PSR	ded 1 is
	Source is a posSource is a neg			-				∞: 2 ⁶⁴ -	- 1 is	return	ed.
[Floating-point operation exceptions	Unimplemented ope Invalid operation ex Inexact exception (I	ception (V)	ion (E)								

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0 [V]	0 (Int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

TRNCF.D	U	W						FI	oa	iting-	poli	nt C															zero (uble pi			
[Instruction format]		TRNCF.	D	UW	reį	g2, 1	reg	3																						
[Operation]		reg3 ← :	tru	inc i	reg2	2 (do	oub	ole -	\rightarrow	uns	ign	ed	woi	rd	l)															
[Format]		Format I	F:I	-																										
[Opcode]	45			4.0				_													_							4-		
	15 r 1	1 r r r (1 ·		1 1	1	1	5 1	1	1 1 0	0	0		Т	31 w v	v	w			1 (0 0		22	1 20		0 0	17 0	16	٦
		reg2												l			eg3				ate	egory	ty	уре			ub-op	[<u> </u> 	1
[Description]			ter and lt e s ou d c is s ac	r pa: d sto is ro sour utsid oper set a ccorr ee is	ir spores ounce rece c le th aatio as an ding a po	the the ded oper ne ra n in g to ositi	fiec res in t canc unge kcep vali dif	d by sult the d is e of ptic id c fere nu	y g in ze in f 2 ons ope	gener gero d finit ³² – s are erati- ces a	ral- nera ire te, 1 1 t no on amo	pu al-j ctic not to (on t cong side	rpos ourp on, 1 :-a-r), an nab d nc g sou g sou	se no nu nl ole un	e reg sse r garo imbo IEE: ed, ti exce rces. rang	gis eg dle er E7 he pr	ter f ter f jiste ess (, or 754- ; pre tion	reg2 r re of th neg def eser occ	g3 ne o ati ine vat sur	un curr ve r d ir ion s. T to	sig en nui tva bi fhe 0,	gned th rou mber alid c t (bit e retu	32- indi , or oper : 4) rn v	-bi ing wrat of val	t fix g mo hen ion the	xed odd n th ex F dif	d-poir e. ne rou ccepti PSR	nt Indi on	led is	l
[Floating-point operation exceptions]	Unimple Invalid o Inexact o	ope	erati	ion (exce				-	on	(E)																	

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0 [V]	0 (Int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

TRNCF.D	W				Flo	ating-p	DOIN		onvert Do								
[Instruction format]	TRN	CF.DW	reg2, reg3	3													
[Operation]	reg3	← trunc	reg2 (dou	ble —	• word)												
[Format]	Form	at F:I															
[Opcode]				_									~~~~	~~			
	15 r r r r reg2	11 10 c 0 1	1 1 1 1	5	<u>4</u> 0 0 0	0 0	03	w w	w w reg3	27 2 w	. 0	23 0 0	22 21 1 0 type	1	0 0 sub-op		16 0
[Description]	the restores The r When range If inv regist follow	egister pa s the resu esult is r n the soun e of 2^{31} – alid open ter is set ws, accon Source is	on arithme ir specific lt in gene ounded in rce operar 1 to -2^{31} ation exc as an inva ding to di a positiv a negativ	ed by ral-pu the z nd is in , an II eptior llid op fferer e num	general irpose r ero dire finite of EEE754 as are n beration aces am iber or	-purperegister ection or not- 4-defin ot ena a and r hong set $+\infty$: 2 ⁻	ose er reg a-n hed blee oure 31 _	regi gard umb inva d, th excep ces. - 1 is	lless of oer, or v alid op ne prese ption o	g2 to the c when eratic ervati ccurs ed.	32-b surrent the r on ex on b	oit fixe nt roun counde ceptic it (bit e retur	ed-poin nding : ed resu on is do 4) of t n valu	nt fo moo lt is etec he l	ormat de. s outsi ted. FPSR	, and	t
[Floating-point operation exceptions] Inval	-	ed operati ion excep tion (I)		-	н (Е)											

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	8	Q-NaN	S-NaN
Operation result [exception]	A (Int	eger)	0 (Int	eger)	Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.

TRNCF.S	L									Flc	oatir	ng-p	ooir	nt C	on	vert	Sinę	gle 1	o Lo	ong, ro	ound	tow	/ard	zero	ว (S	ingl	e)
														(Со	nver	sion	to	fixed	d-point	forr	nat	(sin	gle p	orec	isio	n)
[Instruction format]		TRNC	F.S	L reg	2, re	g3																					
[Operation]		reg3 ←	- tr	unc re	g2 (s	single	e —	→ lo	ng-	wo	rd)																
[Format]		Format	t F:	Ι																							
[Opcode]	45			10			F	4				0	24				07	00	05	00		04	20				10
	15 r	rrr		10	1	1 1	5 1	4	0	0	0	-	31 w	w	v	v w		26 1		0 0	1	0		0		17 · 0	0
		reg2	-				-				0	-			re				1	tegory		pe		sub-			
[Description]			l-p eci sult the of 2 lid r is s, a our	urpose fied by t is rou source $2^{63} - 1$ operation	e regi y ger indec e ope to – tion e an in ing to posi	ster i neral- l in t erand 2^{63} , a except nvalition difference trive	regí pur he : is i an l otio d o `ere nur	2 to rpo zero infi IEE ons per ence mbe	o 64 se ro o di nite EE75 are ratio es an	-bit egi rec or 54-0 not on a mor	t fix ster tion no def en ind ng ∞ : 2	xed r re n, r t-a- ine abl no sou 2 ⁶³	eg3 eg3 -nu d i ed ex urc	oint ard mb nva , th cep es. 1 is	t f les per ali ne j pti	form ss of ; or d op pres on c	at, a the whe erva occu	and e cu en t tion atic urs.	sto nrren he r n ex on b Tho	res th nt rou cound ception it (bit e retu	e re ndin ed r on is 4) (rn v	ng 1 esu s de of t	t in moo lt is etec he l	the de. s out ted. FPS	reg tsid R	giste e th	er
[Floating-point operation exceptions]	Unimp Invalid Inexact	l op	oeratio	n exe	cepti			-	on (E)																

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	8	Q-NaN	S-NaN
Operation result [exception]	A (Int	teger)	0 (Int	eger)	Max Int [V]		–Max Int [V]	

Note 1.] indicates an exception that must occur.

TRNCF.S	U	L	•								FI	oati	ing-	poi	nt (-			•			ng, ro I-point							
[Instruction format]			TRN	NC	F.S	UL	re	g2	, re	g3																							
[Operation]		r	reg3	3 ←	- tr	unc	reg	g2	(sir	ngle	e —	→ u	nsig	gne	ed l	onį	g-'	W	ord	ł)													
[Format]		J	Forr	mat	t F:	I																											
[Opcode]	15				11	10					F	Α				~		24				0.	7 0	6 0	5	00			1 20	`		17	16
	15 r	r	r	r		10 1	1	1	1	1	5 1	4	0	0	0		_				w			62 . (0 0		0) 1		16 0
			reg2	2													Ī			re	eg3		1	С	at	egory	ty	/pe		su	ıb-op	,	
[Description]		t T T T T T T	gene the The Whe resu dete If in regis	eral reg res en f ilt i ecte ister ows Sc	l-pu sult the s o ed. lid r is s, a our	urpo er pa is r sou utsi ope set cco	ose air cour urce de rati as rdin s a	reg spe nd e op the ion an ng po	gist ecif ed i pera e rai inv to c siti	er i ied in t and nge cep vali diff	regi by he i l is of otio d o cere nur	2 to ge zer inf 2 ⁶ ns per nco	o ur ener o d init ⁴ – are ratio es a er o	nsig cal- ire ire ce, 1 1 t nc on umc outs	gne -pu ctic not to (on e anc on g side	ed (rpc on, -a-), a nat d n g sc e th	54 re nu n ole ou	b e r eg; un IE ed ex rco ra	bit f regi ard nbe EEF , th ccej es.	fix ist lle er, E7 ne pt	eed- eer 1 ess c or 1 54- pre ion	poin eg3 of th nega defi serv occ	nt f e c ativ ne vati urs - 1	orr orr ve v d ir on . T	na rer /al nva bi he	ng-po t, and nt rou lue, o alid o alid o t (bit t (bit e retu: or +a	d st ndi r w oper (4) rn v	ing vhe rati	es th g mc en th ion `the ue c	ne i ode ne r exc FF diff	resul counc cepti PSR fers a	lt ir ded ion as	n I is
[Floating-point operation exceptions]	Ι	Unii Inva Inex	alid	l op	erat	tior	n e	xce				-	on	(E))																	

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0 [V]	0 (Int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

TRNCF.S	U	W	/							FI	oat	ing-	poi	int (-				-									ero (e pre		gle) ion)
[Instruction format]		T	RN	ICF.S	SU	JW	reg	g2,	reg.	3																								
[Operation]		re	;g3	← t	trui	nc r	eg2	2 (s	ingl	e –	→ u	insi	gno	ed [•]	woi	rd	.)																	
[Format]		Fo	orn	nat F	F:I																													
[Opcode]																																		
[0]	15			11	11	0				5	4				0)	31					27	26	25	5	2	32	22	21	20			17	16
	r	r	r	r r	: 1	1 1	. 1	. 1	. 1	1	1	0	0) () 1		W	W	,	W	w	w	1	0	(0 () :	1	0	0	0	0	0	0
		re	eg2		1											I			re	eg3				Са	ate	gor	y .	typ	be		sut	b-op)	
[Description]		cc re Tl W re de If re	onte esul he :: Vhe esul etec f inv egis ollo	inst ents It in resu en the It is o cted. valid ster i Sou Sou	of gen lt i e so out d oj is s acco urce	ger nera s ro our tsid pera et a corc e is	nera al-p ounc ce c e th atio s an ding a p	ll-p purp ded oppe ne ra n e n in g to osit	urpoose in rand ang xce val odif tive	ose reg the d is e of ptic id c fere	reg gis zer in f 2 ^{:2} ons ope enc mb	gist ter 1 ro d fini ³² _ are rati	er f reg lire te, 1 e no on am	reg g3. ecti no to (ot e an ong	2 to on, t-a- D, a enat d no g so e th	re ni n ole ou	un eg un IE ed ex rc ra	sig ard nbe EEF l, th cce es. ang	ne er, E7 ne pt	ed 3 ess o or 54- pre ion	2-l of t neg de ser oc	bit he gat fin cu	fiz cutiv ned ations.	xed urre e nu inv on t Th o 0	-p ent un va oit ne	t ro nbe lid : (bi reti	unc r, c ope it 4 urn	din or v era	ng 1 wh atic of t alu	t, ai mo ien on e ihe ie d	nd s de. the exc FP	stor e rot ept: SR ers	und ion as	led is
[Floating-point operation exceptions]	In	iva	nple lid o act e	pe	rati	on	exc	ept			-	on	(E)																			

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	_ ∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0 [V]	0 (Int	eger)	Max U-Int [V]		0 [V]	

Note 1. [] indicates an exception that must occur.

TRNCF.S	W	ľ										Flc	atir	ng-p	oin	t C	on	vert	Sin	gle	to V	/ord,	ro	und	tov	varc	l zei	ro (S	Sing	le)
																	Сс	onve	rsio	n to	fixe	d-pc	int	forr	nat	(sir	ıgle	pre	cisio	on)
[Instruction format]		TF	RNC	CF.S	W re	eg2,	reg	3																						
[Operation]		reş	g3 ←	– tr	unc r	eg2	(sin	ıgle	\rightarrow	• w	ord	l)																		
[Format]		Fo	orma	t F:	I																									
[Opcode]																														
	15 r	r r	r		10 1 1	1	1	1	5 1	4 0	0	0	0		31 w	w	, ,	w v		Τ	0	0			21 0	20	0	0	17 0	16 0
		reg			<u> </u> 						-							g3			<u> </u>	atego						o-op		
[Description]		con gen Th WI ran If i reg	nter nera he re hen inva giste llow	nts of al-presult the of 2 alid er is ys, a	uction of gen urpos t is ro source 2^{31} _ opera set a accord ce is ce is	eral e rej und ce oj 1 to ttior s an ling a po	-pu gist ed i pera -2^{3} i ex inv to c	rpos er re n th und i ¹ , an cept valid diffe ve n	se i eg2 is i is i tion l oj eren	reg 3. zer nfi EE ns per nce	giste o di inite EE7 are catio es a	er re irec e or 54- not on a mo	eg2 tio no def t en and ng ∞ :	to n, r t-a- îne abl no sou 2 ³¹	32 ega -nu ed i led ex urco -	-bi ard umb nva , th acep es. 1 is	it f lle be: ali ne pti	ixed ss o r, or d o presion	d-po f th wh pera serv occ	oint e cu en utio atio urs.	for urre the n ez on t Th	ent roui roui kcep bit (l	, a ou nde otic oit tur	nd ndi ed r on i 4) (rn v	sto ng esu s d of t	res mo ilt i eteo the	the de. s ou cted FPS	res itsic I. SR	ult le t	
[Floating-point operation exceptions]]	Inv	valio	d op	nente peratio kcepti	on e	xce					on ((E)																	

[Operation result]

reg2 (A)	Normal	–Normal	+0	-0	+∞	8	Q-NaN	S-NaN
Operation result [exception]	A (Int	eger)	0 (Int	eger)	Max Int [V]		–Max Int [V]	

Note 1. [] indicates an exception that must occur.



Section 8 RESET

8.1 Status of Registers After Reset

If a reset signal is input by a method defined by the hardware specifications, the program registers and system registers are placed in the status shown by the value after reset of each register in **Section 3**, **REGISTER SET**, and program execution is started. Set the contents of each register to an appropriate value in the program.

The CPU executes a reset to start execution of a program from the reset address specified by **4.4**, **Exception Handler Address**.

Note that because the PSW.ID bit is set (1) immediately after a reset, conditional EI level exceptions will not be acknowledged. To acknowledge conditional EI level exceptions, clear (0) the PSW.ID bit.



APPENDIX A HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS

Certain system registers require the following procedures to resolve hazards when their values are updated by the LDSR instruction.

• Instruction fetching

When an instruction is to be fetched after updating a register covered by the description below, after executing the instruction to update the register, only allow the instruction fetch to start after execution of an EIRET, FERET, or SYNCI instruction.

- PSW.UM, MCFG0.SPID

When an instruction is to be fetched after updating a register covered by the description below, execute the instruction to update the register before allowing the instruction fetch to start.

- All registers related to ASID and MPU (register number : SR*, 5-7)
- Load/Store

When an instruction associated with Load/Store after updating the registers below, execute a SYNCP instruction after executing the instruction to update the registers before Load/Store instruction.

- ASID, MPU protection area setting register (Register number: SR*,6–7)
- Interrupt

Update the registers below when interrupt is inhibited. (PSW.ID = 1).

- PSW.EBV, EBASE, INTBP, ISPR, PMR, ICSR, INTCFG
- Coprocessor instruction

When a coprocessor instruction (floating-point operation instruction) is to be executed after updating the register below, execute instructions of EIRET, FERET, SYNCI or SYNCP after executing the instruction to update the registers and before executing a coprocessor instruction.

- PSW.CU0



APPENDIX B NUMBER OF INSTRUCTION EXECTUION CLOCKS

B.1 Numbers of Clock Cycles for Execution

Numbers of clock cycles for execution are given in this section. Since the G3MH has a pipe-lined architecture that differs from that of other CPUs, the various values given cannot be treated in a uniform manner. Moreover, the number of clock cycles required to execute an actual instruction may differ with the state of execution of the previous and next instruction.



B.2 Number of G3MH Instruction Execution Clocks

(1) Basic instruction

			Instruction	Number	r of Executio	on Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Load instruction	LD.B	disp16 [reg1] , reg2	4	1	1	3 ^{*1}
		disp23 [reg1], reg3	6	1	1	3 ^{*1}
	LD.BU	disp16 [reg1] , reg2	4	1	1	3 ^{*1}
		disp23 [reg1] , reg3	6	1	1	3 ^{*1}
	LD.H	disp16 [reg1] , reg2	4	1	1	3 ^{*1}
		disp23 [reg1] , reg3	6	1	1	3 ^{*1}
	LD.HU	disp16 [reg1] , reg2	4	1	1	3 ^{*1}
		disp23 [reg1] , reg3	6	1	1	3 ^{*1}
	LD.W	disp16 [reg1] , reg2	4	1	1	3 ^{*1}
		disp23 [reg1] , reg3	6	1	1	3 ^{*1}
	LD.DW	disp23 [reg1] , reg3	6	1	1	3 ^{*1}
ep relative	SLD.B	disp7 [ep] , reg2	2	1	1	3 ^{*1}
	SLD.BU	disp4 [ep] , reg2	2	1	1	3 ^{*1}
	SLD.H	disp8 [ep] , reg2	2	1	1	3 ^{*1}
	SLD.HU	disp5 [ep] , reg2	2	1	1	3 ^{*1}
	SLD.W	disp8 [ep] , reg2	2	1	1	3 ^{*1}
Store instrucrion	ST.B	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.H	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.W	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.DW	reg3, disp23 [reg1]	6	1	1	1
ep relative	SST.B	reg2, disp7 [ep]	2	1	1	1
	SST.H	reg2, disp8 [ep]	2	1	1	1
	SST.W	reg2, disp8 [ep]	2	1	1	1
Multiplication instruction	MUL	reg1, reg2, reg3	4	1	1	3
		imm9, reg2, reg3	4	1	1	3
	MULH	reg1, reg2	2	1	1	3
		imm5, reg2	2	1	1	3
	MULHI	imm16, reg1, reg2	4	1	1	3
	MULU	reg1, reg2, reg3	4	1	1	3
		imm9, reg2, reg3	4	1	1	3
Multiply-accumulate operation	MAC	reg1, reg2, reg3, reg4	4	2	2	4
	MACU	reg1, reg2, reg3, reg4	4	2	2	4



			Instruction	Number	of Execution	on Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Arithmetic instruction	ADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	MOV	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		imm32, reg1	6	1	1	1
	MOVEA	imm16, reg1, reg2	4	1	1	1
	MOVHI	imm16, reg1, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
Operation with	ADF	cccc, reg1, reg2, reg3	4	1	1	1
condition	SBF	cccc, reg1, reg2, reg3	4	1	1	1
Saturated operation	SATADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
	SATSUBR	reg1, reg2	2	1	1	1
Logical instruction	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
	XORI	imm16, reg1, reg2	4	1	1	1



			Instruction	Numbe	r of Executio	on Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Data operation	BINS	reg1, pos, width, reg2	4	1	1	1
instruction	BSH	reg2, reg3	4	1	1	1
	BSW	reg2, reg3	4	1	1	1
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
		cccc, imm5, reg2, reg3	4	1	1	1
	HSH	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	ROTL	imm5. reg2. reg3	4	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SASF	cccc. reg2	4	1	1	1
	SETF	cccc. reg2	4	1	1	1
	SHL	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SHR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1
Bit search instruction	SCH0L	reg2, reg3	4	1	1	1
	SCH0R	reg2, reg3	4	1	1	1
	SCH1L	reg2, reg3	4	1	1	1
	SCH1R	reg2, reg3	4	1	1	1
Division instruction	DIV	reg1, reg2, reg3	4	1	19	19
	DIVH	reg1, reg2	2	1	19	19
		reg1, reg2, reg3	4	1	19	19
	DIVHU	reg1, reg2, reg3	4	1	19	19
	DIVU	reg1, reg2, reg3	4	1	19	19
High-speed divide	DIVQ	reg1, reg2, reg3	4	1	N+3 ^{*2}	N+3 ^{*2}
operation	DIVQU	reg1, reg2, reg3	4	1	N+3 ^{*2}	N+3 ^{*2}

			Instruction	Number	of Executio	n Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Branch instructions	Bcond	disp9	2	2 to 6 ^{*3}	2 to 6 ^{*3}	2 to 6 ^{*3}
		disp9 (When Branch prediction is matched)	2	2 to 3 ^{*3}	2 to 3 ^{*3}	2 to 3 ^{*3}
		disp17	4	2 to 6 ^{*3}	2 to 6 ^{*3}	2 to 6 ^{*3}
	JARL	disp22, reg2	4	2 to 3 ^{*3}	2 to 3 ^{*3}	2 to 3 ^{*3}
		disp32, reg1	6	2 to 3 ^{*3}	2 to 3 ^{*3}	2 to 3 ^{*3}
		[reg1], reg3	4	2 to 6 ^{*3}	2 to 6 ^{*3}	2 to 6 ^{*3}
	JMP	[reg1]	2	2 to 6 ^{*3}	2 to 6 ^{*3}	2 to 6 ^{*3}
		disp32 [reg1]	6	2 to 7 ^{*3}	2 to 7 ^{*3}	2 to 7 ^{*3}
	JR	disp22	4	2 to 3 ^{*3}	2 to 3 ^{*3}	2 to 3 ^{*3}
		disp32	6	2 to 3 ^{*3}	2 to 3 ^{*3}	2 to 3 ^{*3}
oop instruction	LOOP	reg1, disp16	4	2 to 6 ^{*3}	2 to 6 ^{*3}	2 to 6 ^{*3}
Bit manipulation	CLR1	bit#3, disp16 [reg1]	4	1	1	4 ^{*4}
nstruction		reg2, [reg1]	4	1	1	4 ^{*4}
	NOT1	bit#3, disp16 [reg1]	4	1	1	4 ^{*4}
		reg2, [reg1]	4	1	1	4 ^{*4}
	SET1	bit#3, disp16 [reg1]	4	1	1	4 ^{*4}
		reg2, [reg1]	4	1	1	4 ^{*4}
	TST1	bit#3, disp16 [reg1]	4	1	1	4 ^{*4}
		reg2, [reg1]	4	1	1	4 ^{*4}
Special instruction						
Table reference branch	SWITCH	reg1	2	11 to 18 ^{*3}	11 to 18 ^{*3}	11 to 18 [*]
Sub routine call	CALLT	imm6	2	17	17	17
	CTRET	-	4	8	8	8
System call exception	SYSCALL	vector8	4	17	17	17
Software exception	FETRAP	vector4	2	8	8	8
	TRAP	vector5	4	8	8	8
Return from	EIRET	_	4	8	8	8
exception processing	FERET	_	4	8	8	8
El level interrupt	DI	_	4	3	3	3
	EI	_	4	3	3	3
Restoration from & storage on stack	DISPOSE	imm5, list12	4	N+1 ^{*5}	N+2 ^{*5}	N+1 ^{*5}
		imm5, list12, [reg1]	4	N+3 to N+8 ^{*5}	N+4 to N+8 ^{*5}	N+3 to N+8 ^{*5}
	PREPARE	list12, imm5	4	N+1 ^{*5}	N+2 ^{*5}	N+1 ^{*5}
		list12, imm5, sp	4	N+2 ^{*5}	N+3 ^{*5}	N+2 ^{*5}
		list12, imm5, imm16	6	N+2 ^{*5}	N+3 ^{*5}	N+2 ^{*5}
		list12, imm5, imm16<<16	6	N+2 ^{*5}	N+3 ^{*5}	N+2 ^{*5}
		list12, imm5, imm32	8	N+2 ^{*5}	N+3 ^{*5}	N+2 ^{*5}
	POPSP	rh-rt	4	N+1 ^{*6}	N+2 ^{*6}	N+1 ^{*6}
	PUSHSP	rh-rt	4	N+1 ^{*6}	N+2 ^{*6}	N+1 ^{*6}



			Instruction	Numbe	r of Executio	on Clocks
pes of Instructions	Mnemonics	nemonics Operand	Length (Number of Bytes)	issue	repeat	latency
System register	LDSR	reg2, regID, selID	4	3 ^{*7}	3 ^{*7}	3
operation	STSR	regID, reg2, seIID	4	1	1	3
Exclusive control	CAXI	[reg1], reg2, reg3	4	1	1	8 ^{*4}
	LDL.W	[reg1], reg3	4	1	1	3 ^{*1}
	STC.W	reg3, [reg1]	4	1	1	6 ^{*4}
Stop	HALT	_	4	1	1	1
	SNOOZE	_	4	*8	*8	*8
Synchronization	SYNCE	_	2	1	1	1
	SYNCI	_	2	*9	*9	*9
	SYNCM	_	2	*10	*10	*10
	SYNCP	_	2	*11	*11	*11
Others	NOP	_	2	1	1	1
	RIE	1_	4	8	8	8

(2) Cache instruction

			Instruction	Number	Number of Execution Clocks		
Type of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency	
Cache operation instruction	CACHE	cacheop, [reg1]	4	1 ^{*12}	1 ^{*12}	1 ^{*12}	
Pre-fetch instruction	PREF	prefop, [reg1]	4	1 ^{*12}	1 ^{*12}	1 ^{*12}	



(3)	Floating-point o	peration	instruction —	single	precision —
-----	------------------	----------	---------------	--------	-------------

			Instruction	Numb	er of Execut	ion Clocks
Type of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Floating-point arithmetic	ABSF.S	reg2, reg3	4	1	1	4 ^{*13}
operation	ADDF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
	NEGF.S	reg2, reg3	4	1	1	4 ^{*13}
	SUBF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
Floating-point	MULF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
multiplication						*10
Multiply-accumulate/ subtract operation	FMAF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
	FMSF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
	FNMAF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
	FNMSF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
Floating-point subtraction	DIVF.S	reg1, reg2, reg3	4	8 ^{*14}	8	11 ^{*13}
Square root of a	RECIPF.S	reg2, reg3	4	8 ^{*14}	8	11 ^{*13}
Floatingpoint value /Reciprocal	RSQRTF.S	reg2, reg3	4	21 ^{*14}	21	24 ^{*13}
	SQRTF.S	reg2, reg3	4	14 ^{*14}	14	17 ^{*13}
Conversion between	CVTF.HS	reg2, reg3	4	1	1	4 ^{*13}
floatingpoint formats/ Conversion between	CVTF.LS	reg2, reg3	4	1	1	4 ^{*13}
fixedpoint and flating	CVTF.SH	reg2, reg3	4	1	1	4 ^{*13}
point formats	CVTF.SL	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.SUL	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.SUW	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.SW	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.ULS	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.UWS	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.WS	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.SL	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.SUL	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.SUW	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.SW	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.SL	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.SUL	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.SUW	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.SW	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.SL	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.SUL	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.SUW	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.SW	reg2, reg3	4	1	1	4 ^{*13}

			Instruction	Number of Execution (on Clocks
Type of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Floating-point comparison	CMPF.S	cond, reg1, reg2, cc	4	1	1	1
Transfer with conditions	CMOVF.S	cc, reg1, reg2, reg3	4	1	1	4 ^{*13}
Bit transfer with conditions	TRFSR	сс	4	1	1	5
Floating-point maximum/	MAXF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}
minimum values	MINF.S	reg1, reg2, reg3	4	1	1	4 ^{*13}

(4) Floating-point operation instruction — double precision —

			Instruction	Number	Number of Execution Clocks		
Type of Instructions	Mnemonics	(1	Length (Number of Bytes)	issue	repeat	latency	
Floating-point arithmetic	ABSF.D	reg2, reg3	4	1	1	4 ^{*13}	
operation	ADDF.D	reg1, reg2, reg3	4	1	1	4 ^{*13}	
	NEGF.D	reg2, reg3	4	1	1	4 ^{*13}	
	SUBF.D	reg1, reg2, reg3	4	1	1	4 ^{*13}	
Floating-point multiplication	MULF.D	reg1, reg2, reg3	4	4	4	7 ^{*13}	
Floating-point division	DIVF.D	reg1, reg2, reg3	4	16 ^{*14}	16	19 ^{*13}	
Square root of a	RECIPF.D	reg2, reg3	4	16 ^{*14}	16	19 ^{*13}	
Floatingpoint value / Reciprocal	RSQRTF.D	reg2, reg3	4	45 ^{*14}	45	48 ^{*13}	
	SQRTF.D	reg2, reg3	4	30 ^{*14}	30	33 ^{*13}	



			Instruction	Numbe	er of Executi	ion Clocks
Type of Instructions	Mnemonics Operand	Length (Number of Bytes)	issue	repeat	latency	
Conversion between	CVTF.DL	reg2, reg3	4	1	1	4 ^{*13}
floatingpoint formats/ Conversion between	CVTF.DS	reg2, reg3	4	1	1	4 ^{*13}
fixedpoint and floating	CVTF.DUL	reg2, reg3	4	1	1	4 ^{*13}
point formats	CVTF.DUW	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.DW	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.LD	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.SD	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.ULD	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.UWD	reg2, reg3	4	1	1	4 ^{*13}
	CVTF.WD	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.DL	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.DUL	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.DUW	reg2, reg3	4	1	1	4 ^{*13}
	CEILF.DW	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.DL	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.DUL	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.DUW	reg2, reg3	4	1	1	4 ^{*13}
	FLOORF.DW	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.DL	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.DUL	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.DUW	reg2, reg3	4	1	1	4 ^{*13}
	TRNCF.DW	reg2, reg3	4	1	1	4 ^{*13}
Floating-point comparison	CMPF.D	cond, reg1, reg2, cc	4	1	1	1
Transfer with conditions	CMOVF.D	cc, reg1, reg2, reg3	4	1	1	4 ^{*13}
Floating-point maximum/	MAXF.D	reg1, reg2, reg3	4	1	1	4 ^{*13}
minimum values	MINF.D	reg1, reg2, reg3	4	1	1	4 ^{*13}

Note 1. When no waiting is required (3 + number of wait states for read access).

Note 2. N = int(((number of valid bits in absolute value of dividend) – (number of valid bits in absolute value of divisor)) ÷ 2) + 1

N becomes 1 for the result of N < 1. Division by 0 leads to N being 0. The range of N is from 0 to 16.

Note 3. Executing an instruction to rewrite the contents of the PSW register immediately beforehand does not affect the number of clock cycles for execution. Even if an immediately preceding instruction has rewritten the contents of the PSW register, parallel execution is possible.

Note 4. When no waiting is required (4 + number of wait states for read access).

Note 5. "N" depends on the total number of registers specified by list12, but not on the register numbers.

Since up to two registers are handled per clock cycle, if no waiting is required, the values are as shown below.

PREPARE: Minimum value is 1, maximum value is 6 (if the EP register is updated, add 1 clock cycle).
DISPOSE: Minimum value is 1, maximum value is 6 (if accompanied by JMP, add 2 clock cycles).
"N" depends on the total number of registers specified by (rh-rt).

Note 6. "N" depends on the total number of registers specified by (rh-rt). Since up to two registers are handled per clock cycle, if no waiting is required, the values are as shown below.

PUSHSH: Minimum value is 1, maximum value is 16.

POPSP: Minimum value is 1, maximum value is 16.

Note 7. When accessing the system register to control the operation of PSW or so, stop issuing the subsequent

RENESAS

instructions.

If not, perform the operation with issue = 1.

- Note 8. The number of execution clocks for the SNOOZE instruction is specified by the hardware specifications. For details, see the hardware manual of the corresponding product.
- Note 9. Wait for the hazard resolution for the instruction.
- Note 10. Perform the synchronization of memory access.
- Note 11. Wait for the synchronization of the pipeline.
- Note 12. Though the execution of the instruction is completed, the completion of the internal processing depends on the state of the instruction fetch unit.
- Note 13. "latency" might be added by 1 depending on the subsequent instruction. For details, see Note 1.
- Note 14. "issue" is set to 1 for the instructions other than the one with the floating-point division (DIVF, RECIPF, RSQRTF, or SQRTF)

Note 1. Example of execution clocks

Symbol	Description
issue	When the other instruction is executed immediately after the execution of the current instruction
repeat	When the same instruction is repeated immediately after the execution of the current instruction
latency	When the following instruction uses the result of the current instruction*1
,	the following case "latency" is added by 1

Note 1. In the following case, "latency" is added by 1.

• When the preceding instruction is the floating-point operation instruction, the instruction other than the floating-point operation one follows.

• When the subsequent instruction is the floating-point operation instruction, the instruction other than the load, store, bit-manipulation, special (memory access), or floating-point operation one precedes.



APPENDIX C REGISTER INDEX

А

ASID C	49
CDBCR CTBP CTPC CTPSW	48 47
E	
EBASE EIIC EIPC EIPSW EIWR	46 40 41

F

FEIC	47
FEPC	42
FEPSW	43
FEWR	49
FPCC	67
FPCFG	68
FPEPC	66
FPSR	
FPST	66

Н

HTCFG0	50
1	

I

Μ

MCA	73
MCC	73
MCFG0	
MCR	74
MCS	73
MCTL	57
MEA	50
MEI	51
MPATn	76

MPBRGN MPLAn MPM MPRC MPTRGN MPUAn P	72 75 71 72 72 75
PC PID PMR PSW R	38 55 60 44
RBASE	53
SCBP	56

SCCFG 56



APPENDIX D INSTRUCTION INDEX

А

ABSF.D	315
ABSF.S	316
ADD	154
ADDF.D	317
ADDF.S	319
ADDI	155
ADF	156
AND	157
ANDI	158

В

С

CACHE CALLT CAXI CEILF.DL CEILF.DUL CEILF.DW CEILF.DW CEILF.SU CEILF.SU CEILF.SU CEILF.SUW CEILF.SW CLL CLR1 CMOVF.D CMOVF.D CMOVF.S CMP CMPF.D CMPF.S CTRET CVTF.DL CVTF.DL CVTF.DL	303 165 167 321 322 323 324 325 326 327 328 169 170 172 329 330 174 331 335 175 339
-	
CVTF.DS	340
CVTF.DUL	341
CVTF.DUW	342
CVTF.DW	343
CVTF.HS	344
CVTF.LD	345
CVTF.LS	346
CVTF.SD	347
CVTF.SH	349
CVTF.SL	348
CVTF.SUL	350
CVTF.SUW	351
CVTF.SW	352
CVTF.ULD	353
CVTF.ULS	354

CVTF.UWD CVTF.UWS CVTF.WD CVTF.WS D	355 356 357 358
DI DISPOSE DIV DIVF.D DIVF.S DIVH DIVH DIVHU DIVQ DIVQ DIVQ DIVU E	176 177 359 361 180 182 183 185 185
EI EIRET F	188 189
FERET FETRAP. FLOORF.DL. FLOORF.DUL. FLOORF.DUW. FLOORF.DW. FLOORF.SL. FLOORF.SUL. FLOORF.SUU. FLOORF.SW. FLOORF.SW. FMAF.S. FMAF.S. FNMAF.S. FNMAF.S.	190 191 363 364 365 366 367 368 369 370 371 373 375 377
H HALT	193
HSH HSW	195 196
J	
JARL JMP JR	
LD.B	201

LD.B	201
LD.BU	202
LD.DW	204
LD.H	205
LD.HU	207

LD.W	. 209
LDL.W	. 211
LDSR	. 212
LOOP	. 213

М

	045
MAC	
MACU	216
MAXF.D	379
MAXF.S	381
MINF.D	382
MINF.S	384
MOV	217
MOVEA	219
MOVHI	220
MUL	221
MULF.D	386
MULF.S	387
MULH	222
MULHI	224
MULU	225

Ν

NEGF.D NEGF.S NOP NOT NOT1	389 226 227
0	
OR ORI P	
POPSP PREF PREPARE PUSHSP	305 234
R	
RECIPF.D	
RECIPF.S RIE ROTL RSQRTF.D. RSQRTF.S	240 392
RIE ROTL RSQRTF.D	240 392

R01US0143EJ0130 Rev.1.30 Dec 22, 2016



SATSUBR	249
SBF	250
SCH0L	251
SCHOR	252
SCH1L	252
SCH1R	253
SET1	254 255
SETF	257
SHL	259
SHR	261
SLD.B	263
SLD.BU	264
SLD.H	265
SLD.HU	266
SLD.W	267
SNOOZE	268
SQRTF.D	394
SQRTF.S	395
SST.B	270
SST.H	271
SST.W	272
ST.B	273
ST.DW	274
ST.H	275
ST.W	277
STC.W	279
STSR	281
SUB	282
SUBF.D	396
SUBF.S	398
	283
SUBR	203 284
SWITCH	
SXB	285
SXH	
SYNCE	287
SYNCI	288
SYNCM	289
SYNCP	290
SYSCALL	291
т	

TRAP	293
TRFSR	399
TRNCF.DL	400
TRNCF.DUL	401
TRNCF.DUW	402
TRNCF.DW	403
TRNCF.SL	404
TRNCF.SUL	405
TRNCF.SUW	406
TRNCF.SW	407
TST	295
TST1	296

XOR XORI Z	
Z ZXB ZXH	

Х



REVISION HISTORY

RH850G3MH User's Manual: Software

Page	Description	Classification
Section 1 O	/ERVIEW	
9	Table 1.2 Changes from the RH850G3M: Changes for Exceptions/interrupts and Cache, modified	(a), (b)
Section 3 RE	GISTER SET	•
52	Table 3.21 Instructions Causing Exceptions and Values of MEI Register (2/2): RW of CACHE, modified	(C)
64	Table 3.36 FPSR Register Contents: Note 2. added	(b)
82	3.6.1, (7) ICERR - Instruction cache error: Description added	(b)
Section 5 ME	MORY MANAGEMENT	
111	5.1.6, (2) Sample code: Modified (be \rightarrow bnz)	(a)
116	5.2.7 Memory Protection for CACHE and PREF Instructions: Description modified	(C)
CHAPTER 6	COPROCESSOR	
127	6.1.2, (3) Expanded floating-point format: Description modified	(a)
132	6.1.6, (1) Inexact exception (I): Note 1. added	(b)
139	6.1.9 Flush to Nearest: Description modified	(b)
Section 7 IN	STRUCTION	•
359	<floating-point instruction=""> DIVF.D: [Description] modified</floating-point>	(b)
361	<ploating-point instruction=""> DIVF.S: [Description] modified</ploating-point>	(b)

Remark: The classification in the table above means as follows.

(a): Error correction (b): Specifications added or changed (c): descriptions or notes added or changed



RH850G3MH	User's Manual: Software			
Publication Date:	Rev.1.00 Rev.1.10 Rev.1.20 Rev.1.30	Mar 05, 2015 Nov 30, 2015 Jun 30, 2016 Dec 22, 2016		
Published by:	Renesas Electronics Corporation			

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renease Electronics Charge CA 95050-2549, U.S.A. Tel: +1-408-588-6000, Fax: +1-408-588-6130 Renease Electronics Canada Limited 9251 Yonge Street, Suite S309 Richmond Hill, Ontario Canada L4C 9T3 Tel: +1-405-237-2004 Renease Electronics Curope Limited Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K Tel: +44-1625-8551-00, Fax: +44-702-858-900 Renease Electronics Europe ImbH Arcadiastrasse 10, 40472 Düsseldorf, Germany Tel: +49-125-8551-00, Fax: +49-211-6503-1327 Renease Electronics (China) Co., Ltd. Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China Tel: +86-10-8235-1155, Fax: +86-10-8235-7679 Renease Electronics Hong Maphal Co., Ltd. Room 3709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Shanghai, P. R. China 200333 Tel: +86-11-2226-0888, Fax: +862-10-8235-7679 Renease Electronics Hong Kong Limited Unit 8001-11611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong Tel: +852-226-28688, Fax: +862-24226-0899 Renease Electronics Shangapore Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong Tel: +852-245-6888, Fax: +852-2886-9022 Renease Electronics Taiwan Co., Ltd. 805, Fu Shing North Road, Taipei 10543, Taiwan Tel: +852-245-6898, Fax: +856-213:000 Renease Electronics Magnore Place, Utd. 805, Fu Shing North Road, Taipei 10543, Taiwan Tel: +856-24175-9900, Fax: +868 2-8175-9670 Renease Electronics Malaysia Sdn.Bhd. 901 1207, Flack B, Manara Amoorp, Amorpor Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia Tel: +65-3755-5930, Fax: +60-37955-98510 Renease Electronics Malaysia Sdn.Bhd. 1011 1207, Flack B, Manara Amoorp, Amorpor Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia Tel: +69-3-7955-9390, Fax: +60-37965-9510 Renease Electronics Malaysia Sdn.Bhd. 1011 1207, Flack B, Manara K, Hou J, 13569, Indiarangar, Bangalore, India Tel: +69-3-7955-9390, Fax: +60-37965-9511

RH850G3MH

