

# 32 R

## RH850G4MH

## 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.2.20 Dec. 2023

## Notice

- 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
- Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
- 3. 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 shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
- 5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
- Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended 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; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

- 7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABLITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
- 8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
- 12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
- 13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
- 14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.
- (Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
- (Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## **Corporate Headquarters**

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan www.renesas.com

## Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: <a href="http://www.renesas.com/contact/">www.renesas.com/contact/</a>

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

#### 1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can 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 must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

#### 2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power is supplied until the power reaches the level at which reseting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. 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. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused 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 power supply 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.

5. 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  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.) due to noise, for example, 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  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.).

6. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

7. 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.

## **Table of Contents**

Section 1		Overvi	iew	11
1.1	Purp	ose of	This User's Manual	11
1.2	-		the RH850G4MH	
	1.2.1		processing Environment	
Section 2	2	Instrue	ction	
2.1	Орсо	odes ai	nd Instruction Formats	
	- 2.1.1	CPU	Instructions	13
	2.1.2	Copro	ocessor Instructions	
	2.1.3	•	rved Instructions	
2.2			uctions	
	2.2.1		view of Basic Instructions	
			al Operations	
	2.2.2	•	•	
	2.2.3		Instruction Set	
		2.3.1 2.3.2	ADD ADDI	
		2.3.3	ADE	
		2.3.4	AND	
		2.3.5	ANDI	-
	2.2	2.3.6	Bcond	
	2.2	2.3.7	BINS	
	2.2	2.3.8	BSH	41
	2.2	2.3.9	BSW	42
	2.2	2.3.10	CALLT	43
	2.2	2.3.11	CAXI	
		2.3.12	CLIP.B	
		2.3.13	CLIP.BU	
		2.3.14	CLIP.H	
		2.3.15	CLIP.HU	
		2.3.16 2.3.17	CLL	-
		2.3.17	CMOV	
		2.3.19	CMP	
		2.3.20	CTRET	
		2.3.21	DI	
	2.2	2.3.22	DISPOSE	59
	2.2	2.3.23	DIV	62
	2.2	2.3.24	DIVH	64
	2.2	2.3.25	DIVHU	66
	2.2	2.3.26	DIVQ	
	2.2	2.3.27	DIVQU	
		2.3.28	DIVU	
	2.2	2.3.29	El	74

2.2.3.30	EIRET	75
2.2.3.31	FERET	76
2.2.3.32	FETRAP	77
2.2.3.33	HALT	79
2.2.3.34	HSH	81
2.2.3.35	HSW	82
2.2.3.36	JARL	83
2.2.3.37	JMP	85
2.2.3.38	JR	86
2.2.3.39	LD.B	
2.2.3.40	LD.BU	91
2.2.3.41	LD.DW	93
2.2.3.42	LD.H	95
2.2.3.43	LD.HU	98
2.2.3.44	LD.W	101
2.2.3.45	LDL.BU	104
2.2.3.46	LDL.HU	106
2.2.3.47	LDL.W	108
2.2.3.48	LDM.MP	110
2.2.3.49	LDSR	112
2.2.3.50	LOOP	114
2.2.3.51	MAC	116
2.2.3.52	MACU	117
2.2.3.53	MOV	118
2.2.3.54	MOVEA	120
2.2.3.55	MOVHI	121
2.2.3.56	MUL	122
2.2.3.57	MULH	124
2.2.3.58	MULHI	126
2.2.3.59	MULU	127
2.2.3.60	NOP	129
2.2.3.61	NOT	130
2.2.3.62	NOT1	131
2.2.3.63	OR	133
2.2.3.64	ORI	134
2.2.3.65	POPSP	135
2.2.3.66	PREPARE	137
2.2.3.67	PUSHSP	140
2.2.3.68	RESBANK	142
2.2.3.69	RIE	144
2.2.3.70	ROTL	146
2.2.3.71	SAR	147
2.2.3.72	SASF	149
2.2.3.73	SATADD	151
2.2.3.74	SATSUB	153
2.2.3.75	SATSUBI	155
2.2.3.76	SATSUBR	156
2.2.3.77	SBF	157
2.2.3.78	SCH0L	159
2.2.3.79	SCH0R	160

2.2.3.80	SCH1L	161
2.2.3.81	SCH1R	162
2.2.3.82	SET1	163
2.2.3.83	SETF	165
2.2.3.84	SHL	167
2.2.3.85	SHR	169
2.2.3.86	SLD.B	171
2.2.3.87	SLD.BU	172
2.2.3.88	SLD.H	173
2.2.3.89	SLD.HU	174
2.2.3.90	SLD.W	175
2.2.3.91	SNOOZE	176
2.2.3.92	SST.B	178
2.2.3.93	SST.H	179
2.2.3.94	SST.W	180
2.2.3.95	ST.B	
2.2.3.96	ST.DW	-
2.2.3.97	ST.H	
2.2.3.98	ST.W	
2.2.3.99	STC B	-
2.2.3.100	STC.H	
2.2.3.101	STC.W	-
2.2.3.102	STM.MP	
2.2.3.102	STSR	
2.2.3.103	SUB	
2.2.3.104	SUBR	-
2.2.3.105	SUBR	
2.2.3.100	SXB	
2.2.3.107	SXH	
2.2.3.108	SYNCE.	
2.2.3.109	SYNCI	
	SYNCM	
2.2.0.111		
2.2.3.112	SYNCP	
2.2.3.113	SYSCALL	
2.2.3.114		
2.2.3.115	TST	
2.2.3.116	TST1	-
2.2.3.117	XOR	-
2.2.3.118	XORI	
2.2.3.119	ZXB	
2.2.3.120	ZXH	221
Cache Instr	uctions	222
2.3.1 Overv	iew of Cache Instructions	222
2.3.2 Cache	Instruction Set	222
2.3.2.1	CACHE	
2.3.2.1	PREF	-
-	int Instructions	
-		
2.4.1 Instruc	ction Formats	227
2.4.2 Overv	iew of Floating-Point Instructions	228

2.3

2.4

2.4.3	Condit	ions for Comparison Instructions	231
2.4.4	Floatin	g-Point Instruction Set	233
2.4.	4.1	ABSF.D	235
2.4.	4.2	ABSF.S	236
2.4.	4.3	ADDF.D	237
2.4.	4.4	ADDF.S	239
2.4.	4.5	CEILF.DL	241
2.4.	4.6	CEILF.DUL	243
2.4.	4.7	CEILF.DUW	245
2.4.	4.8	CEILF.DW	247
2.4.	4.9	CEILF.SL	249
2.4.	4.10	CEILF.SUL	251
2.4.	4.11	CEILF.SUW	253
2.4.	4.12	CEILF.SW	255
2.4.	4.13	CMOVF.D	257
2.4.	4.14	CMOVF.S	258
2.4.	4.15	CMPF.D	259
2.4.	4.16	CMPF.S	263
2.4.	4.17	CVTF.DL	267
2.4.	4.18	CVTF.DS	269
2.4.	4.19	CVTF.DUL	270
2.4.	4.20	CVTF.DUW	272
2.4.	4.21	CVTF.DW	274
2.4.	4.22	CVTF.HS	276
2.4.	4.23	CVTF.LD	277
2.4.	4.24	CVTF.LS	278
2.4.	4.25	CVTF.SD	279
2.4.	4.26	CVTF.SL	280
2.4.	4.27	CVTF.SH	282
2.4.	4.28	CVTF.SUL	283
2.4.	4.29	CVTF.SUW	285
2.4.	4.30	CVTF.SW	287
2.4.	4.31	CVTF.ULD	289
2.4.	4.32	CVTF.ULS	290
2.4.	4.33	CVTF.UWD	291
2.4.	4.34	CVTF.UWS	292
2.4.	4.35	CVTF.WD	293
2.4.	4.36	CVTF.WS	294
2.4.	4.37	DIVF.D	295
2.4.	4.38	DIVF.S	297
2.4.	4.39	FLOORF.DL	299
2.4.	4.40	FLOORF.DUL	301
2.4.	4.41	FLOORF.DUW	303
2.4.	4.42	FLOORF.DW	305
2.4.	4.43	FLOORF.SL	307
2.4.	4.44	FLOORF.SUL	309
2.4.	4.45	FLOORF.SUW	311
2.4.	4.46	FLOORF.SW	313
2.4.	4.47	FMAF.S	315

2.4.	.4.48	FMSF.S	317
2.4.	.4.49	FNMAF.S	319
2.4.	.4.50	FNMSF.S	321
2.4.	.4.51	MAXF.D	
2.4.	.4.52	MAXF.S	325
2.4.	.4.53	MINF.D	
2.4.	.4.54	MINF.S	329
2.4.	.4.55	MULF.D	331
2.4.	.4.56	MULF.S	
2.4.	.4.57	NEGF.D	
2.4.	.4.58	NEGF.S	
2.4.	.4.59	RECIPF.D	
2.4.	.4.60	RECIPF.S	
2.4.	.4.61	ROUNDF.DL	340
2.4.	.4.62	ROUNDF.DUL	342
2.4.	.4.63	ROUNDF.DUW	344
2.4.	.4.64	ROUNDF.DW	346
2.4.	.4.65	ROUNDF.SL	
2.4.	.4.66	ROUNDF.SUL	350
2.4.	.4.67	ROUNDF.SUW	352
2.4.	.4.68	ROUNDF.SW	354
2.4.	.4.69	RSQRTF.D	356
2.4.	.4.70	RSQRTF.S	358
2.4.	.4.71	SQRTF.D	359
2.4.	.4.72	SQRTF.S	
2.4.	.4.73	SUBF.D	361
2.4.	.4.74	SUBF.S	
2.4.	.4.75	TRFSR	365
2.4.	.4.76	TRNCF.DL	
2.4.	.4.77	TRNCF.DUL	
2.4.	.4.78	TRNCF.DUW	
2.4.	.4.79	TRNCF.DW	372
2.4.	.4.80	TRNCF.SL	374
2.4.	.4.81	TRNCF.SUL	376
2.4.	.4.82	TRNCF.SUW	378
2.4.	.4.83	TRNCF.SW	
Exten	nded Fl	oating-point Instructions	
2.5.1		ction Format	
2.5.2		ded Floating-point Instruction Set	
-			
2.5.3		ew of the Extended Floating-point Vector Manipulation Instructions	
2.5.4		ew of the Extended Floating-point Vector Arithmetic Instructions	
2.5.		ABSF.S4	
2.5.		ADDF.S4	
2.5.		ADDRF.S4	
2.5.		ADDSUBF.S4	
2.5.		ADDSUBNF.S4	
2.5.	-	ADDSUBNXF.S4	
2.5.		ADDSUBXF.S4	
2.5.	.4.8	ADDXF.S4	403

2.5

2.5.4.9	CEILF.SUW4	405
2.5.4.10	CEILF.SW4	407
2.5.4.11	CMOVF.W4	409
2.5.4.12	CMPF.S4	410
2.5.4.13	CVTF.HS4	414
2.5.4.14	CVTF.SH4	416
2.5.4.15	CVTF.SUW4	418
2.5.4.16	CVTF.SW4	420
2.5.4.17	CVTF.UWS4	422
2.5.4.18	CVTF.WS4	423
2.5.4.19	DIVF.S4	424
2.5.4.20	FLOORF.SUW4	426
2.5.4.21	FLOORF.SW4	428
2.5.4.22	FLPV.S4	430
2.5.4.23	FMAF.S4	432
2.5.4.24	FMSF.S4	434
2.5.4.25	FNMAF.S4	436
2.5.4.26	FNMSF.S4	438
2.5.4.27	LDV.DW	440
2.5.4.28	LDV.QW	442
2.5.4.29	LDV.W	443
2.5.4.30	LDVZ.H4	445
2.5.4.31	MAXF.S4	447
2.5.4.32	MAXRF.S4	449
2.5.4.33	MINF.S4	451
2.5.4.34	MINRF.S4	453
2.5.4.35	MOVV.W4	455
2.5.4.36	MULF.S4	456
2.5.4.37	MULRF.S4	458
2.5.4.38	MULXF.S4	460
2.5.4.39	NEGF.S4	462
2.5.4.40	RECIPF.S4	463
2.5.4.41	ROUNDF.SUW4	465
2.5.4.42	ROUNDF.SW4	467
2.5.4.43	RSQRTF.S4	469
2.5.4.44	SHFLV.W4	471
2.5.4.45	SQRTF.S4	472
2.5.4.46	STV.DW	474
2.5.4.47	STV.QW	476
2.5.4.48	STV.W	477
2.5.4.49	STVZ.H4	479
2.5.4.50	SUBADDF.S4	480
2.5.4.51	SUBADDNF.S4	482
2.5.4.52	SUBADDNXF.S4	484
2.5.4.53	SUBADDXF.S4	486
2.5.4.54	SUBF.S4	488
2.5.4.55	SUBRF.S4	490
2.5.4.56	SUBXF.S4	492
2.5.4.57	TRFSRV.W4	494
2.5.4.58	TRNCF.SUW4	495

	2.5.4.59	TRNCF.SW4	497
Appendi	x A Numbe	r of Instruction Execution Clocks	. 499
A.1	Numbers of	Clock Cycles for Execution	. 499
A.2	Number of C	64MH Instruction Execution Clocks	. 500

## Section 1 Overview

## 1.1 Purpose of This User's Manual

This user's manual is intended to describe the details of instructions available in the RH850G4MH.

There are some variations of RH850G4MH core. Those can be identified by PID value.

If there is a difference between variations, PID value is shown in this document. Other contents are common for all variations.

The supported RH850G4MH cores are described below.

RH850G4MH Core name	PID[31:24] Bit Value
RH850G4MH	06 <sub>H</sub>
RH850G4MH2	07 <sub>H</sub>

Note: For details of the RH850G4MH architecture, refer to the hardware manual of the product used.

## 1.2 Features of the RH850G4MH

The RH850G4MH has an architecture that is backward compatible with the 32-bit RISC RH850 Series microcontroller at the instruction set level. It provides basic functionalities for multi-core systems including the exclusive control among multiple cores.

**Table 1.1** shows the features of the RH850G4MH.

Table 1.1 Features of the RH850G4MH

Item	Features
CPU	<ul> <li>High performance 32-bit architecture for embedded control</li> </ul>
	32-bit internal data bus
	<ul> <li>Thirty-two 32-bit general-purpose registers</li> </ul>
	<ul> <li>RISC type instruction set (backward compatible with V850, V850E1, V850E2, RH850G3M, and RH850G3MH)</li> </ul>
	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
	Address space identifier SPID: 5 bits
Coprocessor	<ul> <li>A floating point operation coprocessor (FPU) can be installed.</li> </ul>
	Supports single precision (32-bit) and double precision (64-bit)
	Supports IEEE754-compliant data types and exceptions
	Rounding modes: Nearest, 0 direction, $+\infty$ direction, and $-\infty$ direction
	Handling on non-normalized numbers: Truncated to 0, or an exception is reported to comply with IEEE754.
	<ul> <li>An extended floating-point operation coprocessor (FXU) can be installed.</li> </ul>
	Supports 4 single-precision (32-bit) parallel operations
	Supports IEEE754-compliant data types and exceptions
	Rounding modes: Nearest, 0 direction, $+\infty$ direction, and $-\infty$ direction
	Handling on non-normalized numbers: Truncated to 0, or an exception is reported to comply with IEEE754.



Exceptions/interrupts	<ul> <li>16-level interrupt priority that can be specified for each channel</li> </ul>	
	<ul> <li>64-level interrupt priority that can be specified for each channel (Supported only when Architecture Identifier bit PID[31:24] = 07<sub>H</sub> (RH850G4MH2))</li> </ul>	
	<ul> <li>Vector selection method that can be selected according to performance requirements and the amount of consumed memory</li> <li>Direct branch method exception vector (direct vector method)</li> <li>Address-table-referencing indirect branch method exception vector (table reference method)</li> </ul>	
	<ul> <li>Support for high-speed context save and restoration processing on interrupt by using dedicated instructions (PUSHSP, POPSP)</li> </ul>	
	<ul> <li>Support for high-speed context save on interrupt by using the register bank feature</li> </ul>	
	<ul> <li>Support for restoration from the register bank using a dedicated instruction (RESBANK)</li> </ul>	
Memory management	A memory protection unit (MPU) can be installed.	
Caches • An instruction cache can be installed.		

## 1.2.1 Multiprocessing Environment

The RH850G4MH provides a multiprocessing environment for software running on the system. It is equipped with a multi-core support features to support MPMD (Multiple Program Multiple Data Stream) type multiprocessing environments.

A multi-core system incorporates two or more processors which execute different sequences of instructions in parallel. Its total processing performance is enhanced since it allows two or more programs to be executed simultaneously. On the other hand, the transfer of processing that spans over two or more processors will impose heavier software burden. If the processing is split in poor balance, for example, either one of the processors may have no instruction to execute, resulting in decrease in processing efficiency.



## Section 2 Instruction

## 2.1 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.

## 2.1.1 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 2.1.2, 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 2.2.3**, **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 2.1.3, 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 2.2.3**, **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 sub- opcode 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 2.2.3**, **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 2.2.3**, **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 2.2.3**, **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.



## (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.





## 2.1.2 Coprocessor Instructions

Instructions in the following format are defined as coprocessor instructions.

## (1) Coprocessor Instruction Format 1

This is a 32-bit instruction format used as a coprocessor instruction.



## (2) Coprocessor Instruction Format 2

This is a 48-bit instruction format used as an extended floating-point arithmetic instruction.



Coprocessor instructions define the functions of each coprocessor.

## (3) 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 the hardware manual of the product used.

When a coprocessor cannot be used, even if the instruction is an undefined opcode, a coprocessor unusable exception occurs in preference to a reserved instruction exception (RIE) if the opcode is included in the format above.



## (4) Coprocessor Instruction Code and Corresponding Coprocessor

Instruction codes in a coprocessor instruction format 1 are assigned to each coprocessor as shown in the following table. When a coprocessor instruction is executed and a coprocessor unusable exception occurs, the exception cause code is determined according to the following table. For details on the exception cause code of the coprocessor unusable exception, see the hardware manual of the product used.

In addition, since instruction codes in a coprocessor instruction format 2 are all handled as an extended floating-point arithmetic instruction, the exception cause code will be  $81_{\rm H}$ .

Instruction Code						Corresponding	Exception
Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21	Coprocessor	Cause Code
1	0	0	0	_	—	FPU	80 <sub>H</sub>
1	0	0	1	0	0	FPU	80 <sub>H</sub>
				0	1	FPU	80 <sub>H</sub>
				1	0	FXU	81 <sub>H</sub>
				1	1	FPU	80 <sub>H</sub>
1	0	1	0	—	—	Reserved	82 <sub>H</sub>
1	0	1	1	—	—	FXU	81 <sub>H</sub>
1	1	0	0	-	—	Reserved	82 <sub>H</sub>
1	1	0	1	_	—	Reserved	82 <sub>H</sub>
1	1	1	0	_	_	Reserved	82 <sub>H</sub>
1	1	1	1	_	—	Reserved	82 <sub>H</sub>



## 2.1.3 Reserved Instructions

An opcode reserved for future function extension and for which no instruction is defined is defined as a reserved instruction. A reserved instruction exceptions (RIE) can occur for the opcode of any reserved instruction.

The following opcodes are defined for this CPU as RIE instructions that will always cause a reserved instruction exception:





## 2.2 Basic Instructions

## 2.2.1 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<sub>H</sub>), 7FFF FFFF<sub>H</sub> 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
- CLIP.B: Signed data conversion from word to byte with saturation
- CLIP.BU: Unsigned data conversion from word to byte with saturation
- CLIP.H: Signed data conversion from word to halfword with saturation
- CLIP.HU: Unsigned data conversion from word to halfword with saturation
- 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: Restore registers from stack
- EI: Enable interrupt
- EIRET: Return from EI-level trap or interrupt
- FERET: Return from FE-level trap or interrupt
- FETRAP: FE-level trap
- HALT: Halt
- LDSR: Load to system register
- LDL.BU: Load linked byte unsigned
- LDL.HU: Load linked halfword unsigned
- LDL.W: Load linked word
- LDM.MP: Load Multiple MPU entries from memory (Supported only when Architecture Identifier bit  $PID[31:24] = 07_{H} (RH850G4MH2))$
- NOP: No operation
- POPSP: Pop registers from stack
- PREPARE: Save registers to stack
- PUSHSP: Push registers to stack
- RESBANK: Restore contexts from register bank
- RIE: Reserved instruction exception
- SNOOZE: Snooze
- STSR: Store contents of system register
- STC.B: Store conditional byte

RENESAS

- STC.H: Store conditional halfword
- STC.W: Store conditional word
- STM.MP: Store Multiple MPU entries to memory (Supported only when Architecture Identifier bit PID[31:24] =  $07_{\rm H}$  (RH850G4MH2))
- SWITCH: Jump with table look up
- SYNCE: Synchronize exceptions
- SYNCI: Synchronize instruction fetch
- SYNCM: Synchronize memory
- SYNCP: Synchronize pipeline
- SYSCALL: System call
- TRAP: Trap

## 2.2.2 Special Operations

## (1) Divide by Zero

The results of executing a divide instruction by a zero divisor are summarized below.

Quotient	Old Value Retained
Remainder	0
PSW.OV	1
PSW.S	0
PSW.Z	0



## 2.2.3 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 2.1**).
- Operation: Indicates the function of the instruction (for symbols, see **Table 2.2**).
- Format: Indicates the instruction format (see Section 2.1, Opcodes and Instruction Formats).
- Opcode: Indicates the bit field of the instruction opcode (for symbols, see **Table 2.3**).
- 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 2.1 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 selection ID
vector ×	Data to specify vector (x indicates the bit size)
cond	Condition code (see Table 2.4)
CCCC	4-bit data to specify condition code (see Table 2.4)
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> .
eh-et	Indicates multiple system registers of MPU entry (MPLA, MPUA, MPAT), from the entry number indicated by <i>eh</i> to the entry number indicated by <i>et</i> .
[]+	Post increment addressing
[]-	Post decrement addressing

#### Table 2.2Conventions of Operation (1/2)

Symbol	Meaning
<i>←</i>	Assignment
GR [a]	Value stored in general-purpose register <i>a</i>
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> .
CheckException(a)	Checks the conditions for generating the exception "a" and, if one is detected, suspends the instruction execution and performs exception processing.



Table 2.2 Conventi	ions of Operation (2/2)
Symbol	Meaning
zero-extend (n)	Zero-extends "n" to word
sign-extend (n)	Sign-extends "n" to word
load-memory (a, b)	Reads data of size <i>b</i> from address <i>a</i>
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 ≥ 7FFF FFFF <sub>H</sub> , n = 7FFF FFFF <sub>H</sub> .
	lf n ≤ 8000 0000 <sub>H</sub> , n = 8000 0000 <sub>H</sub> .
clip (a, b, c)	Performs saturated processing on the word data "a" assuming the sign "b" and converts it to data of the size "c".
	<ul> <li>If the sign "b" is Sign and the size "c" is Byte:</li> </ul>
	When 0000 $007F_H < a \le 7FFF FFFF_H$ , the result is 0000 $007F_H$ .
	When $8000\ 0000_{H} \le a < FFFFF80_{H}$ , the result is FFFF FF80 <sub>H</sub> .
	If the sign "b" is Unsign and the size "c" is Byte:
	<ul> <li>When 0000 00FF<sub>H</sub> &lt; a, the result is 0000 00FF<sub>H</sub>.</li> <li>If the sign "b" is Sign and the size "c" is Halfword:</li> </ul>
	When 0000 7FFF <sub>H</sub> < a $\leq$ 7FFF FFFF <sub>H</sub> , the result is 0000 7FFF <sub>H</sub> .
	When 8000 $0000_{\rm H} \le a < \rm FFFF 8000_{\rm H}$ , the result is FFFF 8000 <sub>H</sub> .
	<ul> <li>If the sign "b" is Unsign and the size "c" is Halfword:</li> </ul>
	When 0000 $\text{FFFF}_{H} < a$ , the result is 0000 $\text{FFFF}_{H}$ .
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
	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
P-TYPE_Addressing()	Handles post index increment/decrement addressing.
T TI L_AUUESSIIIY()	רמותוכים אישי וותכא וותרכוורבווגיתבטובווובות מתורבססווע.

## Table 2.2Conventions of Operation (2/2)

Table 2.3	Conventions of Opcode
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
1	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 2.4)
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 2.4 Condition 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



## 2.2.3.1 ADD

<Arithmetic instruction>

	Add register/immediate
ADD	Add
[Instruction format]	(1) ADD reg1, reg2
	(2) ADD imm5, reg2
[Operation]	(1) $GR[reg2] \leftarrow GR[reg2] + GR[reg1]$
	(2) $GR[reg2] \leftarrow GR[reg2] + sign-extend (imm5)$
[Format]	(1) Format I
	(2) Format II
[Opcode]	
	(1) 15 0 rrrrr001110RRRR
	(2) 15 0 rrrrr010010iiiii
[Flags]	CY "1" if a carry 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) 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. General- purpose register reg1 is not affected.
	(2) Adds the 5-bit immediate data, sign-extended to word length, to the word data of general-purpose register reg2 and stores the result in general-purpose register reg2.



## 2.2.3.2 ADDI

<Arithmetic instruction>

ADDI		Add immediate
		Add immediate
[Instruction format]	ADDI imm16, reg1, reg2	
[Operation]	$GR [reg2] \leftarrow GR [reg1] + sign-extend (imm16)$	
[Format]	Format VI	
[Opcode]		
	15 0 31 16 rrrrl10000RRRR	
[Flags]	CY "1" if a carry 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]	Adds the 16-bit immediate data, sign-extended to word length, to the wor purpose register reg1 and stores the result in general-purpose register reg register reg1 is not affected.	•



## 2.2.3.3 ADF

<Conditional Operation Instructions>

ADF			Add on condition flag
			Conditional add
[Instruction format]	ADF cc	ecc, reg1, reg2, reg3	
[Operation]	then GI	ations are satisfied R [reg3] ← GR [reg1] + GR [reg2] + 1 R [reg3] ← GR [reg1] + GR [reg2] + 0	
[Format]	Format	XI	
[Opcode]			
	15 rrrrr	0 31 16	
[Flags]	CY	"1" if a carry 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]

Adds 1 to the result of adding the word data of general-purpose register reg1 to the word data of general-purpose register reg2 and stores the result of addition in general-purpose register reg3, if the condition specified as condition code "cccc" is satisfied.

If the condition specified as condition code "cccc" is not satisfied, the word data of generalpurpose register reg1 is added to the word data of general-purpose register reg2, and the result is stored in general-purpose register reg3.

General-purpose registers reg1 and reg2 are not affected. Designate one of the condition codes shown in the following table as [cccc]. (cccc is not equal to 1101.)

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	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 prohibited	



## 2.2.3.4 AND

<Logical instruction>

AND		AND	,
		AND	1
[Instruction format]	AND reg	1, reg2	
[Operation]	GR[reg2]	] ← GR[reg2] AND GR[reg1]	
[Format]	Format I		
[Opcode]	15	0	
		0 01010RRRR	
[Flags]	CY	_	
	OV	0	
	S	"1" if operation result word data MSB is "1"; otherwise, "0".	
	Z	"1" if the operation result is "0"; otherwise, "0".	
	SAT	_	
[Description]	register r	e word data of general-purpose register reg2 with the word data of general-purpose eg1 and stores the result in general-purpose register reg2. General-purpose register of affected.	



## 2.2.3.5 ANDI

<Logical instruction>

ANDI		AND immediate
		AND immediate
[Instruction format]	ANDI imm16, reg1, reg2	
[Operation]	$GR[reg2] \leftarrow GR[reg1]$ AND zero-extend (imm16)	
[Format]	Format VI	
[Opcode]		
	15 0 31 16 rrrr110110RRRRR	
[Flags]	СҮ —	
	OV 0	
	S 0	
	Z "1" if the operation result is "0"; otherwise, "0".	
	SAT —	
[Description]	ANDs the word data of general-purpose register reg1 with the 16-bit extended to word length, and stores the result in general-purpose regi purpose register reg1 is not affected.	



## 2.2.3.6 Bcond

<Branch instruction>

Deend	Branch on condition code
Bcond	Conditional branch
[Instruction format]	(1) Bcond disp9
	(2) Bcond disp17
[Operation]	(1) if conditions are satisfied then $PC \leftarrow PC + sign$ -extend (disp9)
	(2) if conditions are satisfied then PC ← PC + sign-extend (disp17)
[Format]	(1) Format III
	(2) Format VII
[Opcode]	
	15 0
	(1) ddddd1011dddcccc
	ddddddd is the higher 8 bits of disp9.
	cccc is the condition code of the condition indicated by cond (For details, see <b>Table 2.5</b>
	Bcond Instructions).
	(2) 00000111111Dccccdddddddddddddd
	Ddddddddddddd is the higher 16 bits of disp17.
	Bcond 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.
	D:4 0	) of the O bit displacement is mached to "O". The summer DO value used for coloridation

[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.

Instruction		Condition Code (cccc)	Flag Status	Branch Condition
Signed integer	BGE	1110	(S  xor  OV) = 0	Greater than or equal signed
	BGT	1111	((S  xor OV)  or  Z) = 0	Greater than signed
	BLE	0111	((S xor OV) or Z) = 1	Less than or equal signed
	BLT	0110	(S xor OV) = 1	Less than signed
Unsigned integer	BH	1011	(CY or Z) = 0	Higher (Greater than)
	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 2.5 Bcond Instructions



### CAUTIONS

- 1. 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 saturated with a negative value; if the rownal operation, the S flag is not inverted even if an overflow occurs. Thus, the S flag is affected differently when the instruction is a saturate operation, as opposed to an ordinary arithmetic operation. A branch condition which is an XOR of S and OV flags will therefore, have no meaning.
- 2. For Bcond disp17 (instruction format (2)), BR (0101) cannot be specified as the condition code.



### 2.2.3.7 BINS

BINS		Bitfield Insert
		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] (provide the set of the set $	00s-1:0)
[Format]	Format IX	
[Opcode]		
	15 0 31 16	
	rrrr111111RRRRRMMMMK0001001LLL0 (msb ≥ 16, lsb ≥ 16)	
	15 0 31 16	
	$rrrr111111RRRRRMMMMK0001011LLL0$ (msb $\geq$ 16, lsb < 16)	
	15 0 31 16	
	rrrr111111RRRRR MMMMK0001101LLL0 (msb < 16, lsb < 16)	
	Most significant bit of field to be updated: msb = pos+width-1	
	Least significant bit of field to be updated: $lsb = pos$	
	MMMM = lower 4 bits of msb, KLLL = lower 4 bits of lsb	
[Flags]	СҮ —	
	OV 0	
	S "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 $msb < lsb$ .



## 2.2.3.8 BSH

BSH		Byte swap halfword
		Byte swap of halfword data
[Instruction format]	BSH re	g2, reg3
[Operation]	GR[reg	3] ← GR[reg2] (23:16)    GR[reg2] (31:24)    GR[reg2] (7:0)    GR[reg2] (15:8)
[Format]	Format	XII
[Opcode]		
	15	0 31 16
	rrrrrl	L1111100000 wwww01101000010
[Flags]	СҮ	"1" when there is at least one byte value of zero in the lower halfword of the operation result; otherwise; "0".
	OV	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]	Execute	es endian swap.



### 2.2.3.9 BSW

BSW		Byte swap word
		Byte swap of word data
[Instruction format]	BSW re	eg2, reg3
[Operation]	GR[reg	3] ← GR[reg2] (7:0)    GR[reg2] (15:8)    GR[reg2] (23:16)    GR[reg2] (31:24)
[Format]	Format	XII
[Opcode]		
	15	0 31 16
	rrrrr	11111100000 wwww01101000000
[Flags]	CY	"1" when there is at least one byte value of zero in the word data of the operation result; otherwise; "0".
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" if operation result word data is "0"; otherwise, "0".
	SAT	
[Description]	Execute	es endian swap.



# 2.2.3.10 CALLT

<Special instruction>

CALLT	Call with table look up
	Subroutine call with table look up
[Instruction format]	CALLT imm6
[Operation]	adr $\leftarrow$ CTBP + zero-extend (imm6 logically shift left by 1) <sup>Note 1</sup>
	CheckException(MDP)
	$CTPC \leftarrow PC + 2 \text{ (return PC)}$
	$CTPSW(4:0) \leftarrow PSW(4:0)$
	$PC \leftarrow CTBP + zero-extend (Load-memory (adr, Halfword))$
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	Format II
[Opcode]	
	15 0 0000001000iiiii
[Flags]	CY —
	OV —
	S —
	Z —
	SAT —



[Description] The

The follow	ing steps	are taken
1110 10110 11	mg steps	

- <1> 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.
- <2> Confirms whether an exception is detected for the address generated in step <1>.
- <3> Transfers the contents of both return PC and PSW to CTPC and CTPSW.
- <4> Loads the halfword entry data of the address generated in step <1> and zeroextend to word length.
- <5> Adds the CTBP value to the data generated in step <4> to generate a 32-bit target address.
- <6> Branches to the target address generated in step <5>.

#### CAUTIONS

- 1. When an exception occurs during memory access, the instruction execution is aborted after the end of the read cycle. An interrupt might be accepted after the end of the read 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.



# 2.2.3.11 CAXI

<Special instruction>

CAXI	Compare and exchange for interlock
	Comparison and swap
[Instruction format]	CAXI [reg1], reg2, reg3
[Operation]	adr ← GR[reg1] <sup>Note 1</sup> CheckException (MAE) CheckException (MDP)
	token ← Load-memory (adr, Word) result ← GR[reg2] – token If result == 0
	then Store-memory (adr, GR[reg3],Word) GR[reg3] ← token
	else Store-memory (adr, token, Word) GR[reg3] ← token
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format XI
[Opcode]	
	15         0         31         16           rrrrrlllllrRRRR         wwwww00011101110
[Flags]	CY "1" if a borrow to the MSB occurs in the result operation; otherwise, "0".
	OV "1" if overflow occurs in the operation result; 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 generalpurpose 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

- Although this instruction expects an atomic access to be made for the purpose of exclusive control, whether an atomic access can be made is determined by the target memory to be accessed and the bus system specifications. For details, see the hardware manual of the product used.
- 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.



# 2.2.3.12 CLIP.B

<Data manipulation instruction>

CLIP.B	Signed data conversion from word to byte with saturation
	(Signed) Data conversion from word to byte with saturation
[Instruction format]	CLIP.B reg1, reg2
[Operation]	$GR[reg2] \leftarrow clip (GR[reg1], Sign, Byte)$
[Format]	Format IX
[Opcode]	
	15         0         31         16           rrrrr111111RRRRR         000000000000000000000000000000000000
[Flags]	CY 0
	OV "1" if saturation occurs, otherwise, 0.
	S "1" if the operation result is negative, otherwise, 0.
	Z "1" if the operation result is 0, otherwise, 0.
	SAT "1" if $OV = 1$ , otherwise, does not change.
[Description]	Regards the word data in the general-purpose register reg1 as signed word data and stores it in reg2 as signed byte data. If the value of the original data exceeds $0000\ 007F_H$ which is the positive maximum value of byte data, the instruction stores $0000\ 007F_H$ in the general- purpose register reg2. If the value falls below FFFF FF80 <sub>H</sub> which is the negative minimum value, the instruction stores FFFF FF80 <sub>H</sub> in the general-purpose register reg2, with the SAT flag set (to 1), respectively. The general-purpose register reg1 is not affected by this operation.

### CAUTION



# 2.2.3.13 CLIP.BU

<Data manipulation instruction>

	Unsigned data conversion from word to byte with saturation
CLIP.BU	(Unsigned) Data conversion from word to byte with saturation
[Instruction format]	CLIP.BU reg1, reg2
[Operation]	GR[reg2] ← clip (GR[reg1], Unsign, Byte)
[Format]	Format IX
[Opcode]	15 0 31 16
[Flags]	rrrr111111RRRRR $000000000000000000000000000000000000$
[Description]	Regards the word data in the general-purpose register reg1 as unsigned word data and stores it in reg2 as unsigned byte data. If the value of the original data exceeds 0000 00FF <sub>H</sub> which is the maximum value of the byte data, the instruction stores 0000 00FF <sub>H</sub> in the general-purpose register reg2 and sets the SAT flag (to 1). The general-purpose register reg1 is not affected by this operation.

### CAUTION



# 2.2.3.14 CLIP.H

<Data manipulation instruction>

CLIP.H	Signed data conversion from word to halfword with saturation
	(Signed) Data conversion from word to halfword with saturation
[Instruction format]	CLIP.H reg1, reg2
[Operation]	GR[reg2] ← clip (GR[reg1], Sign, Halfword)
[Format]	Format IX
[Opcode]	
	15         0         31         16           rrrrrllllllrRRRR         00000000001100         16
[Flags]	CY 0
	OV "1" if saturation occurs, otherwise, 0.
	S "1" if the operation result is negative, otherwise, 0.
	Z "1" if the operation result is 0, otherwise, 0.
	SAT "1" if $OV = 1$ , otherwise, does not change.
[Description]	Regards the word data in the general-purpose register reg1 as signed word data and stores it in reg2 as signed halfword data. If the value of the original data exceeds 0000 7FFF <sub>H</sub> which is the positive maximum value of halfword data, the instruction stores 0000 7FFF <sub>H</sub> in the general-purpose register reg2. If the value falls below FFFF $8000_H$ which is the negative minimum value, the instruction stores FFFF $8000_H$ in the general-purpose register reg2, with the SAT flag set (to 1), respectively. The general-purpose register reg1 is not affected by this operation.

### CAUTION



# 2.2.3.15 CLIP.HU

<Data manipulation instruction>

CLIP.HU	Unsigned data conversion from word to halfword with saturation
	(Unsigned) Data conversion from word to halfword with saturation
[Instruction format]	CLIP.HU reg1, reg2
[Operation]	GR[reg2] ← clip (GR[reg1], Unsign, Halfword)
[Format]	Format IX
[Opcode]	
	15         0         31         16           rrrrr111111RRRRR         00000000001110
[Flags]	CY 0
	OV "1" if saturation occurs, otherwise, 0.
	S 0
	Z "1" if the operation result is 0, otherwise, 0.
	SAT "1" if OV = 1, otherwise, does not change.
[Description]	Regards the word data in the general-purpose register reg1 as unsigned word data and stores it in reg2 as unsigned halfword data. If the value of the original data exceeds 0000 $\text{FFFF}_{\text{H}}$ which is the maximum value of the halfword data, the instruction stores 0000 $\text{FFFF}_{\text{H}}$ in the general-purpose register reg2 and sets the SAT flag (to 1). The general-purpose register reg1 is not affected by this operation

### CAUTION



# 2.2.3.16 CLL

<Special instruction>

CLL	Clear Load Link
	Clear atomic manipulation link
[Instruction format]	CLL
[Operation]	LLbit $\leftarrow 0$
[Format]	Format X
[Opcode]	
	15         0         31         16           111111111111111111111111111111111111
[Flags]	CY — OV —
	S —
	Z — SAT —
	5A1 —
[Description]	Causes the link that is created with the LDL instruction to be lost. For operations related to the loss of a link, see the hardware manual of the product used.

### 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.



# 2.2.3.17 CLR1

	Clear bit
CLR1	Bit clear
[Instruction format]	(1) CLR1 bit#3, disp16[reg1]
	(2) CLR1 reg2, [reg1]
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>token ← Load-memory (adr, Byte)</li> <li>Z flag ← Not (extract-bit (token, bit#3))</li> <li>token ← clear-bit (token, bit#3)</li> <li>Store-memory (adr, token, Byte)</li> </ul>
	(2) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MDP) token $\leftarrow$ Load-memory (adr, Byte) Z flag $\leftarrow$ Not (extract-bit (token, reg2)) token $\leftarrow$ clear-bit (token, reg2) Store-memory (adr, token, Byte)
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	<ol> <li>Format VIII</li> <li>Format IX</li> </ol>
[Opcode]	
	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$
[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 cleared (0) and the data is written back to the original address. If the specified bit of the byte data read is 0, the Z flag is set (1); if the specified bit is 1, the Z flag is cleared (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 bits indicated by the lower three bits of reg2 are cleared (0), and the data is written back to the original address. If the specified bit of the byte data read is 0, the Z flag is set (1); if the specified bit is 1, the Z flag is cleared (0).
[Supplement]		Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is uted, and does not indicate the content of the specified bit after this instruction is uted.

### CAUTION

Although this instruction expects that atomic accesses are made for the purpose of exclusive control, whether atomic accesses are actually possible is determined by the specifications for the target memory and bus system. For details, see the hardware manual of the product used.



### 2.2.3.18 CMOV

CMOV				Conditional move
CMOV				Conditional move
[Instruction format]	(1) CMOV cccc	c, reg1, reg2, reg3		
	(2) CMOV ccco	c, imm5, reg2, reg3		
[Operation]	then GR[reg	s are satisfied g3] ← GR[reg1] 3] ← GR[reg2]		
	then GR[reg	s are satisfied g3] ← sign-extende 3] ← GR[reg2]	d (imm5)	
[Format]	(1) Format XI			
	(2) Format XII			
[Opcode]				
	15	0 31	16	
	(1) rrrr1111	11RRRRR wwwww01	1001cccc0	
	15	0 31	16	
	(2) rrrr1111	lliiii wwwww0l	1000cccc0	
[Flags]	СҮ —			
	ov —			
	s —			
	Z —			
	SAT —			



#### [Description]

(1) When the condition specified by condition code "cccc" is met, data in generalpurpose register reg1 is transferred to general-purpose register reg3. When that condition is not met, 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 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

(2) When the condition specified by condition code "cccc" is met, 5-bit immediate data sign- extended 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 general- purpose register reg3. Specify one of the condition 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]

See the description of the SETF instruction.



# 2.2.3.19 CMP

<Arithmetic instruction>

СМР	Compare register/immediate (5-bit)
СМР	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]	
	(1) 15 0 rrrrr001111RRRRR
	(2) 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]	<ul> <li>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.</li> <li>Comparison is performed by subtracting the reg1 contents from the reg2 word data.</li> <li>General-purpose registers reg1 and reg2 are not affected.</li> </ul>
	(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.

RENESAS

### 2.2.3.20 CTRET

<Special instruction>

CTRET		Return from CALLT
_		Return from subroutine call
[Instruction format]	CTRET	
[Operation]	$PC \leftarrow C$	TPC
	PSW (4:	$0) \leftarrow \text{CTPSW} (4:0)$
[Format]	Format 2	X
[Opcode]		
	15 000001	0 31 16 1111100000 000000101000100
[Flags]	CY	Value read from CTPSW is set.
	OV	Value read from CTPSW is set.
	S	Value read from CTPSW is set.
	Z	Value read from CTPSW is set.
	SAT	Value read from CTPSW is set.
[Description]		te return PC and PSW (the lower 5 bits) from the appropriate system register and 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 (the lower 5 bits).
	<2>	The values are restored in PC and PSW (the lower 5 bits) and the control is transferred to the return address.

#### CAUTION

When the CTRET instruction is executed, only the lower 5 bits of the PSW register are updated; the higher 27 bits retain their previous values.



### 2.2.3.21 DI

<Special instruction>

DI	Disable interrupt
	Disable EI level maskable exception
[Instruction format]	DI
[Operation]	PSW.ID $\leftarrow 1$ (Disables EI level maskable exception)
[Format]	Format X
[Opcode]	
	00000111111000000000001011000000
[Flags]	СҮ —
	OV —
	s —
	Z —
	SAT —
	ID 1
[Description]	Sets the ID bit of the PSW to 1 to disable EI level maskable exceptions from the next instruction of this instruction.
[Supplement]	Overwrite of the ID bit 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-privileged
	instruction. If the MCTL.UIC bit has been set to 1, this instruction can always be executed.



# 2.2.3.22 DISPOSE

<Special instruction>

	Function dispose
DISPOSE	Restore registers from stac
Tractions forms at ]	(1) DISDOSE $\frac{1}{12}$
[Instruction format]	<ol> <li>(1) DISPOSE imm5, list12</li> <li>(2) DISPOSE imm5, list12 [mog1]</li> </ol>
	(2) DISPOSE imm5, list12, [reg1]
[Operation]	<pre>(1) tmp ← sp + zero-extend (imm5 logically shift by 2) foreach (all regs in list12) {</pre>
	$sp \leftarrow tmp$ (2) $tmp \leftarrow sp + zero-extend (imm5 logically shift by 2)$ foreach (all regs in list12) {
[Format]	Format XIII
[Opcode]	
	15 0 31 16
	(1) 0000011001iiiiiLLLLLLLLL00000
	15 0 31 16
	(2) 0000011001iiiiiLLLLLLLLLRRRRR
	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 bit 21 in list12).

RENESAS

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<sub>H</sub>

[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. When an exception occurs during the execution of the instruction, the execution of the instruction is suspended even when not all general-purpose registers are restored. An interrupt might be accepted during restoring the general-purpose registers, or before updating the sp (r3) after the restoration is completed. In these cases, the sp retains the old value that is established before the instruction is executed. The PC is not altered if the instruction is in the instruction format (2). Once the instruction execution is suspended, it is unable to know which general-purpose registers have been restored. Since the return PC from the exception processing is that of this DISPOSE instruction, unless none of the resources associated with the execution of the DISPOSE instruction are altered during the exception processing, the DISPOSE instruction that has been suspended can be re-executed precisely after control is returned from the exception processing. The re-execution starts at the beginning of the DISPOSE instruction processing.
- 2. For instruction format (2) DISPOSE imm5, list12, [reg1], do not specify r0 for reg1.
- 3. If none of the general-purpose registers is specified in list12, no memory access is made and the instruction execution is completed. Since no memory access is made, no MDP exception is generated. The value of imm5 shifted 2 bits to the left is added to the sp. For the DISPOSE instruction of the instruction format (2), control is transferred to the address that is specified in the general-purpose register reg1.



# 2.2.3.23 DIV

<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         0         31         16           rrrrrlllllrRRRR         wwwww01011000000
[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. An overflow occurs when division by zero is executed. For details, see <b>Section 2.2.2, Special</b>

**Operations**.



[Supplement]Overflow occurs when the maximum negative value (8000 0000<sub>H</sub>) is divided by -1 with the<br/>quotient = 8000 0000<sub>H</sub> and when the data is divided by 0.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<br/>process the exception. The execution resumes at the original instruction address upon

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.



# 2.2.3.24 DIVH

<Divide instruction>

	Divide halfword
DIVH	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]	
	(1) 15 0 rrrrr000010RRRR
	RRRRR $\neq$ 00000 (Do not specify r0 for reg1.)
	rrrrr ≠ 00000 (Do not specify r0 for reg2.) 15 0 31 16
	(2) rrrr111111RRRR wwww0101000000
[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. An overflow occurs when division by zero is executed. For details, see <b>Section 2.2.2, Special Operations</b> .	
	(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. An overflow occurs when division by zero is executed. For details, see <b>Section 2.2.2, Special Operations</b> .	
[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. When an exception occurs during execution of this instruction, 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.	
	(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. If general-purpose register reg2 and general-purpose register reg3 are the same register, the remainder is stored in that register. 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 general- purpose 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).



## 2.2.3.25 DIVHU

<Divide instruction>

DIVHU		Divide halfword unsigned
		Division of (unsigned) halfword data
[Instruction format]	DIVHU 1	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 X	XI
[Opcode]		
	15	0 31 16
	rrrrr11	11111RRRRR wwww01010000010
[Flags]	CY	_
	OV	"1" if overflow occurs; otherwise, "0".
	S	"1" if the MSB of the word data of the quotient as the result of operation is "1"; otherwise, "0".
	Z	"1" if the operation result quotient is "0"; otherwise, "0".
	SAT	_
[Description]	general-	he word data of general-purpose register reg2 by the lower halfword data of purpose register reg1 and stores the quotient to general-purpose register reg2 with inder set to general-purpose register reg3. General-purpose register reg1 is not An overflow occurs when division by zero is executed. For details, see <b>Section</b>

2.2.2, Special Operations.



[Supplement]Overflow occurs by division by zero.If general-purpose register reg2 and general-purpose register reg3 are the same register, the<br/>remainder is stored in that register.When an exception occurs during the DIVHU instruction execution, the execution is<br/>aborted to process the exception. The execution resumes at the original instruction address<br/>upon returning from the exception. General-purpose register reg1 and general-purpose<br/>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.



## 2.2.3.26 DIVQ

<High-speed divide instructions>

DIVQ	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         0         31         16           rrrrrlllllrRRRR         wwwww01011111100
[Flags]	СҮ —
	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.
	An overflow occurs when division by zero is executed. For details, see Section 2.2.2, Special Operations.



[Supplement]	(1)	Overflow occurs when the maximum negative value ( $8000\ 0000_{\text{H}}$ ) is divided by $-1$
		(with the quotient = $8000\ 0000_{\rm H}$ ) and when the data is divided by 0.
		If general-purpose register reg2 and general-purpose register 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.
	(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 **APPENDIX A**.
- 3. If the number of execution cycles must always be constant to guarantee real-time features, use the ordinary division instruction.



Divide word unsigned quickly

# 2.2.3.27 DIVQU

<High-speed divide instructions>

DIVQU	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 0 31 16 rrrr111111RRRRR wwww01011111110
[Flags]	СҮ —
	OV "1" when overflow occurs; otherwise, "0".
	S "1" when the MSB of the word data of the quotient as the result of operation is "1"; 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.
	An overflow occurs when division by zero is executed. For details, see Section 2.2.2, Special Operations.



[Supplement]	(1)	An overflow occurs when there is division by zero.
		If general-purpose register reg2 and general-purpose register 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

### 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.

valid bits is 15 or less, and the operation is completed within 20 cycles.

- 2. For the accurate number of execution cycles, see **APPENDIX A**.
- 3. If the number of execution cycles must always be constant to guarantee real-time features, use the ordinary division instruction.



# 2.2.3.28 DIVU

<Divide instruction>

	Divide word unsigned
DIVU	
	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 0 31 16
	rrrr111111RRRRR wwwww01011000010
[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.
	An overflow occurs when division by zero is executed. For details, see <b>Section 2.2.2</b> ,
	An overnow occurs when division by zero is executed. For details, see Section 2.2.2, Special Operations

**Special Operations**.


[Supplement]

An overflow occurs when there is division by zero.

If general-purpose register reg2 and general-purpose register reg3 are the same register, the remainder is stored in that register.

When an exception occurs during the DIVU 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.



### 2.2.3.29 EI

	Enable interrupt
EI	Enable EI level maskable exception
[Instruction format]	EI
[Operation]	$PSW.ID \leftarrow 0 \text{ (enables EI level maskable exception)}$
[Format]	Format X
[Opcode]	
	15         0         31         16           1000011111100000         00000001011000000
[Flags]	CY — OV — S — Z —
	SAT — ID 0
[Description]	Clears the ID flag of the PSW to "0" and enables the acknowledgement of EI-level maskable exception starting the next instruction.
[Supplement]	If the MCTL.UIC bit has been cleared to 0, this instruction is a supervisor-privileged instruction. If the MCTL.UIC bit has been set to 1, this instruction can always be executed.



# 2.2.3.30 EIRET

EIRET	Return from EI level trap or interrupt
	Return from EI level exception
[Instruction format]	EIRET
[Operation]	$PC \leftarrow EIPC$
	$PSW \leftarrow EIPSW$
[Format]	Format X
[Opcode]	
	15 0 31 16
	0000011111100000000000101001000
[Flags]	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]	Returns execution from an EI level exception. The return PC and PSW are loaded from the EIPC and EIPSW registers and set in the PC and PSW, and control is passed.
	If $EP = 0$ , it means that interrupt (EIINT <i>n</i> ) processing has finished, so the corresponding bit of the ISPR register is cleared. For details, see ISPR register in the hardware manual of the product used.
[Supplement]	This instruction is a supervisor-privileged instruction.



### 2.2.3.31 FERET

<Special instruction>

FERET	Return from FE level trap or	rinterrupt
	Return from FE level e	exception
[Instruction format]	FERET	
[Operation]	$PC \leftarrow FEPC$ $PSW \leftarrow FEPSW$	
	rsw ← rersw	
[Format]	Format X	
[Opcode]		
	15 0 31 16	
	0000011111100000 000000101001010	
[Flags]	CY Value read from FEPSW is set	
	OV Value read from FEPSW is set	
	S Value read from FEPSW is set	
	Z Value read from FEPSW is set	
	SAT Value read from FEPSW is set	
[Description]	Returns execution from an FE level exception. The return PC and PSW are loaded fr FEPC and FEPSW registers and set in the PC and PSW, and control is passed.	om the
[Supplement]	This instruction is a supervisor-privileged 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.



## 2.2.3.32 FETRAP

		FE-level Trap
FETRAP		FE level software exception
[Instruction format]	FETRAP vector4	
[Operation]	$FEPC \leftarrow PC + 2 \text{ (return PC)}$	
	$FEPSW \leftarrow PSW$	
	$FEIC \leftarrow exception \ cause \ code^{Note \ 1}$	
	$\text{PSW.UM} \leftarrow 0$	
	$\text{PSW.NP} \leftarrow 1$	
	$\text{PSW.EP} \leftarrow 1$	
	$\text{PSW.ID} \leftarrow 1$	
	$PC \leftarrow exception handler address^{Note 1}$	
	<b>Note 1.</b> See the hardware manual of the product used.	
[Format]	Format I	
[Opcode]		
	15 0	
	0vvvv00001000000	
	Where vvvv is vector4.	
	Do not set $0_H$ to vector4 (vvvv $\neq 0000$ ).	
[Flags]	СҮ —	
	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 the hardware manual of the product used. Execution then branches to the exception handler address and exception handling is started.

**Table 2.6** shows the correspondence between vector4 and exception cause codes and exception handler address offset. Exception handler addresses are calculated based on the offset addresses listed in **Table 2.6**. For details, see the hardware manual of the product used.

vector4	Exception Cause Code	Exception Cause Code Offset Address			
0 <sub>H</sub>		Not specifiable			
1 <sub>H</sub>	0000 0031 <sub>H</sub>	30 <sub>H</sub>			
2 <sub>H</sub>	0000 0032 <sub>H</sub>				
F <sub>H</sub>	0000 003F <sub>H</sub>				

 Table 2.6
 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset



# 2.2.3.33 HALT

HALT	Halt	
	Halt	]
[Instruction format]	HALT	
[Operation]	Stop execution of subsequent instructions until a HALT state release request is generated.	
[Format]	Format X	
[Opcode]		
	15         0         31         16           00000011111100000         00000001001000000	
[Flags]	СҮ —	
	ov —	
	s —	
	Z —	
	SAT —	



[Description]	Places the CPU core in the HALT state.
	The return to the normal execution state from the HALT state is triggered by the occurrence of a specific exception request.
	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 conditions for releasing the HALT state are given below.
	• A terminating-type exception occurs
	Even if the conditions (values of PSW.ID and PSW.NP) for acknowledging the above exception are not satisfied, the HALT state is released if there is an exception request. (Example: The HALT state is released when an EIINT request occurs even when PSW.ID $= 1$ .)
	The HALT state is not released if the terminating-type exceptions are masked by the following mask functions:
	• Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller <sup>Note 1</sup> .
	• 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. The HALT state is released when the masking is carried out using only the ISPR, PLMR registers and PSW.EIMASK bit (Supported only when Architecture Identifier bit PID[31:24] = 07 <sub>H</sub> (RH850G4MH2)).
[Supplement]	This instruction is a supervisor-privileged instruction.



# 2.2.3.34 HSH

<Data manipulation instructions>

HSH	Halfword swap halfword
	Halfword swap of halfword data
[Instruction format]	HSH reg2, reg3
[Operation]	$GR[reg3] \leftarrow GR[reg2]$
[Format]	Format XII
[Opcode]	
	15         0         31         16           rrrrrllllll00000         wwww0ll01000110
[Flags]	<ul><li>CY "1" if the lower halfword of the operation result is "0"; otherwise, "0".</li><li>OV 0</li></ul>
	S "1" if operation result word data MSB is "1"; otherwise, "0".
	Z "1" if the lower halfword of the operation result is "0"; otherwise, "0".
	SAT —
[Description]	Stores the content of general-purpose register reg2 in general-purpose register reg3, and stores the flag judgment result in PSW.



# 2.2.3.35 HSW

<Data manipulation instructions>

HSW		Halfword swap word
		Halfword swap of word data
[Instruction format]	HSW re	eg2, reg3
[Operation]	GR[reg	3] ← GR[reg2] (15:0)    GR[reg2] (31:16)
[Format]	Format	XII
[Opcode]		
	15	0 31 16
	rrrrr?	11111100000 wwww01101000100
[Flags]	CY	"1" when there is at least one halfword of zero in the word data of the operation result; otherwise; "0".
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" if operation result word data is "0"; otherwise, "0".
	SAT	
[Description]	Execute	es endian swap.



# 2.2.3.36 JARL

<Branch instruction>

			Branch and register link
disp22, reg2			
disp32, reg1			
[reg1], reg3			
$g2] \leftarrow PC + 4$ PC + sign-external			
$g1] \leftarrow PC + 6$ PC + disp32	5		
g3] ← PC + 4 GR[reg1]			
t V			
t VI			
t XI			
0	31	16	
.1110dddddd	dddddddddd	lddd0	
	lddd is the high pecify r0 for reg	er 21 bits of disp22. 2.)	
0	31	16 47	32
)10111RRRRR	dddddddddd	lddd0 DDDDDDDDDDDDI	ססכ
			of disp32.
		16	
.111111RRRRR	WWWWW001011C	0000	
) 0 )1	DDDDDDDDDdd 00000 (Do not s 0 0111111RRRRR	DDDDDDDDDddddddddddd 00000 (Do not specify r0 for reg 0 31 0111111RRRRR wwww0010110	00101111RRRRR dddddddddddddddd DDDDDDDDDDDDD



	CV	
[Flags]	CY	
	OV	—
	S	_
	Z	_
	SAT	_
[Description]	dis	ves the current PC value + 4 in general-purpose register reg2, adds the 22-bit splacement data, sign-extended to word length, to PC; stores the value in and nsfers the control to PC. Bit 0 of the 22-bit displacement is masked to "0".
	dis	ves the current PC value + 6 in general-purpose register reg1, adds the 32-bit eplacement data to PC and stores the value in and transfers the control to PC. Bit 0 the 32-bit displacement is masked to "0".
	val	bres the current PC value + 4 in reg3, specifies the contents of reg1 for the PC lue, and then transfers the control. If reg1 and reg3 are the same, before store the lue of current PC +4, specifies the contents of reg1 for the PC.
[Supplement]		The provided the set of the set
	saves the JMP inst	struction corresponds to the call function of the subroutine control instruction, and e return PC address in the general register which is specified by JARL instruction. truction corresponds to the return function of the subroutine control instruction, and sed to specify general-purpose register containing the return address as reg1 to the C.

### 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.



### 2.2.3.37 JMP

<Branch instruction>

	Jump register
JMP	Linconditional broach (conjeter relative)
	Unconditional branch (register relative)
[Instruction format]	(1) JMP [reg1]
	(2) JMP disp32[reg1]
[Operation]	(1) $PC \leftarrow GR[reg1]$
	(2) $PC \leftarrow GR[reg1] + disp32$
[Format]	(1) Format I
	(2) Format VI
[Opcode]	
	15 0
	(1) 0000000011RRRRR
	15 0 31 16 47 32
	(2) 00000110111RRRRRddddddddddddddddddddddd
	DDDDDDDDDDDDDDDddddddddddddd is the higher 31 bits of disp32.
[Flags]	СҮ —
	OV —
	S —
	Z —
	SAT —
[Description]	(1) Transfers the control to the address specified by general-purpose register reg1. Bit 0 of the address is masked to "0".
	(2) 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.

RENESAS

# 2.2.3.38 JR

<Branch instruction>

JR		Jump relative
		Unconditional branch (PC relative)
[Instruction format]	(1) JR disp22	
	(2) JR disp32	
[Operation]	(1) PC $\leftarrow$ PC + sign-extend (disp	22)
	(2) $PC \leftarrow PC + disp32$	
[Format]	(1) Format V	
	(2) Format VI	
[Opcode]		
	15 0 31 (1) 0000011110ddddddddddd	16 dddddddd0
	ddddddddddddddddd is t	he higher 21 bits of disp22.
	15 0 31	16 47 32
	(2) 000001011100000dddddd	dddddddd0 DDDDDDDDDDDDDDD
	DDDDDDDDDDDDDDDdddddddd	adddddd is the higher 31 bits of disp32.
[Flags]	СҮ —	
	OV —	
	S —	
	Z —	
	SAT —	



[Description]	<ol> <li>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".</li> </ol>
	(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.



# 2.2.3.39 LD.B

<Load instruction>

LD.B	Load byte
	Load of (signed) byte data
[Instruction format]	<ol> <li>LD.B disp16[reg1], reg2</li> <li>LD.B disp23[reg1], reg3</li> <li>LD.B [reg1]+, reg3</li> <li>LD.B [reg1]-, reg3</li> </ol>
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR[reg2] ← sign-extend (Load-memory (adr, Byte))</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR[reg3] ← sign-extend (Load-memory (adr, Byte))</li> </ul>
	<ul> <li>(3) adr ← GR [reg1]<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR [reg3] ← sign-extend (Load-memory (adr, Byte))</li> <li>GR [reg1] ← GR [reg1] + 1</li> </ul>
	<ul> <li>(4) adr ← GR [reg1]<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR [reg3] ← sign-extend (Load-memory (adr, Byte))</li> <li>GR [reg1] ← GR [reg1] - 1</li> </ul>
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	<ol> <li>Format VII</li> <li>Format XIV</li> <li>Format XI</li> <li>Format XI</li> </ol>



[Opcode]



dddddd is the lower 7 bits 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,<br/>sign- extended to word length, to generate a 32-bit address. Byte data is read from the<br/>generated address, sign-extended to word length, and stored in general-purpose<br/>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, sign-extended to word length, and stored in general-purpose register reg3.
- (3) Reads the byte data addressed by the word data specified in the general-purpose register reg1, sign-extends it to word length, and stores the result in the general-purpose register reg3. Adds 1 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
- (4) Reads the byte data addressed by the word data specified in the general-purpose register reg1, sign-extends it to word length, and stores the result in the generalpurpose register reg3. Subtracts 1 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

#### CAUTION

Do not specify a same register in reg1 and reg3 for the instruction formats (3) and (4). If a same register is specified, the results of updating reg1 is stored for this CPU.



### 2.2.3.40 LD.BU

<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
	(3) LD.BU [reg1]+, reg3
	(4) LD.BU [reg1]–, reg3
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException(MDP)</li> <li>GR[reg2] ← zero-extend (Load-memory (adr, Byte))</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR[reg3] ← zero-extend (Load-memory (adr, Byte))</li> </ul>
	(3) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException(MDP) $GR[reg3] \leftarrow zero-extend (Load-memory (adr, Byte))$ $GR[reg1] \leftarrow GR[reg1] + 1$
	(4) adr $\leftarrow$ GR[reg1] <sup>Note 1</sup> CheckException(MDP) GR[reg3] $\leftarrow$ zero-extend (Load-memory (adr, Byte)) GR[reg1] $\leftarrow$ GR[reg1] - 1
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI
[Opcode]	
	1503116(1)rrrr11110bRRRRRddddddddddddddddddddddddddddddd

ddddddddddd is the higher 15 bits of disp16, and b is bit 0 of disp16.  $rrrrr \neq 00000$  (Do not specify r0 for reg2.)



[Flags]

[Description]

	15 0	31 16	47	32
(2)	00000111101RRRRR	wwwwwdddddd0101	וסססססססססססססס	D
Whe	re RRRRR = reg1, ww	www = reg3.		
ddd	dddd is the lower 7 b	its of disp23.		
DDD	odddddddddd is	the higher 16 bits of c	lisp23.	
	15 0	31 16	-	
(3)	000111111111RRRRR	wwwww01101110000		
	15 0	31 16	1	
(4)	00101111111RRRRR	wwwww01101110000		
			1	
CY	—			
ov	_			
S	_			
Z	_			
SAT				
(1)	sign-extended to wor	of general-purpose reg rd length, to generate a ero-extended to word l	a 32-bit address. Byte	e data is read from the
(2)	sign-extended to wor	of general-purpose reg rd length, to generate a ero-extended to word l	a 32-bit address. Byte	e data is read from the

- (3) Reads the byte data addressed by the word data specified in the general-purpose register reg1, zero-extends it to word length, and stores the result in the generalpurpose register reg3. Adds 1 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
- (4) Reads the byte data addressed by the word data specified in the general-purpose register reg1, zero-extends it to word length, and stores the result in the generalpurpose register reg3. Subtracts 1 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

#### CAUTIONS

1. Do not specify r0 in reg2 for the instruction format (1).

2. Do not specify a same register in reg1 and reg3 for the instruction formats (3) and (4). If a same register is specified, the results of updating reg1 is stored for this CPU.



### 2.2.3.41 LD.DW

<Load instruction>

	Load Double-w	/ord
LD.DW	Load of double-word of	Jata
[Instruction format]	LD.DW disp23[reg1], reg3	
[Operation]	adr ← GR[reg1] + sign-extend (disp23) <sup>Note 1</sup> CheckException (MAE) CheckException (MDP) data ← Load-memory (adr, Double-word) GR[reg3 + 1]    GR[reg3] ← data <b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.	
[Format]	Format XIV	
[Opcode]		
	15         0         31         16         47         32           00000111101RRRRR         www.wwdddddd01001         DDDDDDDDDDDDDDDDDDDDDDDDDDD	
	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]	СҮ —	
	OV —	
	S —	
	Z —	
	SAT —	



[Description]	Generates a 32-bit address by adding a 23-bit displacement value sign-extended to word length to the word data of general-purpose register reg1. Double-word data is read from the generated 32-bit address and the lower 32 bits are stored in general-purpose register reg3, and the higher 32 bits in reg3 + 1.
[Supplement]	reg3 must be an even-numbered register. If an odd-numbered register is specified in reg3, bit 0 of the register number is ignored and the register is handled as an even-numbered register.

#### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. No misalignment exception will occur, however, if the address calculation results in a word boundary.



## 2.2.3.42 LD.H

<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
	(3) LD.H [reg1]+, reg3
	(4) LD.H [reg1] <sup>-</sup> , reg3
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg2] ← sign-extend (Load-memory (adr, Halfword))</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg3] ← sign-extend (Load-memory (adr, Halfword))</li> </ul>
	<ul> <li>(3) adr ← GR [reg1]<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR [reg3] ← sign-extend (Load-memory (adr, Halfword))</li> <li>GR [reg1] ← GR [reg1] + 2</li> </ul>
	<ul> <li>(4) adr ← GR [reg1]<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR [reg3] ← sign-extend (Load-memory (adr, Halfword))</li> <li>GR [reg1] ← GR [reg1] - 2</li> </ul>
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI



[Opcode]

		15 0	31 16	_
	(1)	rrrrr111001RRRRR	ddddddddddddd	
	Whe	ere ddddddddddd	dd is the higher 15 bi	ts of disp16.
		15 0	31 16	47 32
	(2)	00000111100RRRR	wwwwwdddddd00111	סססססססססססססססס
	Whe	ere RRRRR = reg1, ww	www = reg3.	
	ddd	dddd is the lower side	bits 6 to 1 of disp23.	
	DDI	ODDDDDDDDDDDD i	s the higher 16 bits of	disp23.
		15 0	31 16	
	(3)	00010111111RRRRR	wwwww01101110100	
		15 0	31 16	1
	(4)	00100111111RRRRR	wwwww01101110100	
[Flags]	CY	_		
	OV			
	S	_		
	Z	_		

\_\_\_\_

SAT



[Description](1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data,<br/>sign-extended to word length, to generate a 32-bit address. Halfword data is read from<br/>this 32-bit address, sign-extended to word length, and stored in general-purpose<br/>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.
- (3) Reads the halfword data addressed by the word data specified in the general-purpose register reg1, sign-extends it to word length, and stores the result in the general-purpose register reg3. Adds 2 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
- (4) Reads the halfword data addressed by the word data specified in the general-purpose register reg1, sign-extends it to word length, and stores the result in the generalpurpose register reg3. Subtracts 2 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

### CAUTIONS

1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

2. Do not specify a same register in reg1 and reg3 for the instruction formats (3) and (4). If a same register is specified, the results of updating reg1 is stored for this CPU.



# 2.2.3.43 LD.HU

<Load instruction>

	Load halfword unsigned
LD.HU	Load of (signed) halfword data
[Instruction format]	(1) LD.HU disp16[reg1], reg2
	(2) LD.HU disp23[reg1], reg3
	(3) LD.HU [reg1]+, reg3
	(4) LD.HU [reg1]–, reg3
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg2] ← zero-extend (Load-memory (adr, Halfword))</li> </ul>
	<ul> <li>(2) adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup> CheckException (MAE) CheckException (MDP) GR[reg3] ← zero-extend (Load-memory (adr, Halfword))</li> </ul>
	(3) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MAE) CheckException (MDP) $GR[reg3] \leftarrow$ zero-extend (Load-memory (adr, Halfword)) $GR[reg1] \leftarrow GR[reg1] + 2$
	(4) adr $\leftarrow$ GR[reg1] <sup>Note 1</sup> CheckException (MAE) CheckException (MDP) GR[reg3] $\leftarrow$ zero-extend (Load-memory (adr, Halfword)) GR[reg1] $\leftarrow$ GR[reg1] - 2
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI



[Opcode]

	15	0	31 16	
(1)	rrrrr111111RRRF	R	dddddddddddd1	
			dd is the higher 15 bi	ts of disp16.
rrr	$rr \neq 00000$ (Do not	S	pecify r0 for reg2.)	
	15	0	31 16	47 32
(2)	00000111101RRRF	R	wwwwwdddddd00111	ססססססססססססססס
Whe	re RRRRR = reg1, w	w	www = reg3.	
ddd	ddd is the lower sid	e	bits 6 to 1 of disp23.	
DDD	DDDDDDDDDDDDD	is	the higher 16 bits of d	lisp23.
	15	0	31 16	
(3)	000111111111RRRF	R	wwwww01101110100	
	15	0	31 16	
(4)	00101111111RRRF	R	wwwww01101110100	
CY	—			
OV				

CY	_
OV	_
S	
Z	
SAT	_



[Description](1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data,<br/>sign-extended to word length, to generate a 32-bit address. Halfword data is read from<br/>this 32-bit address, zero-extended to word length, and stored in general-purpose<br/>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.
- (3) Reads the halfword data addressed by the word data specified in the general-purpose register reg1, zero-extends it to word length, and stores the result in the general-purpose register reg3. Adds 2 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
- (4) Reads the halfword data addressed by the word data specified in the general-purpose register reg1, zero-extends it to word length, and stores the result in the generalpurpose register reg3. Subtracts 2 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

### CAUTIONS

- 1. Do not specify r0 for reg2.
- 2. A misalignment exception (MAE) will occur if the address calculation results in misaligned access.
- 3. Do not specify a same register in reg1 and reg3 for the instruction formats (3) and (4). If a same register is specified, the results of updating reg1 is stored for this CPU.



## 2.2.3.44 LD.W

<Load instruction>

	Load word
LD.W	Load of word data
[Instruction format]	(1) LD.W disp16[reg1], reg2
	(2) LD.W disp23[reg1], reg3
	(3) LD.W [reg1]+, reg3
	(4) LD.W [reg1]–, reg3
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg2] ← Load-memory (adr, Word)</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg3] ← Load-memory (adr, Word)</li> </ul>
	(3) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MAE) CheckException (MDP) $GR[reg3] \leftarrow Load-memory (adr, Word)$ $GR[reg1] \leftarrow GR [reg1] + 4$
	(4) adr ← GR[reg1] <sup>Note 1</sup> CheckException (MAE) CheckException (MDP) GR[reg3] ← Load-memory (adr, Word) GR[reg1] ← GR[reg1] - 4
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI



[Opcode]

	15 0	31 16	
(1)	rrrr111001RRRRR	ddddddddddddd	
Whe	ere dddddddddddd	dd is the higher 15 bi	ts of disp16.
	15 0	31 16	47 32
(2)	00000111100RRRRR	wwwwwdddddd01001	ססססססססססססססססס
	ere RRRRR = reg1, www		
ddd	lddd is the lower side	bits 6 to 1 of disp23.	
DDD	oddddddddddd is	the higher 16 bits of c	lisp23.
	15 0	31 16	_
(3)	00010111111RRRRR	wwwww01101111000	
	15 0	31 16	1
(4)	00100111111RRRRR	wwwww01101111000	
[Flags] CY			
OV			
S	_		

Ζ

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. 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.
(3) Reads the word data addressed by the word data specified in the general-purpose register reg1 into the general-purpose register reg3. Adds 4 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
(4) Reads the word data addressed by the word data specified in the general-purpose register reg1 into the general-purpose register reg3. Subtracts 4 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

### CAUTIONS

1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

2. Do not specify a same register in reg1 and reg3 for the instruction formats (3) and (4). If a same register is specified, the results of updating reg1 is stored for this CPU.



## 2.2.3.45 LDL.BU

LDL.BU	Load Linked byte unsigned
	Load to start atomic byte data manipulation
[Instruction format]	LDL.BU [reg1], reg3
[Operation]	<ul> <li>adr ← GR[reg1]<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>GR[reg3] ← zero-extend (Load-memory (adr, Byte))</li> <li>LLbit ← 1<sup>Note 2</sup></li> <li>Note 1. An MDP exception may occur depending on the results of the address calculation.</li> <li>Note 2. For the link operation, see the hardware manual of the product used.</li> </ul>
[Format]	Format VII
[Opcode]	15 0 31 16 00001111111RRRRR wwww01101110000
[Flags]	CY       —         OV       —         S       —         Z       —         SAT       —



[Description]Reads byte data from memory for an atomic read-modify-write, zero-extends it to word<br/>length, and stores the results in the general-purpose register reg3. Then, generates a link<br/>that corresponds to the address range including the specified address.

Subsequently, the link is lost if specific conditions are established before the STC.B instruction corresponding to the LDL.BU instruction is executed. If the STC.B instruction is executed with the link being lost, the result of the STC.B instruction indicates a failure.

If the STC.B instruction is executed with the link maintained, the result of the STC.B instruction is a success, in which case the link is also lost.

The LDL.BU and STC.B instructions may be used to carry out memory updates precisely in a multi-core system. The LDL.BU instruction and the STC.B instructions are intended always to be used in pair.

#### CAUTION

If a link is generated with the LDL.BU instruction and the STC.H or STC.W instruction is used instead of the STC.B instruction, the result will be a failure and the link be lost.



### 2.2.3.46 LDL.HU

			Load Linked halfword unsigned
LDL.HU			
			Load to start atomic halfword data manipulation
[Instruction format]	LDL.HU	J [reg1], reg3	
[Operation]	adr ← G	R[reg1] <sup>Note 1</sup>	
		xception (MAE)	
	CheckE	cception (MDP)	
	GR[reg3 LLbit ←	$[B] \leftarrow \text{zero-extend (Load-n}]$	nemory (adr, Halfword))
	Note 1.	An MAE or MDP excep calculation.	tion may occur depending on the results of the address
	Note 2.		see the hardware manual of the product used.
[Format]	Format	VII	
[Opcode]			
	15	0 31	16
	000011	11111RRRRR wwww0110	01110100
		i	
[Flags]	CY	—	
	OV	—	
	S	_	
	Z	—	
	SAT	—	



[Description]Reads halfword data from memory for an atomic read-modify-write, zero-extends it to<br/>word length, and stores the results in the general-purpose register reg3. Then, generates a<br/>link that corresponds to the address range including the specified address.Subsequently, the link is lost if specific conditions are established before the STC.H

instruction corresponding to the LDL.HU instruction is executed. If the STC.H instruction is executed with the link being lost, the result of the STC.H instruction indicates a failure.

If the STC.H instruction is executed with the link maintained, the result of the STC.H instruction is a success, in which case the link is also lost.

The LDL.HU and STC.H instructions may be used to carry out memory updates precisely in a multi-core system. The LDL.HU instruction and the STC.H instructions are intended always to be used in pair.

#### CAUTIONS

1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

2. If a link is generated with the LDL.HU instruction and the STC.B or STC.W instruction is used instead of the STC.H instruction, the result will be a failure and the link be lost.



### 2.2.3.47 LDL.W

LDL.W       Load to start atomic word of the start atom	Load Linked word
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	data manipulation
$ \begin{bmatrix} \text{Operation} \end{bmatrix} & \text{adr} \leftarrow \text{GR}[\text{reg1}]^{\text{Note 1}} \\ & \text{CheckException (MAE)} \\ & \text{CheckException (MDP)} \\ & \text{GR}[\text{reg3}] \leftarrow \text{Load-memory (adr, Word)} \\ & \text{LLbit} \leftarrow 1^{\text{Note 2}} \\ & \text{Note 1.}  \text{An MAE or MDP exception might occur depending on the result of a calculation.} \\ & \text{Note 2.}  \text{For the link operation, see the hardware manual of the product used} \\ \end{bmatrix} $	
<ul> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>GR[reg3] ← Load-memory (adr, Word)</li> <li>LLbit ← 1<sup>Note 2</sup></li> <li>Note 1. An MAE or MDP exception might occur depending on the result of a calculation.</li> <li>Note 2. For the link operation, see the hardware manual of the product used</li> </ul>	
<ul> <li>CheckException (MDP)</li> <li>GR[reg3] ← Load-memory (adr, Word)</li> <li>LLbit ← 1<sup>Note 2</sup></li> <li>Note 1. An MAE or MDP exception might occur depending on the result of a calculation.</li> <li>Note 2. For the link operation, see the hardware manual of the product used</li> </ul>	
<ul> <li>GR[reg3] ← Load-memory (adr, Word)</li> <li>LLbit ← 1<sup>Note 2</sup></li> <li>Note 1. An MAE or MDP exception might occur depending on the result of a calculation.</li> <li>Note 2. For the link operation, see the hardware manual of the product used</li> </ul>	
<ul> <li>LLbit ← 1<sup>Note 2</sup></li> <li>Note 1. An MAE or MDP exception might occur depending on the result of a calculation.</li> <li>Note 2. For the link operation, see the hardware manual of the product used</li> </ul>	
calculation. <b>Note 2.</b> For the link operation, see the hardware manual of the product used	
	address
[Format] Format VII	1.
[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 multi- core system. The LDL.W and STC.W instructions are intended always to be used in pair.
[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.

#### CAUTIONS

1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

2. If a link is generated with the LDL.W instruction and the STC.B or STC.H instruction is used instead of the STC.W instruction, the result will be a failure and the link be lost.



#### 2.2.3.48 LDM.MP

<Special instruction>

Load Multiple MPU entries from memory

LDM.MP (Supported only when Architecture Identifier bit PID[31:24] = 07<sub>H</sub> (RH850G4MH2)) Load MPU entries

[Instruction format]	LDM.MP [reg1], eh-et,	
[Operation]	LDM.MP [reg1], en-et, if (PSW.UM==0) then if (eh $\leq$ et ) then cur $\leftarrow$ eh end $\leftarrow$ et tmp $\leftarrow$ reg1 while (cur $\leq$ end) { adr $\leftarrow$ tmp <sup>Note 1, Note 2</sup> CheckException(MDP) MPLA[cur] $\leftarrow$ Load-memory (adr, Word) tmp $\leftarrow$ tmp <sup>Note 1, Note 2</sup> CheckException(MDP) MPUA[cur] $\leftarrow$ Load-memory (adr, Word) tmp $\leftarrow$ tmp + 4 adr $\leftarrow$ tmp <sup>Note 1, Note 2</sup> CheckException(MDP) MPUA[cur] $\leftarrow$ Load-memory (adr, Word) tmp $\leftarrow$ tmp + 4 adr $\leftarrow$ tmp <sup>Note 1, Note 2</sup> CheckException(MDP) MPAT[cur] $\leftarrow$ Load-memory (adr, Word) tmp $\leftarrow$ tmp + 4 cur $\leftarrow$ cur + 1 } else	
	<ul><li>Note 1. The lower 2 bits of adr are masked by 0.</li><li>Note 2. An MDP exception may occur as a result of address calculation.</li></ul>	
[Format]	Format XI	



[-[]	
	15 0 31 16
	rrrr111111RRRRRwwww00101100110
	rrrrr indicates eh.
	wwwww indicates et.
	RRRR indicates reg1.
	CV
[Flags]	CY —
	OV —
	S —
	Z —
	SAT —
[Descriptions]	The word data is read from the address generated from the word data of the general-purpose register reg1 and stored to the MPU protection area setting system registers (MPLA, MPUA, and MPAT) according to the specified order. Word size is added to the address each time the read data is stored to the system register. The contents of these system registers is processed in ascending order, regardless of the value of MPIDX, from the entry number indicated by eh to that indicated by et (eh, eh+1, eh+2,, et). The bank specified by MPBK is only to be processed.
	Because it is an SV privilege instruction, a PIE exception will occur if it is executed when PSW.UM is set (1).
[Supplement]	This instruction stores data directly from memory to multiple target system registers. Using this instruction can make the operation more efficient than loading data from memory to a

get system registers. Using g data from memory to a general-purpose register by using the LD.W instruction, specifying the entry via MPIDX, and storing the data to a system register by using the LDSR instruction.

The lower 2-bit address generated from the general-purpose register reg1 is masked by 0 and aligned on a word boundary. The general-purpose register reg1 retains the original value after the instruction execution is complete.

This instruction is an SV privilege instruction.

#### CAUTION

When an exception or an interrupt occurs during instruction execution and even if data from memory has not been stored to all system registers, the instruction execution can be aborted and the exception or interrupt can be accepted, when the acceptance condition is satisfied. When the execution is suspended, it is impossible to know to which system registers data from memory has been stored. After the return from exception processing, the suspended LDM.MP instruction can be precisely re-executed as long as resources related to execution of the LDM.MP instruction are not changed during exception processing, for the return PC from an exception is considered to be PC of this LDM.MP instruction. This instruction re-execution restarts the LDM.MP instruction processing from the start.



# 2.2.3.49 LDSR

<Special instruction>

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] <sup>Note 1</sup>
	<b>Note 1.</b> An exception might occur depending on the access permission. For details, see the hardware manual of the product used.
[Format]	Format IX
[Opcode]	
	15 0 31 16 rrrr111111RRRRRssss00000100000
	rrrr: regID, ssss: selID, RRRR: reg2
[Flags]	СҮ —
	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 selection ID (regID, selID). General-purpose register reg2 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 the hardware manual of the product used.

#### CAUTIONS

1. 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.

2. The system register number or selection ID is a unique number used to identify each system register. How to access undefined registers is described in the hardware manual of the product used, but accessing undefined registers is not recommended.



# 2.2.3.50 LOOP

<Loop instruction>

LOOP	Lo	ор
	Lo	юр
[Instruction format]	LOOP reg1, disp16	
[Operation]	$\begin{array}{l} GR[reg1] \leftarrow GR[reg1] + (-1)^{Note  1} \\ \text{if } (GR[reg1]  ! = 0) \\ \text{then} \\ PC \leftarrow PC - \text{zero-extend (disp16)} \\ \end{array}$ $\begin{array}{l} \text{Note 1.} & -1 \ (FFFF \ FFFF_{H}) \ \text{is added. The carry flag is updated in the same way as when} \\ \text{the ADD instruction is executed.} \end{array}$	
[Format]	Format VII	
[Opcode]		
	15     0 31     16       00000110111RRRRR     dddddddddddddddddddddd       Where dddddddddddddddddddddddddddddddddd.	
[Flags]	<ul> <li>CY "1" if a carry occurs from MSB in the reg1 operation; otherwise, "0".</li> <li>OV "1" if an overflow occurs in the reg1 operation; otherwise, "0".</li> <li>S "1" if reg1 is negative; otherwise, "0".</li> <li>Z "1" if reg1 is 0; otherwise, "0".</li> <li>SAT —</li> </ul>	



[Description]	<ul> <li>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.</li> <li>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.</li> </ul>		
	• $-1$ (FFFF FFFF <sub>H</sub> ) 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.



#### 2.2.3.51 MAC

<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           rrrrrllllllRRRR         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 register (r0, r2, r4, ..., r30). The result is undefined if an odd-numbered register (r1, r3, ..., r31) is specified.



#### 2.2.3.52 MACU

<Multiply-accumulate instruction>

	Multiply and add word unsigned	
MACU	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           rrrrrllllllRRRR         wwww0011111mmmm0	
[Flags]	СҮ —	
	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 unsigned 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.



# 2.2.3.53 MOV

<Arithmetic instruction>

	Move register/immediate (5-bit)/immediate (3
MOV	Data tra
[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]	
	15 0
	(1) rrrr000000RRRR
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
	15 0
	(2) rrrr010000iiiii
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
	15 0 31 16 47 32
	(3) 00000110001RRRRR iiiiiiiiiiiiiiiiiiiiii

I (bits 47 to 32) refers to the higher 16 bits of 32-bit immediate data.



[Flags]	CY
[1 lags]	CI

[Flags]	CY	—
	OV	_
	S	—
	Ζ	—
	SAT	—
[Description]	• •	opies and transfers the word data of general-purpose register reg1 to general- rpose register reg2. General-purpose register reg1 is not affected.
	. ,	opies and transfers the 5-bit immediate data, sign-extended to word length, to neral- purpose register reg2.
	(3) Co	ppies 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).



# 2.2.3.54 MOVEA

<Arithmetic instruction>

MOVEA	Move effective address		
WOVEA	Effective address transfer		
[Instruction format]	MOVEA imm16, reg1, reg2		
[Operation]	$GR[reg2] \leftarrow GR[reg1] + sign-extend (imm16)$		
[Format]	Format VI		
[Opcode]			
[Flags]	15   0   31   16 $rrrrr110001RRRR iiiiiiiiiiiiiiiiiiiiiiiiiiiii$		
[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 reg	92.		



# 2.2.3.55 MOVHI

<Arithmetic instruction>

ΜΟΥΗΙ	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 0 31 16 rrrr110010RRRRR iiiiiiiiiiiiiii	
	$rrrrr \neq 00000$ (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 reg	j2.	



# 2.2.3.56 MUL

<Multiply instruction>

MUL	Multiply word by register/immediate (9-bit)
MOL	Multiplication of (signed) word data
[Instruction format]	(1) MUL reg1, reg2, reg3
	(2) MUL imm9, reg2, reg3
[Operation]	(1) $GR[reg3] \parallel GR[reg2] \leftarrow GR[reg2] \times GR[reg1]$
	(2) $GR[reg3] \parallel GR[reg2] \leftarrow GR[reg2] \times sign-extend (imm9)$
[Format]	(1) Format XI
	(2) Format XII
[Opcode]	
	15         0 31         16           (1)         rrrr111111RRRRR         wwwww01000100000
	15     0 31     16       (2)     rrrr111111111111111111111111111111111
	iiiii are the lower 5 bits of 9-bit immediate data. IIII are the higher 4 bits of 9-bit immediate data.
[Flags]	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]		en general-purpose register reg2 and general-purpose register reg3 are the same register, y the higher 32 bits of the multiplication result are stored in the register.



#### 2.2.3.57 MULH

<Multiply instruction>

<multiply instruction=""></multiply>	Multiply halfword by register/immediate (5-bit)
MULH	Multiplication of (signed) halfword data
[Instruction format]	(1) MULH reg1, reg2
	(2) MULH imm5, reg2
[Operation]	(1) $GR[reg2] \leftarrow GR[reg2] (15:0) \times GR[reg1] (15:0)$
	(2) $GR[reg2] \leftarrow GR[reg2] \times sign-extend (imm5)$
[Format]	(1) Format I
	(2) Format II
[Opcode]	
	15     0       (1)     rrrrr000111RRRRR
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
	(2) 15 0 rrrrr010111iiiii
	$rrrrr \neq 00000$ (Do not specify r0 for reg2.)
[Flags]	СҮ —
	ov —
	s —
	Z —
	SAT —



[Description]	<ol> <li>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.</li> </ol>	
	(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.



### 2.2.3.58 MULHI

<Multiply instruction>

MULHI	Multiply halfword by immediate (16-bit)		
WOLNI	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]			
	1503116rrrr110111RRRRiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii		
[Flags]	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 reg	g2.		



#### 2.2.3.59 MULU

<Multiply instruction>

MULU

Multiply word unsigned by r	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)	$GR[reg3]   GR[reg2] \leftarrow GR[reg2] \times zero-extend (imm9)$
[Format]	(1)	Format XI
	(2)	Format XII
[Opcode]		
		15 0 31 16
	(1)	rrrr111111RRRRRwwww01000100010
	$(\mathbf{a})$	15 0 31 16
	(2)	rrrrr111111iiiii wwwww01001IIII10
	iii	ii are the lower 5 bits of 9-bit immediate data.
		I are the higher 4 bits of 9-bit immediate data.
[Flags]	CY	_
	ov	_
	S	_
	Z	_

SAT —



[Description]	<ol> <li>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.</li> </ol>
	(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.



# 2.2.3.60 NOP

<Special instruction>

NOP		No operation
		No operation
[Instruction format]	NOP	
[Operation]	No operation is performed.	
[Format]	Format I	
[Opcode]		
	15     0       000000000000000000000000000000000000	
[Flags]	СҮ —	
	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.	



# 2.2.3.61 NOT

<Logical instruction>

NOT		NOT
		Logical negation (1's complement)
[Instruction format]	NOT re	eg1, reg2
[Operation]	GR[reg	$2] \leftarrow \text{NOT}(\text{GR[reg1]})$
[Format]	Format	Ι
[Opcode]		
	15 rrrrr	0 000001RRRRR
[Flags]	СҮ	_
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Ζ	"1" if the operation result is "0"; otherwise, "0".
	SAT	_
[Description]	-	lly negates the word data of general-purpose register reg1 using 1's complement and he result in general-purpose register reg2. General-purpose register reg1 is not l.



#### 2.2.3.62 NOT1

<Bit manipulation instruction>

NOT	NOT bi
NOT1	NOT bi
[Instruction format]	(1) NOT1 bit#3, disp16[reg1]
	(2) NOT1 reg2, [reg1]
[Operation]	<ul> <li>adr ← GR [reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>token ← Load-memory (adr, Byte)</li> <li>Z flag ← Not (extract-bit (token, bit#3))</li> <li>token ← not-bit (token, bit#3)</li> <li>Store-memory (adr, token, Byte)</li> </ul>
	(2) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MDP) token $\leftarrow$ Load-memory (adr, Byte) Z flag $\leftarrow$ Not (extract-bit (token, reg2)) token $\leftarrow$ not-bit (token, reg2) Store-memory (adr, token, Byte)
	<b>Note 1.</b> 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 ddddddddddddddddddd
	15     0 31     16       (2)     rrrrr111111RRRRR     000000011100010
[Flags]	СҮ —
	OV —
	<ul> <li>S —</li> <li>Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1".</li> </ul>
	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 $(0 \rightarrow 1, 1 \rightarrow 0)$ 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 $(0 \rightarrow 1, 1 \rightarrow 0)$ 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]	exec	Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is uted and does not indicate the content of the specified bit resulting from the instruction ution.

#### CAUTION

Although this instruction expects that atomic accesses are made for the purpose of exclusive control, whether atomic accesses are actually possible is determined by the specifications for the target memory and bus system. For details, see the hardware manual of the product used.



٦

# 2.2.3.63 OR

<Logical instruction>

Г

OR			OR
			OR
[Instruction format]	OR reg1	, reg2	
[Operation]	GR[reg2	] ← GR[reg2] OR GR[reg1]	
[Format]	Format I		
[Opcode]	45		
	15 rrrrr0(	0 D1000RRRR	
[Flags]	CY	_	
	OV	0	
	S	"1" if operation result word data MSB is "1"; otherwise, "0".	
	Z	"1" if the operation result is "0"; otherwise, "0".	
	SAT	_	
[Description]	register r	word data of general-purpose register reg2 with the word data of general-purpore reg1 and stores the result in general-purpose register reg2. General-purpose reg ot affected.	



# 2.2.3.64 ORI

<Logical instruction>

ORI	OF	R immediate (16-bit)
		OR immediate
[Instruction format]	ORI imm16, reg1, reg2	
[Operation]	$GR[reg2] \leftarrow GR[reg1] OR zero-extend (imm16)$	
[Format]	Format VI	
[Opcode]		
	15 0 31 16 rrrr110100RRRRR iiiiiiiiiiiiiiii	
[Flags]	CY —	
	OV 0	
	S "1" if operation result word data MSB is "1"; otherwise, "0".	
	Z "1" if the operation result is "0"; otherwise, "0".	
	SAT —	
[Description]	ORs the word data of general-purpose register reg1 with the 16-bit immedia extended to word length, and stores the result in general-purpose register reg purpose register reg1 is not affected.	



#### 2.2.3.65 POPSP

<Special instruction>

POPSP	Pop registers from stack
FUF3F	Pop registers from stack
[Instruction format]	POPSP rh-rt
[Operation]	if $rh \leq rt$
	then cur $\leftarrow$ rt end $\leftarrow$ rh
	$tmp \leftarrow sp$ while (cur $\geq$ end) {
	$adr \leftarrow tmp^{Note 1, Note 2}$
	CheckException(MDP)
	$GR[cur] \leftarrow Load-memory (adr, Word)$
	$\operatorname{cur} \leftarrow \operatorname{cur} - 1$
	$tmp \leftarrow tmp + 4$
	}
	$sp \leftarrow tmp$
	<ul><li>Note 1. An MDP exception might occur depending on the result of address calculation.</li><li>Note 2. The lower 2 bits of adr are masked to 0.</li></ul>
[Format]	Format XI
[Format]	Format XI
	Format XI
[Format] [Opcode]	Format XI
	Format XI
	15 0 31 16 01100111111RRRRR wwww00101100000
	15 0 31 16 01100111111RRRRR wwwww00101100000 RRRRR indicates rh.
	15 0 31 16 01100111111RRRRR wwww00101100000
[Opcode]	15     0 31     16       01100111111RRRRR     wwwww00101100000       RRRRR indicates rh.       wwwww indicates rt.
	15 0 31 16 01100111111RRRRR wwwww00101100000 RRRRR indicates rh.
[Opcode]	15     0 31     16       01100111111RRRRR     wwwww00101100000       RRRRR indicates rh.       wwwww indicates rt.
[Opcode]	15       0 31       16         01100111111RRRRR       wwwww00101100000         RRRRR indicates rh.         wwwww indicates rt.         CY       —
[Opcode]	15       0 31       16         01100111111RRRRR       wwwww00101100000         RRRRR indicates rh.         wwwww indicates rt.         CY       —         OV       —         S       —
[Opcode]	15       0 31       16         01100111111RRRRR       wwwww00101100000         RRRRR indicates rh.         wwwww indicates rt.         CY       —         OV       —



[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).
[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.)

#### CAUTIONS

- 1. If a specification is made such that the registers to be restored include the sp (r3) (rh has a value of 3 to 31), the sp is not loaded with the value that is read from memory. For this reason, the instruction can be resumed correctly even if it is suspended in the middle of execution.
- 2. When an exception occurs during the execution of the instruction, the execution of the instruction is suspended even when not all general-purpose registers are restored. An interrupt might be accepted during restoring the general-purpose registers, or before updating the sp after the restoration is completed. In these cases, the sp retains the old value that is established before the instruction is executed. Once the instruction execution is suspended, it is unable to know which general-purpose registers have been restored. Since the return PC from the exception processing is that of this POPSP instruction, unless none of the resources associated with the execution of the POPSP instruction are altered during the exception processing, the POPSP instruction that has been suspended can be re-executed precisely after control is returned from the exception processing. The re-execution starts at the beginning of the POPSP instruction processing.
- 3. If rh > rt, no memory access is performed and the execution of the instruction is completed. Since no memory access is made, no MDP exception is generated. The sp retains the original value.



### 2.2.3.66 PREPARE

<Special instruction>

	Save registers to stack
PREPARE	Create stack frame
[Instruction format]	(1) PREPARE list12, imm5
	(2) PREPARE list12, imm5, sp/imm <sup>Note 1</sup>
	<b>Note 1.</b> The sp/imm values are specified by bits 19 and 20 of the sub-opcode.
[Operation]	<pre>(1) tmp ← sp foreach (all regs in list12) { tmp ← tmp - 4 adr ← tmp<sup>Note 1, Note 2</sup> CheckException (MDP) Store-memory (adr, GR[reg in list12], Word) } sp ← tmp - zero-extend (imm5 logically shift left by 2)</pre>
	<pre>(2) tmp ← sp foreach (all regs in list12) { tmp ← tmp - 4 adr ← tmp<sup>Note 1, Note 2</sup> CheckException (MDP) Store-memory (adr, GR[reg in list12], Word) } sp ← tmp - zero-extend (imm5 logically shift left by 2)</pre>
	case $ff = 00: ep \leftarrow sp$ $ff = 01: ep \leftarrow sign-extend (imm16)$ $ff = 10: ep \leftarrow imm16$ logically shift left by 16 $ff = 11: ep \leftarrow imm32$
	<ul><li>Note 1. An MDP exception might occur depending on the result of address calculation.</li><li>Note 2. The lower 2 bits of adr are masked to 0.</li></ul>
[Format]	Format XIII



[Opcode]



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<sub>H</sub>

# [Flags] CY – OV –

 OV
 —

 S
 —

 Z
 —

 SAT
 —



[Description] (1)		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]	imm	2 general-purpose registers are saved in ascending order (r20, r21,, r31). 5 is used to create a stack frame that is used for auto variables and temporary data. lower two bits of the address specified by sp are masked to 0 and aligned to the word
	bour	dary.

#### CAUTIONS

- 1. When an exception occurs during the execution of the instruction, the execution of the instruction is suspended even when not all general-purpose registers are saved. An interrupt might be accepted during saving the general-purpose registers, or before updating the sp (r3) after the saving is completed. In these cases, the sp and ep (r30) retain the old values that are established before the instruction is executed. Once the instruction execution is suspended, it is unable to know which general-purpose registers have been saved. Since the return PC from the exception processing is that of this PREPARE instruction, unless none of the resources associated with the execution of the PREPARE instruction are altered during the exception processing, the PREPARE instruction that has been suspended can be re-executed precisely after control is returned from the exception processing. The re-execution starts at the beginning of the PREPARE instruction processing.
- 2. If none of the general-purpose registers is specified in list12, no memory access is made and the instruction execution is completed. Since no memory access is made, no MDP exception is generated. On the sp, a subtraction is performed with imm5 shifted 2 bits to the left. The ep is loaded with the specified value.



#### 2.2.3.67 PUSHSP

<Special instruction>

PUSHSP	Push register	s to stack				
FUSHSF	Push register	s to stack				
[Instruction format]	PUSHSP rh-rt					
[Operation]	if $rh \leq rt$					
	then cur $\leftarrow$ rh					
	$end \leftarrow rt$					
	$tmp \leftarrow sp$					
	while (cur $\leq$ end) {					
	$tmp \leftarrow tmp - 4$					
	$adr \leftarrow tmp^{Note 1, Note 2}$					
	CheckException (MDP)					
	Store-memory (adr, GR[cur], Word) cur ← cur + 1					
	$cur \leftarrow cur + r$					
	$sp \leftarrow tmp$					
	<ul><li>Note 1. An MDP exception might occur depending on the result of address calcula</li><li>Note 2. The lower 2 bits of adr are masked to 0.</li></ul>	ition.				
[Format]	Format XI					
[Opcode]						
[Opcode]						
	15 0 31 16					
	01000111111RRRRR wwww00101100000					
	RRRR indicates rh.					
	wwwww indicates rt.					
[Elage]	СҮ —					
[Flags]						
	ov —					
	S —					
	Z —					
	SAT —					
	5/11					



[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).
[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.)

#### CAUTIONS

- 1. When an exception occurs during the execution of the instruction, the execution of the instruction is suspended even when not all general-purpose registers are saved. An interrupt might be accepted during saving the general-purpose registers, or before updating the sp (r3) after the saving is completed. In these cases, the sp retains the old value that is established before the instruction is executed. Once the instruction execution is suspended, it is unable to know which general-purpose registers have been saved. Since the return PC from the exception processing is that of this PUSHSP instruction, unless none of the resources associated with the execution of the PUSHSP instruction are altered during the exception processing, the PUSHSP instruction that has been suspended can be re-executed precisely after control is returned from the exception processing. The re-execution starts at the beginning of the PUSHSP instruction processing.
- 2. If rh > rt, no memory access is performed and the execution of the instruction is completed. Since no memory access is made, no MDP exception is generated. The sp retains the original value.



#### 2.2.3.68 RESBANK

<Special instruction>

# RESBANK

Restore contexts from register bank

Restore from register bank

```
[Instruction format]
                             RESBANK
[Operation]
                             if (RBNR.BN > 0)
                             then
                                   if (RBCR0.MD == 0)
                                   then
                                                   adr \leftarrow RBIP - RBNR.BN \times 60h
                                                   CheckException (MDP)
                                                   GR[30] ← Load-memory (adr, Word)
                                                   cur \leftarrow 19
                                                   adr \leftarrow adr + 4
                                   else
                                                   adr \leftarrow RBIP – RBNR.BN × 90h
                                                   cur \leftarrow 31
                                                   adr \leftarrow adr + 4
                                   while (cur > 0) {
                                                   CheckException(MDP)
                                                   GR[cur] ← Load-memory (adr, Word)
                                                   cur \leftarrow cur - 1
                                                   adr \leftarrow adr + 4
                                   }
                                   CheckException (MDP)
                                   FPSR ← Load-memory (adr, Word)
                                   adr \leftarrow adr + 4
                                   CheckException (MDP)
                                   EIIC ← Load-memory (adr, Word)
                                   adr \leftarrow adr + 4
                                   CheckException (MDP)
                                   EIPSW \leftarrow Load-memory (adr, Word)
                                   adr \leftarrow adr + 4
                                   CheckException (MDP)
                                   EIPC ← Load-memory (adr, Word)
                                   RBNR.BN \leftarrow RBNR.BN - 1
                                   PSW.ID \leftarrow 1
                             else
                                   FEPC \leftarrow PC (return PC)
                                   FEPSW \leftarrow PSW
                                   FEIC \leftarrow Exception cause code (0000 001D_H)
```



	PSW.UM ← 0 PSW.NP ← 1 PSW.EP ← 1 PSW.ID ← 1 PC ← Exception handler address (offset address of 10 <sub>H</sub> )
[Format]	Format X
[Opcode]	15 0 31 16 0000011111100000 1000000101100000
[Flags]	CY       —         OV       —         S       —         Z       —         SAT       —
[Description]	Restores the values of the system registers and general-purpose registers from a register bank. If the value of RBNR.BN is 0, however, a resumable-type SYSERR exception is generated.
[Supplement]	This instruction is a supervisor-privileged instruction. When an exception occurs during the execution of the instruction, the execution of the instruction is suspended even when not all the system registers and general-purpose registers are restored. An interrupt might be accepted during restoring the general-purpose registers, or before updating the sp after the restoration is completed. In these cases, RBNR.BN and PSW.ID retain their original values established before the instruction is executed. Once the instruction execution is suspended, it is unable to know which registers have been restored. Since the return PC from the exception processing is that of this RESBANK instruction, unless none of the resources associated with the execution of the RESBANK instruction are altered during the exception processing, the RESBANK instruction that has been suspended can be re-executed precisely after control is returned from the exception processing. The re-execution starts at the beginning of the RESBANK instruction processing.



#### 2.2.3.69 RIE

<Special instruction>

<special instruction=""></special>		
RIE		Reserved instruction exception
		Reserved instruction exception
[Instruction format]	<ol> <li>(1) RIE</li> <li>(2) RIE imm5, imm4</li> </ol>	
[Operation]	$\begin{array}{l} \text{FEPC} \leftarrow \text{PC} \text{ (return PC)} \\ \text{FEPSW} \leftarrow \text{PSW} \\ \text{FEIC} \leftarrow \text{exception cause code (0000 0060_{\text{H}})} \\ \text{PSW.UM} \leftarrow 0 \\ \text{PSW.UM} \leftarrow 0 \\ \text{PSW.NP} \leftarrow 1 \\ \text{PSW.EP} \leftarrow 1 \\ \text{PSW.ID} \leftarrow 1 \\ \text{PC} \leftarrow \text{exception handler address (offset address 60_{\text{H}})} \end{array}$	
[Format]	<ol> <li>Format I</li> <li>Format X</li> </ol>	
[Opcode]	(1) $ \begin{array}{c} 15 & 0 \\ 0000000000000 \\ 15 & 0 31 & 16 \\ (2) \\ \hline 15 & 0 0 31 & 16 \\ \hline 15 & 0 0000000000000 \\ \hline 15 & 0 0 31 & 16 \\ \hline 15 & 0 0 31 & 16 \\ \hline 15 & 0 0 31 & 16 \\ \hline 15 & 0 0 31 & 16 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 0 0 \\ \hline 16 & 0 0 0 0 0 \\ \hline 16 & 0 0 0 \\ \hline 16 & 0 0 0 \\ \hline 16 & 0$	
[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 the hardware manual of the product used.

Execution then branches to the exception handler address and exception handling is started. Exception handler addresses are calculated based on the offset address  $60_{\rm H}$ . For details, see the hardware manual of the product used.



# 2.2.3.70 ROTL

<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) rrrr111111iiiii wwwww00011000100
	15 0 31 16
	(2) 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]	<ol> <li>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.</li> </ol>
	(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.



# 2.2.3.71 SAR

<Data manipulation instruction>

SAR	Shift arithmetic right by register/immediate (5-bit)
SAR	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)     rrrr111111RRRRR     0000000101000000
	(2) 15 0 rrrrr010101iiiii
	15     0     31     16       (3)     rrrrr111111RRRRR     wwwww00010100010
[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.



# 2.2.3.72 SASF

<Data manipulation instruction>

SASF						Shift and set flag condition
						Shift and flag condition setting
[Instruction format]	SASF co	ccc, reg2				
[Operation]	then C		← (GR[reg2] l		ft left by 1) OI ft left by 1) OF	
[Format]	Format 1	IX				
[Opcode]						
	15		0 31	16		
	rrrr1	111110cc	ccc 00000010	000000000		
[Flags]	СҮ					
	OV	_				
	S	_				
	Z	_				
	SAT	_				



#### [Description]

When the condition specified by condition code "cccc" is met, logically left-shifts data of general-purpose register reg2 by 1 bit, and sets (1) the least significant bit (LSB). If a condition is not met, logically left-shifts data of reg2 and clears the LSB.

Designate 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) =
1011	Н	(CY or Z) = 0	1111	GT	((S xor OV) or Z) =

[Supplement]

See the SETF instruction.



# 2.2.3.73 SATADD

SATADD       Saturated add register/immediate (5-bit)         [Instruction format]       (1) SATADD reg1, reg2         (2) SATADD imm5, rcg2       (3) SATADD reg1, reg2, reg3         [Operation]       (1) GR[reg2] — saturated (GR[reg2] + GR[reg1])         (2) GR[reg2] — saturated (GR[reg2] + GR[reg1])         (3) GR[reg3] — saturated (GR[reg2] + GR[reg1])         (4) GR[reg3] — saturated (GR[reg2] + GR[reg1])         (5) GR[reg3] — saturated (GR[reg2] + GR[reg1])         (7) Format I         (2) Format XI         [Opcode]         (1) format XI         (2) production (1) for reg2.)         (3) Format XI         (4) production (2) for reg2.)         (5) for 0 31 format 50         (3) format XI         [Opcode]         (4) production (2) for reg2.)         (5) format (3) format 50         (3) format 700000 (Do not specify r0 for reg2.)         (3) production (2) for 0 31 format 50         (3) production (2) for reg2.)         (4) production (2) for 0 31 format 50         (3) production (2) for 0 31 format 50         (3) production (2) for 0 31 format 50         (3) production (2) format 50 form MSB; otherwise, "0".         (3) production (2) format 50 format 50 format 50 format 50 format 50 format 50 format50 format50 format50 format50 format50 format50 forma	<saturated instru<="" operation="" th=""><th>ictions&gt;</th><th></th></saturated>	ictions>	
$[Instruction format] (1) SATADD reg1, reg2(2) SATADD imm5, reg2(3) SATADD reg1, reg2, reg3[Operation] (1) GR[reg2] \leftarrow saturated (GR[reg2] + GR[reg1])(2) GR[reg2] \leftarrow saturated (GR[reg2] + sign-extend (imm5))(3) GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])[Format] (1) Format I(2) Format I(3) Format I(3) Format XI[Opcode] (1) \frac{15  0}{rrrrr000110RRRR}rrrrrr \neq 00000 (Do not specify r0 for reg2.)(2) \frac{15  0}{rrrrrrt} \neq 00000 (Do not specify r0 for reg2.)(3) \frac{15  0}{(3)} \frac{16}{rrrrr111111RRRRR}wwwww01110111010[Flags] CY  ``I`` if a carry occurs from MSB; otherwise, "0".QV  ``I`` if suturated operation result is negative; otherwise, "0".$	CATADD	Saturated add register/immediate (5-b	it)
$[Instruction format] (1) SATADD reg1, reg2(2) SATADD imm5, reg2(3) SATADD reg1, reg2, reg3[Operation] (1) GR[reg2] \leftarrow saturated (GR[reg2] + GR[reg1])(2) GR[reg2] \leftarrow saturated (GR[reg2] + sign-extend (imm5))(3) GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])[Format] (1) Format I(2) Format II(3) Format XI[Opcode] (1) \frac{15 \qquad 0}{rrrrr 00010 IORRRRR}rrrrrr \neq 00000 (Do not specify r0 for reg2.)(2) \frac{15 \qquad 0}{rrrrrr 100011iiii}rrrrrr \neq 00000 (Do not specify r0 for reg2.)(3) \frac{15 \qquad 0}{(1) rrrrr 111111RRRRR}wwwww0111011101[Flags] CY \qquad "1" if a carry occurs from MSB; otherwise, "0".S \qquad "1" if saturated operation result is negative; otherwise, "0".Z \qquad "1" if saturated operation result is negative; otherwise, "0".$	SATADD		
		Saturated addition	n
	[Instruction format]	(1) SATADD reg1, reg2	
$[Operation] (1) GR[reg2] \leftarrow saturated (GR[reg2] + GR[reg1])  (2) GR[reg2] \leftarrow saturated (GR[reg2] + sign-extend (imm5))  (3) GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])  [Format] (1) Format I  (2) Format II  (3) Format XI  [Opcode] (1) \frac{15 \qquad 0}{rrrrr000110RRRR}rrrrr \neq 00000 (Do not specify r0 for reg2.)(2) \frac{15 \qquad 0}{(rrrrr01000111111)}rrrrr \neq 00000 (Do not specify r0 for reg2.)(3) \frac{15 \qquad 0}{rrrrr111111RRRRe}prover \neq 00000 (Do not specify r0 for reg2.)(3) \frac{15 \qquad 0}{rrrrr111111RRRRe}[Flags] CY "1" if a carry occurs from MSB; otherwise, "0".CY "1" if saturated operation result is regative; otherwise, "0".Z "1" if saturated operation result is "0"; otherwise, "0".$			
$[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) Format I  (2) Format II  (3) Format XI  [Opcode]  (1) \frac{15  0}{rrrrr000110RRRR}  rrrrr \neq 00000 (Do not specify r0 for reg2.)  (2) \frac{15  0}{rrrrr01000111111}  rrrrr \neq 00000 (Do not specify r0 for reg2.)  (3) \frac{15  0}{rrrrr111111RRRR}  rrrrr \neq 00000 (Do not specify r0 for reg2.)  (3) \frac{15  0}{rrrrr111111RRRRR}  [Flags] CY "1" if a carry occurs from MSB; otherwise, "0".OV  "1" if overflow occurs; otherwise, "0".  Z  "1" if saturated operation result is "0"; otherwise, "0". $			
$(2)  GR[reg2] \leftarrow saturated (GR[reg2] + sign-extend (imm5))$ $(3)  GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])$ $(1)  Format I$ $(2)  Format II$ $(3)  Format XI$ $(0)pcode]$ $(1)  \frac{15 \qquad 0}{rrrrr 000110 RRRR}$ $rrrrr \neq 00000 (Do not specify r0 for reg2.)$ $(2)  \frac{15 \qquad 0}{rrrrr 01000111111}$ $rrrrr \neq 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrrr \neq 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrrr \neq 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rrr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRRR}$ $rr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 11111 RRRR}$ $rr = 00000 (Do not specify r0 for reg2.)$ $(3)  \frac{15 \qquad 0}{rrrrr 111111 RRR}$ $rr = 00000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not specify r0 for reg2.)$ $rr = 0000 (Do not spec1$		(0) 2	
$[Format] (3) GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])$ $[Format] (1) Format I (2) Format II (3) Format XI [Opcode] (1) \frac{15  0}{rrrrr000110RRRR} rrrrr \neq 00000 (Do not specify r0 for reg2.) (2) \frac{15  0}{rrrrr01000111111} rrrrr \neq 00000 (Do not specify r0 for reg2.) (3) \frac{15  0}{rrrrr111111RRRRNwwwww01110111010} [Flags] CY  "1" if a carry occurs from MSB; otherwise, "0". (3) "1" if saturated operation result is negative; otherwise, "0". (4) "1" if saturated operation result is "0"; otherwise, "0". (5) "1" if saturated operation result is "0"; otherwise, "0".$	[Operation]	(1) $GR[reg2] \leftarrow saturated (GR[reg2] + GR[reg1])$	
$[Format] (3) GR[reg3] \leftarrow saturated (GR[reg2] + GR[reg1])$ $[Format] (1) Format I (2) Format II (3) Format XI [Opcode] (1) \frac{15  0}{rrrrr000110RRRR} rrrrr \neq 00000 (Do not specify r0 for reg2.) (2) \frac{15  0}{rrrrr101000111111} rrrrr \neq 00000 (Do not specify r0 for reg2.) (3) \frac{15  0  31  16}{(3) \frac{rrrrr111111RRRR}wwwww01110111010} [Flags] CY  "1" if a carry occurs from MSB; otherwise, "0". S  "1" if saturated operation result is negative; otherwise, "0".$		(2) $GR[reg2] \leftarrow saturated (GR[reg2] + sign-extend (imm5))$	
$[Format] (1) Format I  (2) Format II  (3) Format XI  [Opcode]   \begin{pmatrix} 1 & \frac{15 & 0}{prrrr000110RRRR} \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 \\ (2) & \frac{15 & 0}{prrrr01000111111} \\ rrrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 & 31 & 16 \\ (3) & \frac{15 & 0 & 31 & 16}{prrrr111111RRRRRwwwww01110111010} \\ [Flags] [Flags] CY & "1" if a carry occurs from MSB; otherwise, "0".  CV & "1" if saturated operation result is negative; otherwise, "0".  Z & "1" if saturated operation result is "0"; otherwise, "0". \\ Z & "1" if saturated operation result is "0"; otherwise, "0". \\ $			
(2) Format II (3) Format XI [Opcode] $ \begin{pmatrix} 1 & 15 & 0 \\ (1) & rrrr 000110RRRR \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 \\ (2) & rrrrr 010001iiiii \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 & 31 & 16 \\ (3) & rrrrr 111111RRRR wwww01110111010 \\ ] [Flags] [Flags] (Y & "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".$			
<ul> <li>(3) Format XI</li> <li>[Opcode]         <ul> <li>(1) <sup>15</sup> 0</li></ul></li></ul>	[Format]	(1) Format I	
[Opcode] $\begin{bmatrix} 15 & 0 \\ (1) & rrrrr 000110RRRR \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 \\ (2) & rrrrrr \neq 00000 (Do not specify r0 for reg2.) \\ (2) & rrrrrr \neq 00000 (Do not specify r0 for reg2.) \\ (3) & rrrrrr \neq 00000 (Do not specify r0 for reg2.) \\ (3) & rrrrrr111111RRRRR wwwww01110111010 \\ \hline \\ [Flags] \qquad CY & "1" if a carry occurs from MSB; otherwise, "0". \\ OV & "1" if overflow occurs; otherwise, "0". \\ S & "1" if saturated operation result is negative; otherwise, "0". \\ Z & "1" if saturated operation result is "0"; otherwise, "0". \\ \end{bmatrix}$		(2) Format II	
$[Flags] \begin{bmatrix} 15 & 0 \\ (1) & rrrrr 000110RRRR \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 \\ (2) & rrrrr 010001iiii) \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 & 31 & 16 \\ (3) & rrrrr111111RRRR wwwww01110111010 \\ \hline \\ CY & "1" if a carry occurs from MSB; otherwise, "0". \\ OV & "1" if overflow occurs; otherwise, "0". \\ S & "1" if saturated operation result is negative; otherwise, "0". \\ Z & "1" if saturated operation result is "0"; otherwise, "0". \\ \end{bmatrix}$		(3) Format XI	
$[Flags] \begin{bmatrix} 15 & 0 \\ (1) & rrrrr 000110RRRR \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 \\ (2) & rrrrr 010001iiii) \\ rrrrr \neq 00000 (Do not specify r0 for reg2.) \\ 15 & 0 & 31 & 16 \\ (3) & rrrrr111111RRRR wwwww01110111010 \\ \hline \\ CY & "1" if a carry occurs from MSB; otherwise, "0". \\ OV & "1" if overflow occurs; otherwise, "0". \\ S & "1" if saturated operation result is negative; otherwise, "0". \\ Z & "1" if saturated operation result is "0"; otherwise, "0". \\ \end{bmatrix}$			
$[Flags] (1) rrrrr000110RRRR (1) rrrrr \neq 00000 (Do not specify r0 for reg2.) (2) rrrrr010001iiiii rrrrr \neq 00000 (Do not specify r0 for reg2.) (3) rrrrr111111RRRR wwwww01110111010 [Flags] CY "1" if a carry occurs from MSB; otherwise, "0". OV "1" if overflow occurs; otherwise, "0". S "1" if saturated operation result is negative; otherwise, "0". Z "1" if saturated operation result is "0"; otherwise, "0".$	[Opcode]		
$[Flags] = CY = (11, 10, 0000) (Do not specify r0 for reg2.)$ $(2) \qquad 15 \qquad 0 \\ (2) \qquad rrrrrr \neq 00000 (Do not specify r0 for reg2.)$ $(3) \qquad 15 \qquad 0 \ 31 \qquad 16 \\ (3) \qquad rrrrrr111111RRRR wwwww01110111010$ $[Flags] = CY \qquad "1" if a carry occurs from MSB; otherwise, "0".$ $OV \qquad "1" if overflow occurs; otherwise, "0".$ $S \qquad "1" if saturated operation result is negative; otherwise, "0".$ $Z \qquad "1" if saturated operation result is "0"; otherwise, "0".$		15 0	
$[Flags] \begin{array}{c ccccccccccccccccccccccccccccccccccc$		(1) rrrr000110RRRR	
$[Flags] \begin{array}{c ccccccccccccccccccccccccccccccccccc$			
(2) $rrrrr 00000 (Do not specify r0 for reg2.)$ $15   0   31   16   (3)   rrrrr111111RRRRR wwww011101100 $ [Flags] (7) (7) (7) (7) (7) (7) (7) (7) (7) (7)			
$[Flags] rrrrr \neq 00000 (Do not specify r0 for reg2.)$ $[Flags] CY  ``1" if a carry occurs from MSB; otherwise, ``0".$ $OV  ``1" if overflow occurs; otherwise, ``0".$ $S  ``1" if saturated operation result is negative; otherwise, ``0".$ $Z  ``1" if saturated operation result is ``0"; otherwise, ``0".$			
150 3116(3)rrrrr111111RRRRRrrrrr111111RRRRRWWWWW01110111010(3)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".		(2) rrrr0100011111	
<ul> <li>(3) rrrrr111111RRRR wwww011101100</li> <li>[Flags] CY "1" if a carry occurs from MSB; otherwise, "0".</li> <li>OV "1" if overflow occurs; otherwise, "0".</li> <li>S "1" if saturated operation result is negative; otherwise, "0".</li> <li>Z "1" if saturated operation result is "0"; otherwise, "0".</li> </ul>		$rrrrr \neq 00000$ (Do not specify r0 for reg2.)	
<ul> <li>[Flags]</li> <li>CY "1" if a carry occurs from MSB; otherwise, "0".</li> <li>OV "1" if overflow occurs; otherwise, "0".</li> <li>S "1" if saturated operation result is negative; otherwise, "0".</li> <li>Z "1" if saturated operation result is "0"; otherwise, "0".</li> </ul>		15 0 31 16	
<ul> <li>OV "1" if overflow occurs; otherwise, "0".</li> <li>S "1" if saturated operation result is negative; otherwise, "0".</li> <li>Z "1" if saturated operation result is "0"; otherwise, "0".</li> </ul>		(3) rrrr111111RRRRRwwww01110111010	
<ul> <li>OV "1" if overflow occurs; otherwise, "0".</li> <li>S "1" if saturated operation result is negative; otherwise, "0".</li> <li>Z "1" if saturated operation result is "0"; otherwise, "0".</li> </ul>			
<ul> <li>S "1" if saturated operation result is negative; otherwise, "0".</li> <li>Z "1" if saturated operation result is "0"; otherwise, "0".</li> </ul>	[Flags]	CY "1" if a carry occurs from MSB; otherwise, "0".	
Z "1" if saturated operation result is "0"; otherwise, "0".		OV "1" if overflow occurs; otherwise, "0".	
		S "1" if saturated operation result is negative; otherwise, "0".	
SAT "1" if $OV = 1$ ; otherwise, does not change.		Z "1" if saturated operation result is "0"; otherwise, "0".	
		SAT "1" if $OV = 1$ ; otherwise, does not change.	



[Description]	(1)	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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg2, and when it exceeds the maximum negative value 8000 0000 <sub>H</sub> , $8000\ 0000_{\text{H}}$ 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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg2, and when it exceeds the maximum negative value $8000\ 0000_{\text{H}}$ , $8000\ 0000_{\text{H}}$ 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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg3, and when it exceeds the maximum negative value 8000 0000 <sub>H</sub> , $8000\ 0000_{\text{H}}$ is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.
[Supplement]	clear	SAT flag is a cumulative flag. The saturate result sets the flag to "1" and will not be red to "0" even if the result of the subsequent operation is not saturated. 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.



# 2.2.3.74 SATSUB

<Saturated operation instructions>

# SATSUB

Saturated subtract

Saturated subtraction

[Instruction format]		ATSUB reg1, reg2 ATSUB reg1, reg2, reg3
	(2) 51	11501 1051, 1052, 1055
[Operation]	(1) G	$R[reg2] \leftarrow saturated (GR[reg2] - GR[reg1])$
	(2) G	$R[reg3] \leftarrow saturated (GR[reg2] - GR[reg1])$
[Format]		ormat I
	(2) Fo	ormat XI
[Opcode]		
	(1) rr	0 rrr000101RRRRR
	rrrrr	$\neq$ 00000 (Do not specify r0 for reg2.)
	(2) [15]	0 31 16
[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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> 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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg3, and when it exceeds the maximum negative value 8000 $0000_{\text{H}}$ , 8000 $0000_{\text{H}}$ is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.
[Supplement]	clear	SAT flag is a cumulative flag. The saturate result sets the flag to "1" and will not be red to "0" even if the result of the subsequent operation is not saturated. 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.



# 2.2.3.75 SATSUBI

<Saturated operation instructions>

SATSUBI	Saturated subtract immediat
	Saturated subtractio
[Instruction format]	SATSUBI imm16, reg1, reg2
[Operation]	$GR[reg2] \leftarrow saturated (GR[reg1] - sign-extend (imm16))$
[Format]	Format VI
[Opcode]	
	$\frac{15 \qquad 0 \ 31 \qquad 16}{rrrrr110011RRRRR}$ iiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
[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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg2; if the result exceeds the maximum negative value 8000 0000 <sub>H</sub> , 8000 0000 <sub>H</sub> 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".

#### CAUTIONS

1. Use LDSR instruction and load data to the PSW to clear the SAT flag to "0".

2. Do not specify r0 for reg2.



# 2.2.3.76 SATSUBR

<Saturated operation instructions>

	Saturated subtract reverse
SATSUBR	
	Saturated reverse subtraction
[Instruction format]	SATSUBR reg1, reg2
[Operation]	$GR[reg2] \leftarrow saturated (GR[reg1] - GR[reg2])$
[Format]	Format I
[Opcode]	
	15 0
	rrrr000100RRRR
	$rrrrr \neq 00000$ (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 <sub>H</sub> , 7FFF FFFF <sub>H</sub> is stored in reg2; if the result exceeds the maximum negative value 8000 $0000_{\text{H}}$ , 8000 $0000_{\text{H}}$ 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".

#### CAUTIONS

1. Use LDSR instruction and load data to the PSW to clear the SAT flag to "0"

2. Do not specify r0 for reg2.



# 2.2.3.77 SBF

<Conditional operation instructions>

SBF		Subtract on condition flag
		Conditional subtraction
[Instruction format]	SBF cccc, reg1, reg2, reg3	
[Operation]	if conditions are satisfied	
	then $GR[reg3] \leftarrow GR[reg2] - GR[reg1] - 1$	
	else GR[reg3] $\leftarrow$ GR[reg2] – GR[reg1] – 0	
[Format]	Format XI	
[Opcode]		
	15 0 31 16	
	rrrr111111RRRRR wwww011100cccc0	
[Flags]	CY "1" if a borrow occurs from MSB; otherwise, "	ʻ0".
	OV "1" if overflow occurs; otherwise, "0".	
	S "1" if operation result is negative; otherwise, "	0".
	Z "1" if operation result is "0"; otherwise, "0".	
	SAT —	



[Description]

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 the result in general-purpose register reg3.

General-purpose registers reg1 and register 2 are not affected.

Designate one of the condition codes shown in the following table as [cccc]. (However, cccc cannot equal 1101.)

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	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 p	rohibited



# 2.2.3.78 SCH0L

SCH0L		Search zero from left
		Bit (0) search from MSB side
[Instruction format]	SCH0L	reg2, reg3
[Operation]	GR[reg:	3] ← search zero from left of GR[reg2]
[Format]	Format	IX
[Opcode]		
	15 rrrrr1	0 31 16 1111100000 wwww01101100100
[Flags]	СҮ	"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]	writes th general-	s word data of general-purpose register reg2 from the left side (MSB side), and ne number of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to purpose register reg3 (e.g., when bit 31 of reg2 is 0, $01_{\rm H}$ is written to reg3).
		it (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If 0) found is the LSB, the CY flag is set (1).



# 2.2.3.79 SCH0R

SCH0R		Search zero from right
		Bit (0) search from LSB side
[Instruction format]	SCH0R	reg2, reg3
[Operation]	GR[reg	3] ← search zero from right of GR[reg2]
[Format]	Format	IX
[Opcode]		
	15 rrrrr1	0 31 16 .1111100000 wwww01101100000
[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]	writes tl general-	is word data of general-purpose register reg2 from the right side (LSB side), and the number of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to - purpose register reg3 (e.g., when bit 0 of reg2 is 0, $01_{\rm H}$ is written to reg3).
		it (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If 0) found is the MSB, the CY flag is set (1).



# 2.2.3.80 SCH1L

SCH1L		Search one from left
		Bit (1) search from MSB side
[Instruction format]	SCH1L	reg2, reg3
[Operation]	GR[reg	3] $\leftarrow$ search one from left of GR[reg2]
[Format]	Format	IX
[Opcode]		
	15 rrrrr1	0 31 16 .1111100000 wwww01101100110
[Flags]	СҮ	"1" if bit (1) is found eventually; otherwise, "0".
	OV	0
	S	0
	Z	"1" if bit (1) is not found; otherwise, "0".
	SAT	_
[Description]	writes th general- When b	is word data of general-purpose register reg2 from the left side (MSB side), and the number of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to - purpose register reg3 (e.g., when bit 31 of reg2 is 1, $01_{\rm H}$ is written to reg3). it (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If 1) found is the LSB, the CY flag is set (1).



# 2.2.3.81 SCH1R

SCH1R		Search one from right
		Bit (1) search from LSB side
[Instruction format]	SCH1R	reg2, reg3
[Operation]	GR[reg	3] $\leftarrow$ search one from right of GR [reg2]
[Format]	Format	IX
[Opcode]		
	15 rrrrr	0 31 16
[Flags]	CY	"1" if bit (1) is found eventually; otherwise, "0".
	OV	0
	S	0
	Z	"1" if bit (1) is not found; otherwise, "0".
	SAT	
[Description]	writes t general When b	es word data of general-purpose register reg2 from the right side (LSB side), and he number of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to - purpose register reg3 (e.g., when bit 0 of reg2 is 1, $01_{\rm H}$ is written to reg3). bit (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If
		(1) found is the MSB, the CY flag is set (1).



# 2.2.3.82 SET1

<Bit manipulation instruction>

	Set bit
SET1	
	Bit setting
[Instruction format]	(1) SET1 bit#3, disp16[reg1]
	(2) SET1 reg2, [reg1]
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup> CheckException (MDP) token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, bit#3)) token ← set-bit (token, bit#3) Store-memory (adr, token, Byte)</li> </ul>
	<ul> <li>(2) adr ← GR[reg1]<sup>Note 1</sup> CheckException (MDP) token ← Load-memory (adr, Byte) Z flag ← Not (extract-bit (token, reg2)) token ← set-bit (token, reg2) Store-memory (adr, token, Byte)</li> </ul>
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	<ol> <li>Format VIII</li> <li>Format IX</li> </ol>
[Opcode]	
	1503116(1)00bbb111110RRRRRddddddddddddddddddddddddddddddd
	15     0 31     16       (2)     rrrrr111111RRRRR     000000011100000
[Flags]	CY – OV – S –
	<ul> <li>Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1".</li> <li>SAT —</li> </ul>



[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, 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]		Z flag of PSW indicates the initial status of the specified bit (0 or 1) and does not cate the content of the specified bit resulting from the instruction execution.

#### CAUTION

Although this instruction expects that atomic accesses are made for the purpose of exclusive control, whether atomic accesses are actually possible is determined by the specifications for the target memory and bus system. For details, see the hardware manual of the product used.



# 2.2.3.83 SETF

<Data manipulation instruction>

SETF		Set flag condition
•=		Flag condition setting
[Instruction format]	SETF cccc, reg2	
[Operation]	if conditions are satisfied	
	then GR[reg2] $\leftarrow 0000 \ 0001_{\text{H}}$	
	else GR[reg2] $\leftarrow 0000\ 0000_{\rm H}$	
[Format]	Format IX	
[Opcode]		
	<u>15 0 31 16</u>	
	rrrr1111110cccc000000000000000000000000	
[Flags]	СҮ —	
	OV —	
	S —	
	Z —	
	SAT —	



#### [Description]

When the condition specified by condition code "cccc" is met, stores "1" to generalpurpose register reg2 if a condition is met and stores "0" if a condition is not met.

Designate one of the condition 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) =
1011	Н	(CY or Z) = 0	1111	GT	((S xor OV) or Z) =

#### [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 (a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, 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<sub>n</sub>. 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* (*an*) is stored in register Ra. By performing a logical operation to Ra<sub>n</sub> 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.



# 2.2.3.84 SHL

<Data manipulation instruction>

SHL	Shift logical left by register/immediate (5-bit)
SHL	Logical left shift
[Instruction format]	(1) SHL reg1, reg2
	(2) SHL imm5, reg2
	(3) SHL reg1, reg2, reg3
[Operation]	(1) $GR[reg2] \leftarrow GR[reg2]$ logically shift left by $GR[reg1]$
	(2) $GR[reg2] \leftarrow GR[reg2]$ logically shift left by zero-extend (imm5)
	(3) $GR[reg3] \leftarrow GR[reg2]$ logically shift left by $GR[reg1]$
[Format]	(1) Format IX
	(2) Format II
	(3) Format XI
[Opcode]	
	15     0 31     16       (1)     rrrr111111RRRRR     000000011000000
	(2) 15 0 rrrrr010110iiiii
	15     0 31     16       (3)     rrrrr111111RRRRR     wwwww00011000010
[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) Logically left-shifts the word data of general-purpose register reg2 by "n" (0 to +31),<br/>the position specified by the lower 5 bits of general-purpose register reg1, by shifting<br/>"0" to LSB. The result is written to general-purpose register reg2. General-purpose<br/>register reg1 is not affected.

- (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.



# 2.2.3.85 SHR

<Data manipulation instruction>

SHR	Shift logical right by register/immediate (5-bit)
SIIK	Logical right shift
[Instruction format]	(1) SHR reg1, reg2
	(2) SHR imm5, reg2
	(3) SHR reg1, reg2, reg3
[Operation]	(1) $GR[reg2] \leftarrow GR[reg2]$ logically shift right by $GR[reg1]$
	(2) $GR[reg2] \leftarrow GR[reg2]$ logically shift right by zero-extend (imm5)
	(3) $GR[reg3] \leftarrow GR[reg2]$ logically shift right by $GR[reg1]$
[Format]	(1) Format IX
	(2) Format II
	(3) Format XI
[Opcode]	
	(1) rrrr111111RRRRR000000000000000000000000
	(2) rrrr010100iiiii
	15 0 31 16
	(3) rrrr111111RRRRR wwww00010000010
[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) 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 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.



# 2.2.3.86 SLD.B

<Load instruction>

SLD.B	Short format load byte
	Load of (signed) byte data
[Instruction format]	SLD.B disp7[ep], reg2
[Operation]	$adr \leftarrow ep + zero-extend (disp7)^{Note 1}$
	CheckException(MDP)
	$GR[reg2] \leftarrow sign-extend (Load-memory (adr, Byte))$
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0 rrrrr0110dddddd
[Flags]	СҮ —
	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.



# 2.2.3.87 SLD.BU

<Load instruction>

SLD.BU	Short format load byte unsigned
320.00	Load of (unsigned) byte data
[Instruction format]	SLD.BU disp4[ep], reg2
[Operation]	$adr \leftarrow ep + zero-extend (disp4)^{Note 1}$
	CheckException (MDP)
	$GR[reg2] \leftarrow$ zero-extend (Load-memory (adr, Byte))
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	$\frac{15 \qquad 0}{rrrrr \neq 00000 \text{ (Do not specify r0 for reg2.)}}$
[Flags]	СҮ —
	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.



#### 2.2.3.88 SLD.H

<Load instruction>

SLD.H	Short format load halfword
	Load of (signed) halfword data
[Instruction format]	SLD.H disp8[ep], reg2
[Operation]	adr $\leftarrow$ ep + zero-extend (disp8) <sup>Note 1</sup>
	CheckException (MAE)
	CheckException (MDP)
	$GR[reg2] \leftarrow sign-extend (Load-memory (adr, Halfword))$
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0 rrrr1000dddddd
	dddddd is the higher 7 bits of disp8.
[Flags]	СҮ —
	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. Halfword data is read from this 32-bit address, sign-extended to word length, and stored in general-purpose register reg2.

#### CAUTION

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.



# 2.2.3.89 SLD.HU

<Load instruction>

<load instruction=""></load>	
SLD.HU	Short format load halfword unsigned
	Load of (unsigned) halfword data
[Instruction format]	SLD.HU disp5[ep], reg2
[Operation]	$adr \leftarrow ep + zero-extend (disp5)^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	$GR [reg2] \leftarrow zero-extend (Load-memory (adr, Halfword))$
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0
	rrrr0000111dddd
	$rrrrr \neq 00000$ (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 data, zero-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.

#### CAUTIONS

- 1. Do not specify r0 for reg2.
- 2. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.



#### 2.2.3.90 SLD.W

<Load instruction>

SLD.W	Short format load word
SLD.W	Load of word data
[Instruction format]	SLD.W disp8 [ep], reg2
[Operation]	$adr \leftarrow ep + zero-extend (disp8)^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	GR[reg2] ← Load-memory (adr, Word)
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0
	rrrrr1010dddddd
	dddddd is the higher 6 bits of disp8.
[Flags]	СҮ —
	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.

#### CAUTION

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

# 2.2.3.91 SNOOZE

<Special instruction>

SNOOZE		S
		s
[Instruction format]	Snooze	
[Operation]	Snooze while hardware-defined period	
[Format]	Format X	
[Opcode]		
	15         0         31         16           0000111111100000         00000001001000000         16	
[Flags]	СҮ —	
	ov —	
	S —	
	Z —	
	SAT —	



[Description]	Temporarily suspends the execution of instructions for the period specified in the SNZCFG register.
	Upon the lapse of the specified period or a state transition occurs, execution is automatically resumed starting at the next instruction.
	The SNOOZE state is released under the following conditions:
	• The specified period has elapsed.
	• A terminating-type exception occurs
	Even if the conditions (values of the PSW.ID and PSW.NP) for acknowledging the above exceptions are not satisfied, the SNOOZE state is released if there is an exception request. (Example: The SNOOZE state is released when an EIINT request occurs even when PSW.ID = 1.).
	The SNOOZE state is not released if the terminating-type exceptions are masked by the following mask functions:
	• Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller <sup>Note 1</sup> .
	• 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.
	<b>Note 1.</b> The SNOOZE state is released when the masking is carried out using only the ISPR , PLMR registers and PSW.EIMASK bit (Supported only when Architecture Identifier bit PID[31:24] = $07_H$ (RH850G4MH2)).
[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.



# 2.2.3.92 SST.B

<Store instruction>

SST.B	Short format store byte
	Storage of byte data
[Instruction format]	SST.B reg2, disp7[ep]
[Operation]	$adr \leftarrow ep + zero-extend (disp7)^{Note 1}$
	CheckException (MDP)
	Store-memory (adr, GR[reg2], Byte)
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0
	rrrr0111dddddd
[Flags]	СҮ —
	OV —
	s —
	Z —
	SAT —
[Description]	Adds the element pointer to the 7-bit displacement data, zero-extended to word length, to generate a 32-bit address and stores the data of the lowest byte of reg2 to the generated address.



# 2.2.3.93 SST.H

<Store instruction>

	Short format store halfword
SST.H	Storage of halfword data
[Instruction format]	SST.H reg2, disp8[ep]
[Operation]	$adr \leftarrow ep + zero-extend (disp8)^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	Store-memory (adr, GR[reg2], Halfword)
	<b>Note 1.</b> 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.
	addada is the higher 7 bits of dispo.
[Flags]	СҮ —
	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.

#### CAUTION

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.



#### 2.2.3.94 SST.W

<Store instruction>

CCT W	Short format store word
SST.W	Storage of word data
[Instruction format]	SST.W reg2, disp8[ep]
[Operation]	$adr \leftarrow ep + zero-extend (disp8)^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	Store-memory (adr, GR[reg2], Word)
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format IV
[Opcode]	
	15 0
	rrrr1010ddddd1
	ddddd is the higher 6 bits of disp8.
[Flags]	СҮ —
	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.
CAUTION	

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.


## 2.2.3.95 ST.B

<Store instruction>

	Store byte
ST.B	
	Storage of byte data
[Instruction format]	(1) ST.B reg2, disp16[reg1]
	(2) ST.B reg3, disp23[reg1]
	(3) ST.B reg3, [reg1]+
	(4) ST.B reg3, [reg1]-
[Operation]	<ul> <li>adr ← GR [reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg2], Byte)</li> </ul>
	<ul> <li>(2) adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup> CheckException (MDP)</li> <li>Store-memory (adr, GR[reg3], Byte)</li> </ul>
	<ul> <li>(3) adr ← GR[reg1]<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg3], Byte)</li> <li>GR[reg1] ← GR[reg1] + 1</li> </ul>
	(4) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MDP) Store-memory (adr, GR[reg3], Byte) $GR[reg1] \leftarrow GR[reg1] - 1$
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	<ol> <li>Format VII</li> <li>Format XIV</li> </ol>
	<ul><li>(2) Format XIV</li><li>(3) Format XI</li></ul>
	(4) Format XI



[Opcode]



Where RRRR = reg1, wwww = reg3.

dddddd is the lower 7 bits of disp23.

DDDDDDDDDDDDDDD is the higher 16 bits of disp23.



[Flags]

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 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, signextended to word length, to generate a 32-bit address and stores the lowest byte data of general-purpose register reg3 to the generated address.
- (3) Stores the byte data from the lowest byte field of the general-purpose register reg3 in the address that is generated from the word data in the general-purpose register reg1. Adds 1 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
- (4) Stores the byte data from the lowest byte field of the general-purpose register reg3 in the address that is generated from the word data in the general-purpose register reg1. Subtract 1 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

### 2.2.3.96 ST.DW

<Store instruction>

ST.DW	Store double-word
31.000	Storage of double-word data
[Instruction format]	ST.DW reg3, disp23[reg1]
[Operation]	adr $\leftarrow$ GR[reg1] + sign-extend (disp23) <sup>Note 1</sup>
	CheckException (MAE)
	CheckException (MDP)
	data $\leftarrow$ GR[reg3+1]    GR[reg3]
	Store-memory (adr, data, Double-word)
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	Format XIV
[Opcode]	
	15 0 31 16 47 32
	00000111101RRRRR www.wdddddd01111 DDDDDDDDDDDDDDDDDD
	Where RRRRR = reg1, wwwww = reg3. dddddd is the lower side bits 6 to 1 of disp23. DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.
[Flags]	СҮ —
	OV —
	S —
	Z —
	SAT —



[Description]	Adds the data of general-purpose register reg1 to a 23-bit displacement value sign-extended to word length to generate a 32-bit address. Double-word data consisting of the lower 32 bits of the word data of general-purpose register reg3 and the higher 32 bits of the word data of reg3 + 1 is then stored at this address.
[Supplement]	reg3 must be an even-numbered register. If an odd-numbered register is specified in reg3, bit 0 of the register number is ignored and the register is handled as an even-numbered register.

#### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. No misalignment exception will occur, however, if the address calculation results in a word boundary.



## 2.2.3.97 ST.H

<Store instruction>

оти	Store halfword
ST.H	Storage of halfword data
[Instruction format]	(1) ST.H reg2, disp16[reg1]
	(2) ST.H reg3, disp23[reg1]
	(3) ST.H reg3, [reg1]+
	(4) ST.H reg3, [reg1]–
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg2], Halfword)</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg3], Halfword)</li> </ul>
	(3) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MAE) CheckException (MDP) Store-memory (adr, GR[reg3], Halfword) GR[reg1] \leftarrow GR[reg1] + 2
	<ul> <li>(4) adr ← GR[reg1]<sup>Note 1</sup> CheckException (MAE) CheckException (MDP) Store-memory (adr, GR[reg3], Halfword) GR[reg1] ← GR[reg1] - 2</li> </ul>
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI



[Opcode]

	15 0 31 16
	(1) rrrr111011RRRRR ddddddddddddddd
	Where ddddddddddddd is the higher 15 bits of disp16.
	15 0 31 16 47 32
	(2) 00000111101RRRRR www.wdddddd01101 DDDDDDDDDDDDDDDDDDDD
	Where $RRRR = reg1$ , wwwww = reg3.
	dddddd is the lower side bits 6 to 1 of disp23.
	DDDDDDDDDDDDDD is the higher 16 bits of disp23.
	15 0 31 16
	(3) 00010111111RRRRR www.w01101110110
	15 0 31 16
	(4) 00100111111RRRRR wwww01101110110
[Flags]	СҮ —
	OV —
	S —
	Z —
	SAT —
[Description]	(1) 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 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, sign- extended to word length, to generate a 32-bit address and stores the lower halfword data of general-purpose register reg3 to the generated address.
	<ul> <li>(3) Stores the halfword data from the lower-order field of the general-purpose register reg3 in the address generated from the word data in the general-purpose register reg1.</li> <li>Adds 2 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.</li> </ul>
	<ul><li>(4) Stores the halfword data from the lower-order field of the general-purpose register reg3 in the address generated from the word data in the general-purpose register reg1. Subtract 2 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.</li></ul>

### CAUTION

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.



٦

## 2.2.3.98 ST.W

<Store instruction>

	Store word
ST.W	Storage of word data
[Instruction format]	(1) ST.W reg2, disp16[reg1]
	(2) ST.W reg3, disp23[reg1]
	(3) ST.W reg3, [reg1]+
	(4) ST.W reg3, $[reg1]$ -
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg2], Word)</li> </ul>
	<ul> <li>adr ← GR[reg1] + sign-extend (disp23)<sup>Note 1</sup></li> <li>CheckException (MAE)</li> <li>CheckException (MDP)</li> <li>Store-memory (adr, GR[reg3], Word)</li> </ul>
	(3) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MAE) CheckException (MDP) Store-memory (adr, GR[reg3], Word) GR[reg1] \leftarrow GR[reg1] + 4
	(4) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MAE) CheckException (MDP) Store-memory (adr, GR[reg3], Word) GR[reg1] $\leftarrow$ GR[reg1] – 4
	<b>Note 1.</b> An MAE or MDP exception might occur depending on the result of address calculation.
[Format]	(1) Format VII
	(2) Format XIV
	(3) Format XI
	(4) Format XI



[Opcode]

		15 0	31 16	
	(1)	rrrrr111011RRRRR	dddddddddddd	
	Whe	re ddddddddddd	dd is the higher 15 bi	ts of disp16.
		15 0	31 16	47 32
	(2)	00000111100RRRR	wwwwwdddddd01111	ססססססססססססססס
		re RRRRR = reg1, www		
	ddd	ddd is the lower side	bits 6 to 1 of disp23.	
	DDD	DDDDDDDDDDDD is	the higher 16 bits of d	lisp23.
		15 0	31 16	_
	(3)	00010111111RRRRR	wwww01101111010	
		15 0	31 16	1
	(4)	00100111111rrrrr	wwww01101111010	
[Flags]	CY	—		
	OV	_		
	S	—		

Ζ

SAT

\_\_\_\_

\_\_\_\_



[Description]	(1)	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 word data of general- purpose register reg2 to the generated 32-bit 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 word data of general- purpose register reg3 to the generated 32-bit address.
	(3)	Stores the word data from the general-purpose register reg3 in the address generated from the word data in the general-purpose register reg1. Adds 4 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.
	(4)	Stores the word data from t the general-purpose register reg3 in the address generated from the word data in the general-purpose register reg1. Subtract 4 from the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

### CAUTION

A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.



## 2.2.3.99 STC.B

	Store conditional byte
STC.B	Conditional storage when storais hats data manipulation is complete
	Conditional storage when atomic byte data manipulation is complete
[Instruction format]	STC.B reg3, [reg1]
[Operation]	$adr \leftarrow GR[reg1]^{Note 1}$
	CheckException (MDP)
	data $\leftarrow$ GR[reg3]
	token $\leftarrow$ LLbit <sup>Note 2</sup>
	if (token == 1)
	then Store-memory (adr, data, Byte)
	$GR[reg3] \leftarrow 1$
	else $GR[reg3] \leftarrow 0$
	LLbit $\leftarrow 0^{\text{Note 2}}$
	<b>Note 1.</b> An MDP exception may occur depending on the results of the address calculation.
	<b>Note 2.</b> For the link operation, see the hardware manual of the product used.
[Format]	Format VII
[Opcode]	
[opeode]	45 0.24 46
	15 0 31 16
	00000111111RRRRR wwww01101110010
[Flags]	СҮ —
	0V —
	S —
	Z —
	SAT —



[Description]This instruction can only be executed successfully if a link exists that corresponds to the<br/>specified address. If a corresponding link exists, the byte data of general-purpose register<br/>reg3 is stored in the memory and an atomic read-modify-write is executed.<br/>If the corresponding link has been lost, the data is not stored in the memory and execution<br/>of this instruction fails.Whether execution of the STC.B 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.B 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.BU instruction to ensure accurate updating of the memory in a multi-core system. The LDL.BU instruction and the STC.B instructions are intended always to be used in pair.

#### CAUTION

If a link is generated with the LDL.BU instruction and the STC.H or STC.W instruction is used instead of the STC.B instruction, the result will be a failure and the link be lost.



## 2.2.3.100 STC.H

	Store conditional halfword
STC.H	
	Conditional storage when atomic halfword data manipulation is complete
[Instruction format]	STC.H reg3, [reg1]
[Operation]	$adr \leftarrow GR[reg1]^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	data $\leftarrow$ GR[reg3]
	token $\leftarrow$ LLbit <sup>Note 2</sup>
	if (token $== 1$ )
	then Store-memory (adr, data, Halfword)
	$GR[reg3] \leftarrow 1$
	else $GR[reg3] \leftarrow 0$
	LLbit $\leftarrow 0^{\text{Note 2}}$
	<b>Note 1.</b> An MAE or MDP exception may occur depending on the results of the address calculation.
	<b>Note 2.</b> For the link operation, see the hardware manual of the product used.
[Format]	Format VII
[Opcode]	
	15 0 31 16
	00000111111RRRRR wwww01101110110
[Flags]	СҮ —
	ov —
	S —
	Z —
	SAT —



[Description]This instruction can only be executed successfully if a link exists that corresponds to the<br/>specified address. If a corresponding link exists, the halfword data of general-purpose<br/>register reg3 is stored in the memory and an atomic read-modify-write is executed.<br/>If the corresponding link has been lost, the data is not stored in the memory and execution<br/>of this instruction fails.<br/>Whether execution of the STC.H instruction has succeeded or not can be ascertained by

Whether execution of the STC.H 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.H 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.HU instruction to ensure accurate updating of the memory in a multi-core system. The LDL.HU instruction and the STC.H instructions are intended always to be used in pair.

### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. If a link is generated with the LDL.HU instruction and the STC.B or STC.W instruction is used instead of the STC.H instruction, the result will be a failure and the link be lost.



## 2.2.3.101 STC.W

<Store instruction>

	Store conditional word
STC.W	Conditional storage when atomic word data manipulation is complete
[Instruction format]	STC.W reg3, [reg1]
[Operation]	$adr \leftarrow GR[reg1]^{Note 1}$
	CheckException (MAE)
	CheckException (MDP)
	data $\leftarrow$ GR[reg3]
	token $\leftarrow$ LLbit <sup>Note 2</sup>
	if (token $== 1$ )
	then Store-memory (adr, data, Word)
	$GR[reg3] \leftarrow 1$
	else $GR[reg3] \leftarrow 0$
	LLbit $\leftarrow 0^{\text{Note 2}}$
	<b>Note 1.</b> An MAE or MDP exception may occur depending on the results of the address calculation.
	<b>Note 2.</b> For the link operation, see the hardware manual of the product used.
[Format]	Format VII
[Opcode]	
	15 0 31 16
	00000111111RRRRR wwww01101111010
[Flags]	СҮ —
-	OV —
	S —
	Z —
	SAT —



[Description]	<ul><li>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.</li><li>If the corresponding link has been lost, the data is not stored in the memory and execution of this instruction fails.</li></ul>		
	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. The LDL.W instruction and the STC.W instructions are intended always to be used in pair.		
[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.		

#### CAUTIONS

1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.

2. If a link is generated with the LDL.W instruction and the STC.B or STC.H instruction is used instead of the STC.W instruction, the result will be a failure and the link be lost.



### 2.2.3.102 STM.MP

<Special instruction>

Store Multiple MPU entries to memory

**STM.NP** (Supported only when Architecture Identifier bit PID[31:24] = 07<sub>H</sub> (RH850G4MH2)) Store MPU entries

```
[Instruction format]
                                 STM.MP eh-et, [reg1]
[Operation]
                                 if (PSW.UM==0)
                                 then
                                       if ( eh \leq et )
                                       then
                                             cur ← eh
                                             end \leftarrow et
                                             tmp \leftarrow reg1
                                             while (cur \leq end) {
                                                   adr \leftarrow tmp^{Note \ 1, \ Note \ 2}
                                                   CheckException(MDP)
                                                   Store-memory (adr, MPLA[cur], Word)
                                                   tmp \leftarrow tmp + 4
                                                   adr \leftarrow tmp^{Note \; 1, \; Note \; 2}
                                                   CheckException(MDP)
                                                   Store-memory (adr, MPUA[cur], Word)
                                                   tmp \leftarrow tmp + 4
                                                   adr \leftarrow tmp^{Note \ 1, \ Note \ 2}
                                                   CheckException(MDP)
                                                   Store-memory (adr, MPAT[cur], Word)
                                                   tmp \leftarrow tmp + 4
                                                   \operatorname{cur} \leftarrow \operatorname{cur} + 1
                                             }
                                       else
                                 else
                                 Note 1.
                                              The lower 2 bits of adr are masked by 0.
                                 Note 2.
                                              An MDP exception may occur as a result of address calculation.
[Format]
                                 Format XI
```



### [Opcode]

	15 0 31 16
	rrrr111111RRRRRwwww00101100100
	rrrrr indicates eh.
	wwwww indicates et.
	RRRRR indicates reg1.
[Flags]	СҮ —
	ov —
	S —
	Z —
	SAT —
[Descriptions]	The word data of the MPU protection area setting system registers (MPLA, MPUA, and MPAT) is stored to the address generated from the word data of the general-purpose register reg1 according to the specified order. Word size is added to the address each time the word data of the system register is stored. The contents of these system registers is processed in ascending order, regardless of the value of MPIDX, from the entry number indicated by eh to that indicated by et (eh, eh+1, eh+2,, et). The bank specified by MPBK is only to be processed.
	Because it is an SV privilege instruction, a PIE exception will occur if it is executed when PSW.UM is set (1).
[Supplement]	This instruction stores the target MPU protection area setting directly to memory. This instruction can perform the operation more effectively than by specifying the entry via MPIDX, reading the value of system register into a general-purpose register by STSR instructions, and storing it to memory by ST.W instructions.
	The lower 2-bit address generated from the general-purpose register reg1 is masked by 0 and aligned on a word boundary. The general-purpose register reg1 retains the original value after the instruction execution is complete.

This instruction is an SV privilege instruction.



#### CAUTION

When an exception or an interrupt occurs during instruction execution and even if the contents of all system registers has not been stored to the memory, instruction execution can be aborted and exceptions or interrupts can be accepted, as long as the acceptance condition is satisfied. When the execution is suspended, it is impossible to know the contents of which system registers has been stored to the memory. After the return from exception processing, the suspended STM. MP instruction can be precisely re-executed as long as resources related to execution of the STM.MP instruction are not changed during exception processing, for the return PC from an exception is considered to be the PC of STM.MP instruction. This instruction re-execution restarts the STM.MP instruction processing from the start.



## 2.2.3.103 STSR

<Store instruction>

	Store contents of system register
STSR	
	Storage of contents of system register
[Instruction format]	STSR regID, reg2, selID
[Instruction format]	STSR regID, reg2
[Operation]	$GR[reg2] \leftarrow SR[regID, selID]^{Note 1}$
	<b>Note 1.</b> An exception might occur depending on the access permission. For details, see the hardware manual of the product used.
[Format]	Format IX
[Opcode]	
	15 0 31 16
	rrrr111111RRRRRssss00001000000
	rrrrr: reg2, sssss: selID, RRRRR: regID
	11111. ltg2, 55555. 5011D, KKKKK. ltg1D
[Flags]	СҮ —
	OV —
	s —
	Z —
	SAT —
[Description]	Stores the system register contents specified by the system register number and selection ID (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 the hardware manual of the product used.

### CAUTION

The system register number or selection ID is a unique number used to identify each system register. How to access undefined registers is described in the hardware manual of the product used, but accessing undefined registers is not recommended.



# 2.2.3.104 SUB

<Arithmetic instruction>

SUB		Subtract
		Subtraction
[Instruction format]	SUB reg	1, reg2
[Operation]	GR[reg2]	$] \leftarrow GR[reg2] - GR[reg1]$
[Format]	Format I	
[Opcode]	15	0
	rrrr0(	)1101RRRRR
[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]	purpose 1	the word data of general-purpose register reg1 from the word data of general- register reg2 and stores the result in general-purpose register reg2. General-purpose eg1 is not affected.



# 2.2.3.105 SUBR

<Arithmetic instruction>

SUBR		Subtract reverse
		Reverse subtraction
[Instruction format]	SUBR reg1, reg2	
[Operation]	$GR[reg2] \leftarrow GR[reg1] - GR[reg2]$	
[Format]	Format I	
[Opcode]		
	15 0 rrrr001100RRRR	
[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]	Subtracts the word data of general-purpose register reg2 from the word data purpose register reg1 and stores the result in general-purpose register reg2 register reg1 is not affected.	-



## 2.2.3.106 SWITCH

SWITCH	Jump with table look
[Instruction format]	SWITCH reg1
[Operation]	$adr \leftarrow (PC + 2) + (GR[reg1] logically shift left by 1)^{Note 1}$ CheckException (MDP) $PC \leftarrow (PC + 2) + (sign-extend (Load-memory (adr, Halfword)))) logically shift left by 1Note 1. An MDP exception might occur depending on the result of address calculation.$
[Format]	Format I
[Opcode]	$\frac{15 \qquad 0}{0000000010RRRR}$ RRRRR $\neq 00000$ (Do not specify r0 for reg1.)
[Flags]	CY       —         OV       —         S       —         Z       —         SAT       —



[Description] The following steps are taken

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

<4> Jumps to the target address generated in step <3>.

generate a 32-bit target address.

#### CAUTIONS

- 1. Do not specify r0 for reg1.
- 2. In the SWITCH instruction memory read operation executed in order to read the table, memory protection is performed.
- 3. When an exception occurs during memory access, the instruction execution is aborted after the end of the read cycle. An interrupt might be accepted after the end of the read cycle.



## 2.2.3.107 SXB

<Data manipulation instruction>

SXB		Sign extend byte
		Sign-extension of byte data
[Instruction format]	SXB reg1	
[Operation]	$GR[reg1] \leftarrow sign-extend (GR[reg1] (7:0))$	
[Format]	Format I	
[Opcode]		
	15 0 0000000101RRRRR	
[Flags]	СҮ —	
	ov —	
	s —	
	Z —	
	SAT —	

[Description] Sign-extends the lowest byte of general-purpose register reg1 to word length.



## 2.2.3.108 SXH

<Data manipulation instruction>

SXH	Sign extend halfword
	Sign-extension of halfword data
[Instruction format]	SXH reg1
[Operation]	$GR[reg1] \leftarrow sign-extend (GR[reg1] (15:0))$
[Format]	Format I
[Opcode]	15 0 0000000111RRRRR
[Flags]	CY       —         OV       —         S       —         Z       —         SAT       —
[Description]	Sign-extends the lower halfword of general-purpose register reg1 to word length.



### 2.2.3.109 SYNCE

<Special instruction>

SYNCE	Synchronize except	tions
•	Exception synchronization instruct	ction
[Instruction format]	SYNCE	
[Operation]	No operation is performed.	
[Format]	Format I	
[Opcode]		
	15 0 00000000011101	
[Flags]	СҮ —	
	ov —	
	s —	
	Z —	
	SAT —	
[Description]	Performs no specific processing that involves synchronization processing. When the	

[Description]Performs no specific processing that involves synchronization processing. When the<br/>execution of the SYNCE instruction is completed, the PC proceeds to the next instruction.<br/>However, an interrupt can be acknowledged.



## 2.2.3.110 SYNCI

SYNCI	Synchronize instruction fetch
	Instruction fetch synchronization instruction
[Instruction format]	SYNCI
[Operation]	Performs instruction fetch synchronization processing.
[Format]	Format I
[Opcode]	
	15 0 00000000011100
[Flags]	CY —
	OV —
	S —
	Z —
	SAT —
[Description]	Discards unexecuted instructions in the CPU, and re-fetches the subsequent instructions. Prior to the instruction execution, waits for the completion of the execution of the preceding instruction that is being executed, and the completion of the cache operation and load processing which are executed independently of the instruction execution. The SYNCI instruction does not wait for the completion of store processing.
[Supplement]	For details on synchronization processing, see the hardware manual of the product used.
	To implement "Self-modifying code" which rewrites instructions in memory, it is also necessary to disable the instruction cache.



# 2.2.3.111 SYNCM

SYNCM	Synchronize memory
<b>U</b> I I U III	Memory synchronize instruction
[Instruction format]	SYNCM
[Operation]	Performs memory access synchronization processing.
[Format]	Format I
[Opcode]	
	15 0 00000000011110
[Flags]	CY —
	OV —
	S —
	Z —
	SAT —
[Description]	Prior to the instruction execution, waits for the completion of execution of all preceding instructions that are being executed, and the completion of the store and load processing which are executed independently of the instruction execution. The execution of the SYNCM instruction makes the master devices in the system ready for referencing the results of memory accesses that precede the SYNCM instruction within the scope of the store processing for which the SYNCM instruction can wait. The SYNCM instruction does not wait for the completion of cache operation.
[Supplement]	For details on synchronization processing, see the hardware manual of the product used. For the scope of store processing for which the SYNCM instruction can wait, see the hardware manual of the product used.



## 2.2.3.112 SYNCP

SYNCP	Synchronize pipeline
JINCF	Pipeline synchronize instruction
[Instruction format]	SYNCP
[Operation]	Performs pipeline synchronization processing.
[Format]	Format I
[Opcode]	
	15 0 0000000011111
[Flags]	СҮ —
	ov —
	S —
	Z —
	SAT —
[Description]	Prior to the instruction execution, waits for the completion of execution of all preceding instructions that are being executed. The SYNCP instruction waits for the completion of load processing that is executed independently of the instruction execution. This guarantees that the load data is stored in a general-purpose register. The SYNCP instruction does not wait for the completion of store processing and cache operation which are executed independently of the instruction which are executed independently of the instruction which are executed independently of the instruction executed.
[Supplement]	For details on synchronization processing, see the hardware manual of the product used.



# 2.2.3.113 SYSCALL

SYSCALL	System call
STOCALL	System call exception
[Instruction format]	SYSCALL vector8
[Operation]	$tmp \leftarrow PSW$
	$PSW.UM \leftarrow 0$
	$PSW.EP \leftarrow 1$
	PSW.ID ← 1 if (vector8 <= SCCFG.SIZE) is satisfied
	then adr $\leftarrow$ SCBP + zero-extend (vector8 logically shift left by 2) <sup>Note 3</sup>
	else adr $\leftarrow$ SCBP <sup>Note 3</sup>
	CheckException (MDP) <sup>Note 2</sup>
	$EIPC \leftarrow PC + 4 \text{ (return PC)}$
	$EIPSW \leftarrow tmp$
	EIIC $\leftarrow$ exception cause code <sup>Note 1</sup>
	$PC \leftarrow SCBP + Load-memory (adr, Word)$
	<ul><li>Note 1. See the hardware manual of the product used.</li><li>Note 2. When an exception occurs, the PSW before execution stored in tmp is saved.</li><li>Note 3. An MDP exception might occur depending on the result of address calculation.</li></ul>
[Format]	Format X
[Opcode]	
	15 0 31 16
	11010111111vvvvv000vv00101100000
	Where VVV is the higher 3 bits of vector8 and vvvvv is the lower 5 bits of vector8.
[Flags]	СҮ —
	ov —
	s —
	Z —
	SAT —



[Description] Calls OS's system services. <1> 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. <2> Confirms whether an exception is detected for the address generated in step <1>. Saves the contents of the return PC (address of the instruction next to the <3> SYSCALL instruction) and PSW to EIPC and EIPSW. <4> Stores the exception cause code corresponding to vector8 in the EIIC register. The exception cause code is the value of vector8 plus  $8000_{\text{H}}$ . Updates the PSW according to the exception causes listed in the hardware <5> manual of the product used. Loads the word of the address generated in <1>. <6> <7> Generates a 32-bit target address by adding the value of the SCBP register to the data in <6>. <8> Branches to the target address generated in step <7>.

#### CAUTIONS

- 1. This instruction is dedicated to call OS's system services. For the procedure to use it in a user program, refer to the functional specifications for your OS.
- 2. The memory reads for table lookup during the SYSCALL instruction are subject to memory protection with the supervisor privilege.
- 3. When an exception occurs during memory access, the instruction execution is aborted after the end of the read cycle. An interrupt might be accepted after the end of the read cycle.



# 2.2.3.114 TRAP

TRAP		Тгар
		Software exception
[Instruction format]	TRAP vector5	
[Operation]	EIPC $\leftarrow$ PC + 4 (return PC)	
	$EIPSW \leftarrow PSW$	
	EIIC $\leftarrow$ exception cause code <sup>Note 1</sup>	
	$PSW.UM \leftarrow 0$	
	$PSW.EP \leftarrow 1$	
	$\text{PSW.ID} \leftarrow 1$	
	$PC \leftarrow exception handler address^{Note 1}$	
	<b>Note 1.</b> See the hardware manual of the product used.	
[Format]	Format X	
[Opcode]		
	15 0 31 16 00000111111vvvvv00000000000000	
	vvvvv = vector5	
[Flags]	СҮ —	
	ov —	
	S —	
	Z —	
	SAT —	



[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 the hardware manual of the product used.

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 the hardware manual of the product used.

vector5	Exception Cause Code	Offset Address
00 <sub>H</sub>	0000 0040 <sub>H</sub>	40 <sub>H</sub>
01 <sub>H</sub>	0000 0041 <sub>H</sub>	
0F <sub>H</sub>	0000 004F <sub>H</sub>	
10 <sub>H</sub>	0000 0050 <sub>H</sub>	50 <sub>H</sub>
11 <sub>H</sub>	0000 0051 <sub>H</sub>	
1F <sub>H</sub>	0000 005F <sub>H</sub>	



# 2.2.3.115 TST

<Logical instruction>

TST		Test
		Test
[Instruction format]	TST reg	1, reg2
[Operation]	result ←	- GR[reg2] AND GR[reg1]
[Format]	Format 1	ι
[Opcode]		
	15 rrrrr0	0 01011RRRRR
[Flags]	СҮ	_
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" if the operation result is "0"; otherwise, 0.
	SAT	_
[Description]	register	ne word data of general-purpose register reg2 with the word data of general-purpose reg1. The result is not stored with only the flags being changed. General-purpose reg1 and reg2 are not affected.



## 2.2.3.116 TST1

<Bit manipulation instruction>

TOTA	Test bit
TST1	Bit test
[Instruction format]	<ol> <li>(1) TST1 bit#3, disp16[reg1]</li> <li>(2) TST1 reg2, [reg1]</li> </ol>
[Operation]	<ul> <li>adr ← GR[reg1] + sign-extend (disp16)<sup>Note 1</sup></li> <li>CheckException (MDP)</li> <li>token ← Load-memory (adr, Byte)</li> <li>Z flag ← Not (extract-bit (token, bit#3))</li> </ul>
	(2) $adr \leftarrow GR[reg1]^{Note 1}$ CheckException (MDP) token $\leftarrow$ Load-memory (adr, Byte) Z flag $\leftarrow$ Not (extract-bit (token, reg2))
	<b>Note 1.</b> An MDP exception might occur depending on the result of address calculation.
[Format]	<ol> <li>Format VIII</li> <li>Format IX</li> </ol>
[Opcode]	
	15     0     31     16       (1)     11bbb111110RRRRR     dddddddddddddddddddddddddddddddd
	15     0     31     16       (2)     rrrrlllllrrrrr     000000011100110
[Flags]	CY – OV – S –
	<ul> <li>Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1".</li> <li>SAT —</li> </ul>


[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 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;

(2) Reads the word data of general-purpose register register register register as 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.



### 2.2.3.117 XOR

<Logical instruction>

XOR		Exclusive OF
		Exclusive OF
[Instruction format]	XOR re	g1, reg2
[Operation]	GR[reg	2] ← GR[reg2] XOR GR[reg1]
[Format]	Format	Ι
[Opcode]		
	15 rrrrr(	0 001001RRRRR
[Flags]	CY	_
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" if the operation result is "0"; otherwise, "0".
	SAT	
[Description]	general	vely ORs the word data of general-purpose register reg2 with the word data of -purpose register reg1 and stores the result in general-purpose register reg2. - purpose register reg1 is not affected.



### 2.2.3.118 XORI

<Logical instruction>

XORI		Exclusive OR immediate (16-bit)
		Exclusive OR immediate
[Instruction format]	XORI i	mm16, reg1, reg2
[Operation]	GR[reg	2] ← GR[reg1] XOR zero-extend (imm16)
[Format]	Format	VI
[Opcode]		
	15 rrrrr1	0 31 16 110101RRRRR iiiiiiiiiiiiiiii
[Flags]	СҮ	
	OV	0
	S	"1" if operation result word data MSB is "1"; otherwise, "0".
	Z	"1" if the operation result is "0"; otherwise, "0".
	SAT	—
[Description]		vely ORs the word data of general-purpose register reg1 with the 16-bit immediate ro-extended to word length, and stores the result in general-purpose register reg2.

General-purpose register reg1 is not affected.



### 2.2.3.119 ZXB

<Data manipulation instruction>

ZXB	Zero extend byte
	Zero-extension of byte data
[Instruction format]	ZXB reg1
[Operation]	$GR[reg1] \leftarrow zero-extend (GR[reg1] (7:0))$
[Format]	Format I
[Opcode]	15 0 0000000100RRRR
[Flags]	CY       —         OV       —         S       —         Z       —         SAT       —
[Description]	Zero-extends the lowest byte of general-purpose register reg1 to word length.



### 2.2.3.120 ZXH

<Data manipulation instruction>

ZXH		Zero extend halfword
		Zero-extension of halfword data
[Instruction format]	ZXH reg1	
[Operation]	$GR[reg1] \leftarrow zero-extend (GR[reg1] (15:0))$	
[Format]	Format I	
[Opcode]		
	15 0 0000000110RRRR	
[Flags]	СҮ —	
	ov —	
	s —	
	Z —	
	SAT —	

[Description] Zero-extends the lower halfword of general-purpose register reg1 to word length.



# 2.3 Cache Instructions

### 2.3.1 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

### 2.3.2 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.



### 2.3.2.1 CACHE

<Cache instruction>

CACHE	Cache
	Cache operation
[Instruction format]	CACHE cacheop, [reg1]
[Operation]	Manipulates the cache specified by cacheop.
[Format]	Format X
[Opcode]	
	15         0         31         16           111pp111111RRRRR         PPPPP00101100000
	ppPPPP indicates cacheop.
[Flags]	СҮ —
	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 the hardware manual of the product used.
[Supplement]	Each cache operation has its own instruction execution privilege. For details about the correspondence between cache operations and instruction execution privileges, see the hardware manual of the product used.
	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 the hardware manual of the product used.
	<b>Table 2.7</b> shows the cache operation of each CACHE instruction and the corresponding cacheop. If a cacheop which is not defined in the table is specified, no operation is performed, and also no memory protection is performed like a NOP instruction.



Table 2.7 Cache Operation

cacheop	Target	Processing	Cache Specification	Operation
	Instruction	CHBII	Address	(Cache Hit Block Invalidate, Instruction cache)
				If the specified address hits an address in the instruction cache, the corresponding cache line is disabled.
				The lock is released if the cache line is locked.
				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.
				The lock is released if the cache line is locked.
				This instruction can be used in cases such as when the entire 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 of the specified index.
				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.
				For details, see the hardware manual of the product used.
1111110	_	CLL	_	(Clear Load Link)
				When this cacheop is specified, the instruction is not a CACHE instruction, but it is the CLL instruction. The privilege check and memory protection for the CACHE instruction are not performed in this case.



### 2.3.2.2 PREF

<Cache instruction>

PREF	Prefetch
	Prefetch
[Instruction format]	PREF prefop, [reg1]
[Operation]	Executes the prefetch operation specified by prefop.
[Format]	Format X
[Opcode]	
	15 0 31 16
	1101111111RRRRR PPPPP00101100000
	PPPPP indicates prefop.
[Flags]	СҮ —
	OV —
	s —
	Z —
	SAT —
[Description]	Executes the prefetch operation specified by prefop on the word data of general-purpose register reg1 used as a 32-bit address.
[Supplement]	Even if the MPU detects a memory protection violation during the prefetch operation, no MDP exception occurs. In such a case, the prefetch operation is canceled and no operation is performed like a NOP instruction.
	If the cache is disabled, the instruction performs the prefetch operation specified by prefop. <b>Table 2.8</b> shows the prefetch operation of the PREF instruction and the corresponding prefop. If a prefop which is not defined in the table is specified, no operation is performed like a NOP instruction.



#### CAUTION

Be aware that even after the prefetch instruction has finished executing, a prefetch operation might not necessarily have been performed.

#### Table 2.8 Prefetch Operation

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.

#### CAUTIONS

1. The size of data that is prefetched by a single prefetch operation is the cache line size of the instruction cache. For details, see the hardware manual of the product used.

2. If the data at the specified address is already stored in the instruction cache, only the LRU information of the cache line is updated.



# 2.4 Floating-Point Instructions

### 2.4.1 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 sub- opcode field, three fields that specify general-purpose registers, a 3-bit category field, and a 2- bit type field.





### 2.4.2 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)
- MINF.S: Floating-point Minimum (Single)
- MULF.D: Floating-point Multiply (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.DL: Floating-point Convert Double to Long, round toward positive (Double)
- 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 Word (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 Word (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 Word, 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 Word, 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)

- ROUNDF.DL: Floating-point Convert Double to Long, round to nearest (Double)
- ROUNDF.DW: Floating-point Convert Double to Word, round to nearest (Double)
- ROUNDF.DUL: Floating-point Convert Double to Unsigned-Long, round to nearest (Double)
- ROUNDF.DUW: Floating-point Convert Double to Unsigned-Word, round to nearest (Double)
- ROUNDF.SL: Floating-point Convert Single to Long, round to nearest (Single)
- ROUNDF.SW: Floating-point Convert Single to Word, round to nearest (Single)
- ROUNDF.SUL: Floating-point Convert Single to Unsigned-Long, round to nearest (Single)
- ROUNDF.SUW: Floating-point Convert Single to Unsigned-Word, round to nearest (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 Word, 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 Word, 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)



### 2.4.3 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 2.9** 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 2.10** 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.

Mnemonic	Definition	Inverted	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

 Table 2.9
 List of Conditions for Comparison Instructions



				Bit Definition o	f Condition Code f	cond(3:0)	
Mnemonic	Conditio	n Code fcond	Less than	Equal to	Unordered	Invalid operation exception occurs when unordered	Inverted Logic
(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 2.10 Definitions of Condition Code Bits and Their Logical Inversions



### 2.4.4 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 2.11**).
- Operation: Indicates the function of the instruction. (symbols are listed in **Table 2.12**).
- Format: Indicates the instruction format (see Section 2.4.1, Instruction Formats).
- Opcode: Indicates the instruction opcode in bit fields (symbols are listed in **Table 2.13**).
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.

#### Table 2.11 Instruction 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 2.4.3, Conditions for Comparison Instructions).



Table 2.12	Operations
Symbol	Explanation
~	Assignment (input for)
GR [ <i>a</i> ]	Value stored in general-purpose register a
SR [ <i>a</i> , <i>b</i> ]	Value stored in system register (RegID = $a$ , SeIID = $b$ )
result	Result is reflected in flag
==	Comparison (true upon a match)
+	Add
-	Subtract
П	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 ( <i>a</i> , <i>b</i> , <i>c</i> )	Result of multiplying <i>a</i> and <i>b</i> and then adding <i>c</i>
fms ( <i>a</i> , <i>b</i> , <i>c</i> )	Result of multiplying a and b and then subtracting c
Table 2.13	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
1	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



#### 2.4.4.1 ABSF.D

<Floating-point instruction>

ABSF.D	Floating-point Absolute Value (Double)
	Floating-point absolute value (double precision)
[Instruction format]	ABSF.D reg2, reg3
[Operation]	$reg3 \leftarrow abs (reg2)$
[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 0 w w w w 0 1 0 0 0 1 0 1
	reg2 reg3 category type sub-op
[Description]	This instruction takes the absolute value from the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores it in the register pair specified by 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.



### 2.4.4.2 ABSF.S

ABSF.S	Floating-point Absolute Value (Single)
	Floating-point absolute value (single precision)
[Instruction format]	ABSF.S reg2, reg3
[Operation]	$reg3 \leftarrow abs (reg2)$
[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 r 1 1 1 1 1 1 0 0 0 0 0 w w w w w 1 0 0 0 1 0 0 1 0 0 0
	reg2 reg3 category type sub-op
[Description]	This instruction takes the absolute value from the single-precision floating-point format contents of general-purpose register reg2, and stores it 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.



### 2.4.4.3 ADDF.D

ADDF.D																	Floa	iting-p	oint	Add	(Doul	ble)
															Floa	ting-j	oint	t add	(doı	ıble p	recisi	ion)
[Instruction format]	ADDF.D reg1	, reg2	, reg	3																		
[Operation]	$reg3 \leftarrow reg2 +$	⊦ reg1																				
[Format]	Format F: I																					
[Opcode]																						
	15 11	10			5	4			0	31			27	26	25	2	32	2 21	20		17	16
	rrrr 0	1 1	1 1	1	1	R	R R	R	0	W	w	w w	0	1	0	0	) 1	1 1	1	0 0	0 0	0
	reg2						reg	1			re	eg3			cat	egor	/	type		sub-c	р	
[Description]	This instruction specified by g contents of the in the register if it were of in rounding mod	enera e regis pair s finite	l-purj ster p pecif	pose air s fied	reg pec by g	gist cifie gen	er re ed by eral-	g1 v / ger ·purj	vitl ner pos	h th al-j se r	ne d purj egi:	oub pose ster	le-p reş reg	orec gisto 3. T	isic er r 'he	on fl eg2 ope	oat an rati	ing-j id sto ion i	poi ore s ez	nt fo s the xecu	rma rest	at ult
[Floating-point operation exceptions]	Unimplemente Invalid operat Inexact except Overflow exce Underflow exce	ion ex tion (l eption	(O)	ion (	-	otio	n (E	)														



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		P						
+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.



### 2.4.4.4 ADDF.S

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 r r r r 1 1 1 1 1 1 R R R R R w w w w w 1 0 0 0 1 1 0 0 0 0 0
	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)



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		P						
+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.



## 2.4.4.5 CEILF.DL

	Floating-point Convert Double to Long, round toward positive (Double											
CEILF.DL	Conversion to fixed-point format (double precision											
[Instruction format]	CEILF.DL reg2, reg3											
[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											
	r r r r 0       1       1       1       1       0       0       1       0       w w w w 0       1       0       0       0       1       0											
[Description]	This instruction arithmetically converts the double-precision floating-point format content 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 $+\infty$ 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 $2^{63} - 1$ to $-2^{63}$ , 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.											
	follows, according to differences among sources.											
	<ul> <li>Source is a positive number or +∞: 2<sup>63</sup> – 1 is returned.</li> </ul>											



[Floating-point operation exceptions]

Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)

[Operation result]

reg2 (A)	+Normal	-Normal	+0	-0	+∞	-∞	S-NaN	
Operation result [exception]	A (Int	teger)	0 (Int	teger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.6 CEILF.DUL

<Floating-point instruction>

CEILF.DUL	-		Flo	atin	g-poir	nt Con	iver	t Dou	uble	to Un	sign	ed-L	.ong	, rounc	l towar	d po	sitiv	e (Dou	ıble)
							С	onve	ersic	on to u	nsig	ned	fixeo	d-point	format	(do	uble	precis	sion)
[Instruction format]	CEILF.DUL	reg2, reg3																	
[Operation]	reg3 ← ceil re	reg3 $\leftarrow$ ceil reg2 (double $\rightarrow$ unsigned long-word)																	
[Format]	Format F: I	Format F: I																	
[Opcode]																			
	15 11	10	5	4			0	31			27	26	25	23	22 21	20		1	7 16
	rrrr 0	1 1 1 1	1 1	1	0 (	) 1	0	w	W	w w	0	1	0	0 0	1 0	1	0	1 0	0
	reg2								r	eg3			cat	egory	type		sub	-ор	
[Description]	This instruction of the register format, and st	pair specifi	ed by	ge	neral	l-pur	ро	se r	egi	ster	eg/	2 to	un	signe	ed 64-	bit	fix	ed-p	oint
	The result is r	ounded in th	ie +∞	diı	rectio	on re	ga	rdle	SS	of th	e cı	ırre	nt 1	ound	ling n	nod	e.		
	When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} - 1$ to 0, an IEEE754-defined invalid operation exception is detected.																		
	If invalid oper register is set follows, accor	as an invalid	l oper	ati	on ai	nd no	o e	xcej	-										s
	• Source is a	a positive nu	mber	out	tside	the 1	ran	ige (	of 2	$2^{64} -$	1 to	0,	or -	+∞: 2	$2^{64} - 1$	is	retu	irnec	l.

• Source is a negative number, not-a-number, or  $-\infty$ : 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	teger)	Max U-Int[V]		0[V]	

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.7 CEILF.DUW

<Floating-point instruction>

Floating-point Convert Double to Unsigned-Word, round toward positive (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	CEILF.DUW reg2, reg3
[Operation]	reg3 $\leftarrow$ ceil reg2 (double $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

15	5				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	1	0	0	1	0	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
		I	reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-ор		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to unsigned 32-bit fixed-point<br/>format, and stores the result in general-purpose register reg3.

The result is rounded in the  $+\infty$  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  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 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.



### 2.4.4.8 CEILF.DW

CEILF.DW					Fle	oatin	ig-poi	int C	onv	ert D	ouble	to V	Vorc	l, roui	nd	towa	rd p	osit	tive (	Doul	ble)
										Con	/ersic	on to	fixe	d-poii	nt f	orma	t (de	oub	ole pr	ecisi	ion)
[Instruction format]	CEILF.DW re	eg2, reg3																			
[Operation]	reg3 ← ceil re	eg2 (double -	→ wo	rd	)																
[Format]	Format F: I																				
[Opcode]																					
	15 11	10	5	4			0	31			27	26	25	2	3	22 2	1 2	0		17	16
	rrrr 0	1 1 1 1	1 1	0	0	0 1	0	w	W	w	ww	1	0	0 (	C	1 0	) 1	L (	0 0	0	0
	reg2									reg3			ca	tegory	/	type		s	ub-op	)	
[Description]	This instruction the register particular the stores the results	ir specified b	y gei	ner	al-p	urpo	ose	regi	•					• •							
	The result is r	ounded in the	; +∞	dir	ectio	on r	egaı	dle	ss	of tł	ie cu	ırre	nt r	oun	diı	ng n	ıod	le.			
	When the sour range of $2^{31}$ –	1																		ide	the
	If invalid oper register is set follows, accor	as an invalid	opera	atic	on ar	nd n	io ey	cep	-												

- Source is a positive number or  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  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 (Int	teger)	0 (Int	teger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.9 CEILF.SL

CEILF.SL	Floating-point Convert Single to Long, round toward positive (Single)
	Conversion to fixed-point format (single precision)
[Instruction format]	CEILF.SL reg2, reg3
[Operation]	$reg3 \leftarrow ceil reg2 (single \rightarrow long-word)$
[Format]	Format F: I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16
	rrrr111111000100wwww010001000100
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the single-precision floating-point format contents of 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 $+\infty$ 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 $2^{63} - 1$ to $-2^{63}$ , 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 $+\infty$ : $2^{63} - 1$ is returned.
	• Source is a negative number, not-a-number, or $-\infty$ : $-2^{63}$ 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 (Int	teger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.10 CEILF.SUL

CEILF.SUL	Floating-point Convert Single to Unsigned-Long, round toward positive (Si	ngle)
	Conversion to unsigned fixed-point format (single preci	sion)
[Instruction format]	CEILF.SUL reg2, reg3	
[Operation]	reg3 $\leftarrow$ ceil reg2 (single $\rightarrow$ unsigned long-word)	
[Format]	Format F: I	
[Opcode]		
	<u>15 11 10 5 4 0 31 27 26 25 23 22 21 20 1</u>	7 16
	rrrr 1 1 1 1 1 1 1 1 0 0 1 0 w w w w 0 1 0 0 0 1 0 0 0 1 0	0 0
	reg2 reg3 category type sub-op	
[Description]	This instruction arithmetically converts the single-precision floating-point format contens specified by general-purpose register reg2 to unsigned 64-bit fixed-point format, and stort the result in the register pair specified by general-purpose register reg3. The result is rounded in the $+\infty$ direction regardless of the current rounding mode. When the source operand is infinite, not-a-number, or negative number, or when the rour result is outside the range of $2^{64} - 1$ to 0, an IEEE754-defined invalid operation exception	res
	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.	5
	• Source is a positive number outside the range of $2^{64} - 1$ to 0, or $+\infty$ : $2^{64} - 1$ is returned	d.
	• Source is a negative number, not-a-number, or $-\infty$ : 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.


Floating-point Convert Single to Unsigned-Word, round toward positive (Single)

Conversion to unsigned fixed-point format (single precision)

### 2.4.4.11 CEILF.SUW

<Floating-point instruction>

CEILF.SUW										
[Instruction format]	CEILF.SUW reg2, reg3									

[Operation]  $\operatorname{reg3} \leftarrow \operatorname{ceil} \operatorname{reg2} (\operatorname{single} \rightarrow \operatorname{unsigned} \operatorname{word})$ 

[Format] Format F: I

[Opcode]

15	5				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	0	0	1	0	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
		I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	-ор		$\overline{ }$

[Description]This instruction arithmetically converts the single-precision floating-point format contents<br/>specified by general-purpose register reg2 to unsigned 32-bit fixed-point format, and stores<br/>the result in general-purpose register reg3.

The result is rounded in the  $+\infty$  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  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



# 2.4.4.12 CEILF.SW

CEILF.SW	Floa	ating-point	t Convert Single	e to W	/ord, rour	id towa	rd po	ositive (	Single
			Conversio	on to f	fixed-poir	it forma	at (sii	ngle pre	ecision
[Instruction format]	CEILF.SW reg2, reg3								
[Operation]	$reg3 \leftarrow ceil reg2 (single \rightarrow word)$								
[Format]	Format F: I								
[Opcode]									
	15 11 10 5 4	0 31	27	26 2	25 23	22 21	1 20		17 1
	r r r r r r 1 1 1 1 1 1 0 0 0	1 0 w	www	1	0 0 0	1 0	0	0 0	0 0
	reg2		reg3		category	type		sub-op	
[Description]	This instruction arithmetically converts the general-purpose register reg2 to 32-bit fix purpose register reg3.	-	-		• •				
	The result is rounded in the $+\infty$ direction	regardle	ess of the cu	rren	t round	ing m	ode		
	When the source operand is infinite or not range of $2^{31} - 1$ to $-2^{31}$ , an IEEE754-defi								de th
	If invalid operation exceptions are not enarregister is set as an invalid operation and a	no excej	ption occurs						
	follows, according to differences among s	sources.							
	<ul> <li>Source is a positive number or +∞: 2<sup>31</sup></li> </ul>								



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	teger)	0 (Int	teger)	+Max Int[V]		-Max Int[V]		

Note 1. [ ] indicates an exception that must occur.



# 2.4.4.13 CMOVF.D

. . . ----...

<pre><floating-point instruction=""></floating-point></pre>	Floating-point Conditional Move (Double)
CMOVF.D	Conditional move (double precision)
[Instruction format]	CMOVF.D fcbit, reg1, reg2, reg3
[Operation]	if (FPSR.CCn == 1) then
	$reg3 \leftarrow reg1$
	else
	$reg3 \leftarrow reg2$
	endif
	Remark: n = fcbit
[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 R R R R 0 w w w w 0 1 0 0 0 0 1 f f f 0
	reg2 reg1 reg3*1 category type sub-op
	Remark: fcbit: fff Note 1. reg3: wwww != 0
	wwww ≠ 0000 (do not set reg3 to r0)
[Description]	When the CC(7:0) bits of the FPSR register specified by fcbit in the opcode are true (1), data from the register pair specified by reg1 is stored in the register pair specified by reg3. When these bits are false (0), data from the register pair specified by reg2 is stored in the register pair specified by 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.
CAUTION	
Do not set reg3 to r0.	



# 2.4.4.14 CMOVF.S

<floating-point condition="" institution<="" p=""></floating-point>							Floa	ating-p	oint (	Conditio	nal N	Nove (\$	Singl	e)
CMOVF.S								Cond	itiona	al move	(sin	gle pre	cisio	n)
[Instruction format]	CMOVF.S fcbit, re	g1, reg2, reg	3											
[Operation]	if (FPSR.CCn == 1	) then												
	$reg3 \leftarrow reg1$													
	else													
	$reg3 \leftarrow reg2$													
	endif													
	Remark: n = fcbit													
[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	R R	RRR	w	w w w w	1	0 0	0	0 0	0	f f	f	0
	reg2		r	eg1		reg3*1		categ	ory	type	5	sub-op		
	Remark: fcbit: fff Note 1. reg3: www wwwww ≠ 00	ww != 0 0000 (do not s	et reg3	to r0)										
[Description]	When the CC(7:0) data from reg1 is st reg3.			-	-	•			-					
[Floating-point operation exceptions]	None													
[Supplement]	A subnormal input	will not be f	lushed	even if	fthe	e FS bit of t	he ]	FPSR	reg	gister	is 1.			
CAUTION														
Do not set reg3 to r0.														



# 2.4.4.15 CMPF.D

CMPF.D	Compare floating-point values(Double)
CIVIFF.D	Floating-point comparison (double precision)
[Instruction format]	CMPF.D fcond, reg2, reg1, fcbit
	CMPF.D fcond, reg2, reg1
[Operation]	if isNaN(reg1) or isNaN(reg2) then result.less ← 0 result.equal ← 0 result.unordered ← 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 ← (fcond[2] & result.less)   (fcond[1] & result.equal)   (fcond[0] & result.unordered)
	Remark: n = fcbit
[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 R R R R 0 0 F F F F 1 0 0 0 0 1 1 f f f 0
	reg2 reg1 category type sub-op



[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 2.14, 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 a relationship of two numerical values and triggers an error when an unordered result is detected.

Comparis Condition				Detection of Invalid Operation Exception by
	fcond	Definition	Description	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	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 2.14 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
                                              0
                                TRFSR
                                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 < r14
                         L2:
                         # Enter code for processing when r12 < r14</pre>
                                 :
                         # When using a comparison to detect Q-NaN
                                              LT,r12,r14,0
                                                                   \# Check if r12 < r14
                                CMPF.D
                                TRFSR
                                              0
                                                                   # If true, go to L2
                                ΒT
                                              L2
                         # Enter code for processing when not r12 < r14</pre>
                         L2:
                         # Enter code for processing when r12 < r14</pre>
                                 :
[Floating-point operation
                         Invalid operation exception (V)
exceptions]
[Supplement]
                         A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.
```



#### [Operation result]

#### [Condition code (fcond) = 0 to 7]

reg1(B) reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN		
±Normal										
±0		Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)								
±∞	uic	the comparison condition (icond) in the PPSR.CCh Dit (n = icDit)								
Q-NaN		Unorderd								
S-NaN		Unorderd[V]								

#### [Condition code (fcond) = 8 to 15]

reg1(B) reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN		
±Normal										
±0		Stores result of comparison (true or false) executed under the comparison condition (fcond) in the EPSR CCo bit (n = fcbit)								
±∞	10	the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)								
Q-NaN		Unorderd[V]								
S-NaN				Unord	eld[v]					

Note: [ ] indicates an exception that must occur.



# 2.4.4.16 CMPF.S

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 ← 0 result.equal ← 0
	result.unordered ← 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 ← (fcond[2] & result.less)   (fcond[1] & result.equal)   (fcond[0] & result.unordered)
	Remark: n: fcbit
[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 R R R R R 0 F F F F 1 0 0 0 0 1 0 f f f 0
	reg2 reg1 category type sub-op



[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 2.15, 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 a relationship of two numerical values and triggers an error when an unordered result is detected.

Comparia Condition		-		Detection of Invalid Operation Exception by
	fcond	Definition	Description	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	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 2.15 Comparison Conditions

Note: ?: Unordered (invalid comparison)



```
# When explicitly testing Q-NaN
                                CMPF.S
                                              OLT, r12, r14, 0 # Check if r12 < r14
                                              UN,r12,r14,1
                                                                   # Check if Unordered
                                CMPF.S
                                              0
                                TRFSR
                               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 < r14
                         L2:
                         # Enter code for processing when r12 < r14</pre>
                                 :
                         # When using a comparison to detect Q-NaN
                                              LT,r12,r14,0
                                                                   \# Check if r12 < r14
                                CMPF.S
                                TRFSR
                                              0
                                                                   # If true, go to L2
                                ΒT
                                              L2
                         # Enter code for processing when not r12 < r14</pre>
                         L2:
                         # Enter code for processing when r12 < r14</pre>
                                 :
[Floating-point operation
                         Invalid operation exception (V)
exceptions]
[Supplement]
                         A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.
```



#### [Operation result]

#### [Condition code (fcond) = 0 to 7]

reg1(B) reg2(A)	+Normal	Q-NaN	S-NaN											
±Normal														
±0	Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)													
±∞	uie	the comparison condition (rcond) in the PPSR.CCh bit (n = rcbit)												
Q-NaN	Unorderd													
S-NaN	Unorderd[V]													

#### [Condition code (fcond) = 8 to 15]

reg1(B) reg2(A)	+Normal	-Normal	-∞	Q-NaN	S-NaN								
±Normal													
±0		Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit) Unorderd[V]											
±∞	110												
Q-NaN													
S-NaN													

Note: [ ] indicates an exception that must occur.



# 2.4.4.17 CVTF.DL

CVTF.DL	Floating-point Convert Double to Long (Double
CVIT.DL	Conversion to fixed-point format (double precision
[Instruction format]	CVTF.DL reg2, reg3
[Operation]	$reg3 \leftarrow cvt reg2 (double \rightarrow long-word)$
[Format]	Format F: I
[Opcode]	
	15         11         10         5         4         0         31         27         26         25         23         22         21         20         17
	r r r r 0 1 1 1 1 1 1 0 0 1 0 0 w w w w 0 1 0 0 0 1 0 1
	reg2 reg3 category type sub-op
[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, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg3.
	When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to $-2^{63}$ , 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 $+\infty$ : $2^{63} - 1$ is returned.
	• Source is a negative number, not-a-number, or $-\infty$ : $-2^{63}$ is returned.



Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)

[Operation result]

reg	j2(A)	+Normal	-Normal	+0 -0		+∞	-∞	Q-NaN	S-NaN
re	eration esult eption]	A (Int	teger)	0 (Int	eger)	+Max Int[V]		–Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.18 CVTF.DS

<Floating-point instruction>

CVTF.DS	Floating-point Convert Double to Single(Double
	Conversion to floating-point format (double precision
[Instruction format]	CVTF.DS reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (double $\rightarrow$ single)
[Format]	Format F: I
[Opcode]	
	15         11         10         5         4         0         31         27         26         25         23         22         21         20         17         1
	r r r r 0 1 1 1 1 1 1 0 0 0 1 1 w w w w w 1 0 0 0 1 0 1
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by 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	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Overflow exception (O)
	Underflow exception (U)

[Operation result]

reg2(A	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception	A (S	ngle)	+0	-0	+∞	-∞	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.19 CVTF.DUL

**CVTF.DUL** 

<Floating-point instruction>

Floating-point Convert Double to Unsigned-Long (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	CVTF.DUL reg	g2, reg3									
[Operation]	reg3 ← cvt reg2	eg3 $\leftarrow$ cvt reg2 (double $\rightarrow$ unsigned long-word)									
[Format]	Format F: I	ormat F: I									
[Opcode]											
	15 11 1	10 5	4 0	31 27	26 25 23	22 21	20 1	7 16			
	rrrr 0 1	1 1 1 1 1 1	1 0 1 0 0	wwww0	1 0 0 0	1 0	1 0 1	0 0			
	reg2			reg3	category	type	sub-op				
[Description]	the register pair format, in accore	n arithmetically co r specified by ger rdance with the co neral-purpose reg	neral-purpose r urrent rounding	register reg2 to	unsigned 6	4-bit fi	ixed-poin	t			
		the range of $2^{64}$		-							
	register is set as	tion exceptions a s an invalid opera ling to differences	ation and no ex	ception occurs				3			
	• Source is a p	positive number of	outside the ran	ge of $2^{64} - 1$ to	0, or +∞: 2	$2^{64} - 1$	is returne	d.			
	• Source is a n	negative number,	not-a-number	, or $-\infty$ : 0 is ret	urned.						



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.



#### 2.4.4.20 CVTF.DUW

**CVTF.DUW** 

<Floating-point instruction>

Floating-point Convert Double to Unsigned-Word (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	CVTF.DUW reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2(double $\rightarrow$ word)

[Format] Format F: I

[Opcode]

15					11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
r	ľ	C	r	r	0	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
		r	eg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-op		$\left  \right $

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to unsigned 32-bit fixed-point<br/>format, and stores the result in general-purpose register reg3.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



### 2.4.4.21 CVTF.DW

CVTF.DW													Flo	ating	J-poi	nt Cc	nve	rt D	oub	le to	Wor	d (D	oub	le)
												C	onve	ersio	n to	fixed	poir	nt fo	rma	at (do	uble	e pre	cisio	on
[Instruction format]	CVTF.DW r	eg2	, reg3	ł																				
[Operation]	reg3 ← cvt r	$reg3 \leftarrow cvt reg2 (double \rightarrow word)$																						
[Format]	Format F: I																							
[Opcode]																								
	15 1	11 10	C		5	4				0	31			27	26	25	2	32	22	21 20	)		17	1
	rrrr	0 1	. 1 3	. 1	1 1	0	0	1	0	0	W	wv	v w	w	1	0 (	) (	) 1	L (	0 1	0	0	0	(
	reg2											re	g3			cate	gory	/	type	•	sub	o-op		
[Description]	This instruct	ion	orithr	action	llvo	01	Nor	to th	n d	011	bla	nra	oisi	ont	٦٥٥	tina	nc	int	fo	rmo	tao	nto	nto	
[Description]	This instruct the register p stores the res When the sou range of 2 <sup>31</sup> - If invalid ope register is set follows, acco • Source is • Source is	oair sult urce – 1 t erati t as ordin a po	speci in gen to $-2$ ion ex an in ng to ositiv	fied b neral- and is <sup>31</sup> , an ccepti valid differ e num	y gen purp infin IEEI ons a opera ence	ner os nit E7 are ati- s a	ral-j e re $\frac{1}{2}$ e on $\frac{1}{2}$ 54- e no on $\frac{1}{2}$ on $\frac{1}{2}$ amo $+\infty$	purj gist no defi t en and ng s : 2 <sup>3</sup>	pos er r t-a-: inec able no sou:	e r eg nu l in ed rce 1 i	egi 3. mb nva , th cep es. s re	er, c lid c e pr otior	reg or w open esen oc	2 to hen catic cvat	the on e ion s. T	bit rou xce bit (	fixe nd ptic (bit etu	ed ed on i	poi res is d of	int f ult i letec the	orm s ou cted	nat, utsi I. SR	an de 1	Ċ
	the register p stores the res When the sor range of 2 <sup>31</sup> - If invalid ope register is set follows, acco • Source is • Source is	pair : sult i urce - 1 t erati t as ordin a po a ne	speci in ger oper to $-2$ ion $e_2$ an in ng to ositiv egativ	fied b heral- and is <sup>31</sup> , an cepti valid differ e num	y ger purp i infin IEEI ons a opera ence hber nber,	ner os nit E7 are ati- s a or	ral-j e re 54- e no on a amo + $\infty$ ot-a	purj gist no defi t en and ng : : 2 <sup>3</sup>	pos er r t-a-: inec able no sou:	e r eg nu l in ed rce 1 i	egi 3. mb nva , th cep es. s re	er, c lid c e pr otior	reg or w open esen oc	2 to hen catic cvat	the on e ion s. T	bit rou xce bit (	fixe nd ptic (bit etu	ed ed on i	poi res is d of	int f ult i letec the	orm s ou cted	nat, utsi I. SR	an de 1	Ċ
[Description] [Floating-point operation exceptions]	the register p stores the res When the sourange of 2 <sup>31</sup> - If invalid oper register is set follows, acco	pair : sult i urce – 1 t erati t as prdin a po a no hted	speci in ger c oper to – 2 ion ey an in ng to ositiv egativ oper	fied b leral-j and is and is 31, an ccepti valid differ e num e num	y gen purp i infin IEEI ons a opera ence hber nber,	ner oso nit E7 are ations a or no pti	ral-j e re 54- e no on a amo + $\infty$ ot-a	purj gist no defi t en and ng : : 2 <sup>3</sup>	pos er r t-a-: inec able no sou:	e r eg nu l in ed rce 1 i	egi 3. mb nva , th cep es. s re	er, c lid c e pr otior	reg or w open esen oc	2 to hen catic cvat	the on e ion s. T	bit rou xce bit (	fixe nd ptic (bit etu	ed ed on i	poi res is d of	int f ult i letec the	orm s ou cted	nat, utsi I. SR	an de 1	d



[Operation result]

reg2(A)	+Normal	ormal −Normal +0 −0 +∞		-∞	S-NaN			
Operation result [exception]	A (Int	eger)			+Max Int[V]		-Max Int[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 the hardware manual of the product used.

### 2.4.4.22 CVTF.HS

<Floating-point instruction>

CVTF.HS	Floating-point Convert Half to Single (Single)								
	Conversion to floating-point format (single precision)								
[Instruction format]	CVTF.HS reg2, reg3								
[Operation]	reg3 $\leftarrow$ cvt reg2 (half $\rightarrow$ single)								
[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 0 0 0 1 0 w w w w w 1 0 0 0 1 0 0 0 0								
	reg2 reg3 category type sub-op								
[Description]	This instruction arithmetically converts the half-precision floating-point format contents in the lower 16 bits of general-purpose register reg2 to single-precision floating-point format, rounding the result in accordance with the current rounding mode, and stores the result in general-purpose register reg3.								
[Floating-point operation exceptions]	Invalid operation exception (V)								
[Supplement]	With the exception of not-a-number values, all half-precision floating-point format values can be accurately converted into single-precision floating-point format values. A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.								

[Operation result]

reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Si	ngle)	+0	-0	+∞	-∞	Q-NaN	Q-NaN[V]

Note: [ ] indicates an exception that must occur.



# 2.4.4.23 CVTF.LD

CVTF.LD	Floating-point Convert Long to Double (Double)
	Conversion to floating-point format (double precision)
[Instruction format]	CVTF.LD reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (long-word $\rightarrow$ double)
[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 arithmetically converts the 64-bit fixed-point format contents of the register pair specified by general-purpose register reg2 to double-precision floating-point format in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg3.
[Floating-point operation exceptions]	Inexact exception (I)
[Operation result]	

reg2(A)	+Integer	+Integer -Integer				
Operation result [exception]	A (No	ormal)	+0			



# 2.4.4.24 CVTF.LS

CVTF.LS	Floating-point Convert Long to Single (Single)
	Conversion to floating-point format (single precision)
[Instruction format]	CVTF.LS reg2, reg3
[Operation]	$reg3 \leftarrow cvt reg2 (long-word \rightarrow single)$
[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 1 0 0 0 1 0 0 0 1 0
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the 64-bit fixed-point format contents of the register pair specified by 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)
Operation result [exception]	A (No	ormal)	+0



# 2.4.4.25 CVTF.SD

<Floating-point instruction>

[Operation]       reg3 $\leftarrow$ cvt reg2 (single $\rightarrow$ double)         [Format]       Format F: I         [Opcode]       15       11       10       5       4       0       3 $\boxed{r \ r \ r \ r \ r \ 1}$ 1       1       1       1       0       0       1       0       v	Conversion to floating-point format (double precision           31         27         26         25         23         22         21         20         17
[Operation]       reg3 $\leftarrow$ cvt reg2 (single $\rightarrow$ double)         [Format]       Format F: I         [Opcode]       15       11       10       5       4       0       3 $[x \ x \ x \ x \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ $	31 27 26 25 23 22 21 20 17
[Format]Format F: I[Opcode] $15$ $11$ $10$ $5$ $4$ $0$ $3$ $r \ r \ r \ r \ r \ r \ 1 \ 1 \ 1 \ 1 \ $	31 27 26 25 23 22 21 20 17
[Opcode] $15$ $15$ $11$ $10$ $5$ $4$ $0$ $11$	31 27 26 25 23 22 21 20 17
[Description] 15   11   10   5   4   0   3   15   11   10   1   1   1   1   1   1   0   0	31 27 26 25 23 22 21 20 17
r r r r r 1 1 1 1 1 1 0 0 0 1 0 v         reg2         [Description]         This instruction arithmetically converts the sing of general-purpose register reg2 to double-precisivith the current rounding mode, and stores the regulation of the stores of the store	31 27 26 25 23 22 21 20 17
reg2         [Description]         This instruction arithmetically converts the sing of general-purpose register reg2 to double-precisivith the current rounding mode, and stores the precision of general-purpose register reg2 to double-precisivith the current rounding mode, and stores the precision of general-purpose register reg2 to double-precisivity of general-purpose reg1 to double-precisivity of general-purpose reg1	
[Description] This instruction arithmetically converts the sing of general-purpose register reg2 to double-preci with the current rounding mode, and stores the	w w w w 0 1 0 0 0 1 0 1 0 1
of general-purpose register reg2 to double-preci with the current rounding mode, and stores the	reg3 category type sub-op
	cision floating-point format, in accordance
[Floating-point operation Unimplemented operation exception (E)	
exceptions] Invalid operation exception (V)	
Inexact exception (I)	
[Operation result]	

reg2(A)	+Normal	Normal -Normal		-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Do	ouble)	+0	-0	+∞	-∞	Q-NaN	Q-NaN[V]

Note 1.  $\ \ [ \ \ ]$  indicates an exception that must occur.



# 2.4.4.26 CVTF.SL

CVTF.SL	Floating-point Convert Single to Long (Single									
	Conversion to fixed-point format (single precision									
[Instruction format]	CVTF.SL reg2, reg3									
[Operation]	$reg3 \leftarrow cvt reg2 (single \rightarrow long-word)$									
[Format]	Format F: I									
[Opcode]										
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 1									
	r r r r r 1 1 1 1 1 1 0 0 1 0 0 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 64-bit fixed-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg3.									
	When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to $-2^{63}$ , 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 $+\infty$ : $2^{63} - 1$ is returned.									
	• Source is a negative number, not-a-number, or $-\infty$ : $-2^{63}$ is returned.									



Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)

[Operation result]

reg2	2(A)	+Normal	-Normal	+0 -0 +∞		-∞	Q-NaN	S-NaN	
res	ation sult ption]	A (Int	teger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.27 CVTF.SH

<Floating-point instruction>

CVTF.SH	Floating-point Convert Single to Half (Single
	Conversion to half-precision floating-point format (single precision
[Instruction format]	CVTF.SH reg2, reg3
[Operation]	reg3 $\leftarrow$ zero-extend (cvt reg2 (single $\rightarrow$ half))
[Format]	Format F: I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 <sup>7</sup>
	rrrr111111000111 w w w w w 1 0 0 0 1 0 0 0 0
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the single-precision floating-point format contents in general-purpose register reg2 to half-precision floating-point format, rounding the resul in accordance with the current rounding mode. The result is zero-extended to word length and stored in general-purpose register reg3.
[Floating-point operation	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Overflow exception (O)
	Underflow exception (U)

[Operation result]

reg2	(A) +N	Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Opera rest [excep	ult	A (H	lalf)	+0	-0	+∞	-∞	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



# 2.4.4.28 CVTF.SUL

**CVTF.SUL** 

<Floating-point instruction>

Floating-point Convert Single to Unsigned-Long (Single)

Conversion to unsigned fixed-point format (single precision)

[Instruction format]	CVI	ſF.S	SUL	. reş	g2,	reg.	3																						
[Operation]	reg3	reg3 $\leftarrow$ cvt reg2 (single $\rightarrow$ unsigned long-word)																											
[Format]	Form	nat	F: I																										
[Opcode]																													
	15			11	10				5	4				0	31				27	26	25	2	3 22	2 21	20		1	7	16
	rı	c r	r	r	1	1 1	. 1	1	1	1	0	1	0	0	w	W	W	w	0	1	0 0	) (	1	. 0	0	0	1 (	C	0
		reg	g2														reg	3			cate	gory	t	ype		sub	-ор		
[Description]	This gene curre regis	eral ent	-pur roui	pos ndir	e re	egis	ter 1	reg	2 to	un	nsig	gne	d 6	4-t	oit f	fix	ed-j	poi	nt f	orr	nat,	in a	icco	orda	anc	e w	rith t	he	;
	When result determined	lt is	out			-												-											
	If in regis follo	ster	is s	et a	s ar	ı inv	vali	d oj	pera	atic	on	and	l no	) ex	cej		-											5	
	• S	oui	rce i	s a j	pos	itiv	e nı	ımł	ber	out	tsic	le t	he	ran	ge	of	264	-	l to	0,	or +	$\infty$ :	2 <sup>64</sup>	- 1	is	retu	irne	d.	
	• S	our	rce i	s a 1	neg	ativ	e n	um	ber,	, nc	ot-a	a-ni	ım	ber	, or	ſ —	xo: (	) is	ret	urr	ed.								



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.



### 2.4.4.29 CVTF.SUW

**CVTF.SUW** 

<Floating-point instruction>

Floating-point Convert Single to Unsigned-Word (Single)

Conversion to unsigned fixed-point format (single precision)

[Instruction format]	CVTF.SUW reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (single $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
r	2	r	r	r	r	1	1	1	1	1	1	1	0	1	0	0	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
		I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	o-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg3.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



### 2.4.4.30 CVTF.SW

<Floating-point instruction>

**CVTF.SW** 

Floating-point Convert Single to Word (Single)

Conversion to fixed-point format (single precision)

[Instruction format]	CVTF.SW reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (single $\rightarrow$ word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
I	r	r	r	r	r	1	1	1	1	1	1	0	0	1	0	0	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
			reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	o-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to 32-bit fixed-point format, and stores the result in general-purpose register reg3.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of  $2^{31} - 1$  to  $-2^{31}$ , 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  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  is returned.



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	teger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.


# 2.4.4.31 CVTF.ULD

<floating-point instruction=""></floating-point>	
CVTF.ULD	Floating-point Convert Unsigned-Long to Double (Double
	Conversion to floating-point format (double precision
[Instruction format]	CVTF.ULD reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (unsigned long-word $\rightarrow$ double)
[Format]	Format F: I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 1
	r r r r 0 1 1 1 1 1 1 1 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 arithmetically converts the unsigned 64-bit fixed-point format contents of the register pair specified by general-purpose register reg2 to double-precision floating-point format in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg3.
[Floating-point operation exceptions]	Inexact exception (I)
[Operation result]	

reg2(A)	+Integer	+Integer -Integer							
Operation result [exception]	A (No	ormal)	+0						



# 2.4.4.32 CVTF.ULS

**CVTF.ULS** 

<Floating-point instruction>

Floating-point Convert	Unsigned-Long to	Single (Single)
r loading point convort	Children Long to	

Conversion to floating-point format (single precision)

[Instruction format]	CVTF.ULS re	g2, reg	g3																				
[Operation]	reg3 ← cvt re	g2 (un	signed	long	-wo	ord –	→ si	ngl	le)														
[Format]	Format F: I																						
[Opcode]																							
	15 11	10		5	4			0	31				27	26	25	2	32	22 21	20		17	7 16	i
	rrrr 0	1 1	1 1	1 1	1	0 0	0	1	W	W	W	w	w	1	0	0 0	) [	1 0	0	0	0 1	. 0	
	reg2										reg3				cat	tegory	,   .	type	1	sub-o	эр		Ī
[Description]	This instruction the register particular format, and structure with	ir spec ores th	ified b e resul	by ge It in §	nera gene	al-pu eral-j	ırpo purț	se 1 DOS	reg	ist	er r	eg2	to	si	ngl	e-pr	eci	isior	n flo	oatin	ıg-p		t
[Floating-point operation exceptions]	Inexact excep	tion (I)	)																				
[Operation result]																							

reg2(A)	+Integer	+Integer -Integer							
Operation result [exception]	A (No	ormal)	+0						



## 2.4.4.33 CVTF.UWD

<Floating-point instruction>

	Floating-point Convert Unsigned-Word to Double (Double)
CVTF.UWD	Conversion to floating-point format (double precision
[Instruction format]	CVTF.UWD reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (unsigned word $\rightarrow$ double)
[Format]	Format F: I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 10
	r r r r r     1     1     1     1     0     0     0     w w w w 0     1     0     0     0     1     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0 </td
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the unsigned 32-bit fixed-point format contents of general-purpose register reg2 to double-precision floating-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg3.
	This conversion operation is performed accurately, without any loss of precision.
[Floating-point operation exceptions]	None

reg2(A)	+Integer	-Integer	0 (Integer)				
Operation result [exception]	A (No	ormal)	+0				



16 0

#### **CVTF.UWS** 2.4.4.34

**CVTF.UWS** 

<Floating-point instruction>

Floating-point Convert Unsigned-Word to Single (Single)

Conversion to floating-point format (single precision)

[Instruction format]	CVT	F.UV	VS r	eg2,	, reg	3																					
[Operation]	reg3	← cv	vt re	g2 (1	unsi	gne	d w	ore	1 —	→ S	ing	le)															
[Format]	Form	nat F:	Ι																								
[Opcode]																											
	15		11	10				5	4				0	31				27	26	25	23	22	2 21	20		17	16
	r r	rı	r r	1	1 1	. 1	1	1	1	0	0	0	0	w	W	W	w	w	1	0 0	0 0	1	0	0	0 (	0 1	0
		reg2													r	eg3				cate	gory	ty	/pe	İ	sub-c	р	Ì
[Description]	gene in ge	instru ral-pu eneral ding 1	ırpo -pur	se re pose	egist	ter 1	eg2	to	sir	ıgl	e-p	rec	isi	on	floa	atir	ıg-	poi	nt	forn	ıat,	and	d st	ore	s the	e res	
[Floating-point operation exceptions]	Inexa	act ex	cep	tion	(I)																						

reg2(A)	+Integer	-Integer -Integer							
Operation result [exception]	A (No	ormal)	+0						



## 2.4.4.35 CVTF.WD

<Floating-point instruction>

CVTF.WD	Floating-point Convert Word to Double (Double)
	Conversion to floating-point format (double precision
[Instruction format]	CVTF.WD reg2, reg3
[Operation]	reg3 $\leftarrow$ cvt reg2 (word $\rightarrow$ double)
[Format]	Format F: I
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 1
	rrrr11111100000wwww01000101010010
	reg2 reg3 category type sub-op
[Description]	This instruction arithmetically converts the 32-bit fixed-point format contents of general- purpose register reg2 to double-precision floating-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general- purpose register reg3.
	This conversion operation is performed accurately, without any loss of precision.
[Floating-point operation exceptions]	None

reg2(A)	+Integer	-Integer	0 (Integer)				
Operation result [exception]	A (No	ormal)	+0				



## 2.4.4.36 CVTF.WS

CVTF.WS	Floating-point Convert Word to Single (single
	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]	
	15 11 10 5 4 0 31 27 26 25 23 22 21 20 17
	r r r r r 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
	reg2 reg3 category type sub-op
[Description]	reg2reg3categorytypesub-opThis 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)
Operation result [exception]	A (No	ormal)	+0



## 2.4.4.37 DIVF.D

<Floating-point instruction>

	Floating-point Divide (Double)												
DIVF.D	Floating-point division (double precision)												
[Instruction format]	DIVF.D reg1, reg2, reg3												
[Operation]	$reg3 \leftarrow reg2 \div reg1$												
[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 R R R R 0 w w w 0 1 0 0 0 1 1 1 1 1 1 0												
	reg2 reg1 reg3 category type sub-op												
[Description]	This instruction divides double-precision floating-point format contents of the register pair specified by general-purpose register reg2 by the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, 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.												
[Floating-point operation	Unimplemented operation exception (E)												
exceptions]	Invalid operation exception (V)												
	Inexact exception (I)												
	Division by zero exception (Z)												
	Overflow exception (O)												
	Underflow exception (U)												



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal		P			+∞	-∞		
-Normal		B	÷A		-∞	+∞		
+0			0.14	- N II) (1	+∞	-∞		
-0	±∝	<sup>,</sup> [Z]	Q-Na	aN[V]	-∞	+∞		
+∞	+0	-0	+0	-0	0.14			
-∞	-0	+0	-0	+0	Q-Na	an[v]		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



# 2.4.4.38 DIVF.S

<Floating-point instruction>

DIVERS         [Instruction format]       DIVF.S reg1, reg2, reg3         [Operation]       reg3 $\leftarrow$ reg2 $\div$ reg1         [Format]       Format F: I         [Opcode] $15  11  10  5  4  0  31  27  26  25  23  22  21  20  17  16  1  1  1  1  1  1  R  R  R  R$	DIVF.S	Floating-point Divide (Single)													
[Operation] $\operatorname{reg3} \leftarrow \operatorname{reg2} \div \operatorname{reg1}$ [Format]       Format F: I         [Opcode] $15$ $11$ $10$ $5$ $4$ $0$ $31$ $27$ $26$ $25$ $23$ $22$ $21$ $20$ $17$ $16$ [Opcode] $15$ $11$ $11$ $11$ $11$ $11$ $10$ $0$ $0$ $11$ $0$ $11$ $11$ $10$ $11$ $11$ $10$ $11$ $10$ $11$ $10$ $11$ $10$ $11$ $10$ $11$ $10$ $11$	DIVF.5	Floating-point division (single precision)													
[Format]       Format F: I         [Opcode] $15$ $11$ $10$ $5$ $4$ $0$ $31$ $27$ $26$ $25$ $23$ $22$ $21$ $20$ $17$ $16$ $\boxed{x \ r \ r \ r \ r \ 1}$ $1$	[Instruction format]	DIVF.S reg1, reg2, reg3													
[Opcode] $15$ $11$ $10$ $5$ $4$ $0$ $31$ $27$ $26$ $25$ $23$ $22$ $21$ $20$ $17$ $16$ $x \ r \ r \ r \ r \ r \ r \ r \ r \ r \ $	[Operation]	$reg3 \leftarrow reg2 \div reg1$													
$\begin{bmatrix} 15 & 11 & 10 & 5 & 4 & 0 & 31 & 27 & 26 & 25 & 23 & 22 & 21 & 20 & 17 & 16 \\ \hline r & r & r & r & r & 1 & 1 & 1 & 1 & 1 &$	[Format]	Format F: I													
r r r r r $r r r r$ $r r r r$ $r r r r$ $r r r r r r$ $r r r r r r r r r r r r r r r r r r r$	[Opcode]														
[Description]This instruction divides the single-precision floating-point format contents of general-purpose register reg2 by the single-precision floating-point format contents of general-purpose register reg1, 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) Division by zero exception (Z)		15 11 10 5 4 0 31 27 26 25 23 22 21 20 17 16													
[Description]This instruction divides the single-precision floating-point format contents of general- purpose register reg2 by the single-precision floating-point format contents of general- purpose register reg1, 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) Division by zero exception (Z)		r r r r r         1 1 1 1 1 1 R R R R R         w w w w w 1         0 0 0         1 1         0 1 1 1         0													
purpose register reg2 by the single-precision floating-point format contents of general- purpose register reg1, 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) Division by zero exception (Z)		reg2 reg1 reg3 category type sub-op													
exceptions] Invalid operation exception (V) Inexact exception (I) Division by zero exception (Z)	[Description]	purpose register reg2 by the single-precision floating-point format contents of general- purpose register reg1, 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													
Inexact exception (I) Division by zero exception (Z)		Unimplemented operation exception (E)													
Division by zero exception (Z)	exceptions]	Invalid operation exception (V)													
		Inexact exception (I)													
Overflow exception (O)		Division by zero exception (Z)													
		Overflow exception (O)													
Underflow exception (U)		Underflow exception (U)													



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-8		
-Normal		B	÷A		-∞	+∞		
+0		.[7]			+∞	-8		
-0	±∞	·[∠]	Q-Na	aN[V]	-∞	+∞		
+∞	+0	-0	+0	-0	Q-Na			
-∞	-0	+0	-0	+0	Q-Na	antvi		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



### 2.4.4.39 FLOORF.DL

**FLOORF.DL** 

<Floating-point instruction>

Floating-point Convert Double to Long, round toward negative (Double)

Conversion to fixed-point format (double precision)

[Instruction format]	FLOORF.DL reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (double $\rightarrow$ long-word)

Format F: I

[Format]

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
1	r	r	r	r	0	1	1	1	1	1	1	0	0	0	1	1	w	w	W	w	0	1	0	0	0	1	0	1	0	1	0	0
Ī			reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	o-op		

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to 64-bit fixed-point format, and<br/>stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{63} - 1$  to  $-2^{63}$ , an IEEE754-defined invalid operation exception is detected.

- Source is a positive number or  $+\infty$ :  $2^{63} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{63}$  is returned.



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	teger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.40 FLOORF.DUL

<Floating-point instruction>

FLOORF.DUL

Floating-point Convert Double to Unsigned-Long, round toward negative (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	FLOORF.DUL reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (double $\rightarrow$ 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
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to unsigned 64-bit fixed-point<br/>format, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{64} - 1$  to 0, an IEEE754-defined invalid operation exception is detected.

- Source is a positive number outside the range of  $2^{64} 1$  to 0, or  $+\infty$ :  $2^{64} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.41 FLOORF.DUW

<Floating-point instruction>

FLOORF.DUW

Floating-point Convert Double to Unsigned-Word, round toward negative (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	FLOORF.DUW reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (double $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
ı	2	r	r	r	0	1	1	1	1	1	1	1	0	0	1	1	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
			reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to unsigned 32-bit fixed-point<br/>format, and stores the result in general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{32} - 1$  to 0, an IEEE754-defined invalid operation exception is detected.

- Source is a positive number outside the range of  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.42 FLOORF.DW

**FLOORF.DW** 

<Floating-point instruction>

Floating-point Convert Double to Word, round toward negative (Double)

Conversion to fixed-point format (double precision)

[Instruction format]	FLOORF.DW reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (double $\rightarrow$ word)

[Format] Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
1	2	r	r	r	0	1	1	1	1	1	1	0	0	0	1	1	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
		I	reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to 32-bit fixed-point format, and<br/>stores the result in general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{31} - 1$  to  $-2^{31}$ , an IEEE754-defined invalid operation exception is detected.

- Source is a positive number or  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  is returned.



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	teger)	0 (Int	teger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.43 FLOORF.SL

**FLOORF.SL** 

<Floating-point instruction>

Floating-point Convert Single to Long, round toward negative (Single)

Conversion to fixed-point format (single precision)

[Instruction format]	FLOORF.SL reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (single $\rightarrow$ long-word)
[Format]	Format F: I

[Opcode]

15	5				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	0	0	0	1	1	w	w	w	w	0	1	0	0	0	1	0	0	0	1	0	0
		r	reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	o-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of 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  $-\infty$  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  $2^{63} - 1$  to  $-2^{63}$ , an IEEE754-defined invalid operation exception is detected.

- Source is a positive number or  $+\infty$ :  $2^{63} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{63}$  is returned.



Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)

[Operation result]

reg2	2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Opera res [excep	ult	A (Int	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.44 FLOORF.SUL

<Floating-point instruction>

FLOORF.SUL
------------

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
ſ	r	r	r	r	r	1	1	1	1	1	1	1	0	0	1	1	w	w	W	W	0	1	0	0	0	1	0	0	0	1	0	0
Ī			regź	2															rega	3			са	iteg	ory	ty	pe		sub	o-op		

[Description]This instruction arithmetically converts the single-precision floating-point format contents of<br/>general-purpose register reg2 to unsigned 64-bit fixed-point format, and stores the result in<br/>the register pair specified by general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{64} - 1$  to 0, an IEEE754-defined invalid operation exception is detected.

- Source is a positive number outside the range of  $2^{64} 1$  to 0, or  $+\infty$ :  $2^{64} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.45 FLOORF.SUW

<Floating-point instruction>

FLOORF.SUW

Floating-point Convert Single to Unsigned-Word, round toward negative (Single)

Conversion to unsigned fixed-point format (single precision)

[Instruction format]	FLOORF.SUW reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (single $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
1	r	r	r	r	r	1	1	1	1	1	1	1	0	0	1	1	w	w	w	w	W	1	0	0	0	1	0	0	0	0	0	0
			reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-op		$\left[ \right]$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{32} - 1$  to 0, an IEEE754-defined invalid operation exception is detected.

- Source is a positive number outside the range of  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.46 FLOORF.SW

**FLOORF.SW** 

<Floating-point instruction>

Floating-point Convert Single to Word, round toward negative (Single)

Conversion to fixed-point format (single precision)

[Instruction format]	FLOORF.SW reg2, reg3
[Operation]	reg3 $\leftarrow$ floor reg2 (single $\rightarrow$ 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	0	0	0	1	1	w	w	W	w	W	1	0	0	0	1	0	0	0	0	0	0
Ī			reg2	2															reg3	3			ca	itego	ory	ty	pe		sub	-op		

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the  $-\infty$  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  $2^{31} - 1$  to  $-2^{31}$ , an IEEE754-defined invalid operation exception is detected.

- Source is a positive number or  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  is returned.



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	teger)	0 (Int	teger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.47 FMAF.S

<Floating-point instruction>

	Floating-point Fused-Multiply-add (Single)
FMAF.S	Floating-point fused-multiply-add operation (single precision)
[Instruction format]	FMAF.S reg1, reg2, reg3
[Operation]	$reg3 \leftarrow fma (reg2, reg1, reg3)$
[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 R R R R R w w w w 1 0 0 1 1 1 0 0 0 0 0
	reg2 reg1 reg3 category type sub-op
[Description]	This instruction multiplies the single-precision floating-point format contents in general- purpose register reg2 with the single-precision floating-point format contents in general- purpose register reg1, adds the single-precision floating-point format contents in general- purpose register reg3, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy. The result of the multiply operation is not rounded, but the result of the add operation is rounded, in accordance with the current rounding mode.
[Floating-point operation	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Overflow exception (O)
	Underflow exception (U)



[Operation result]

	reg2(B)								
reg3(C)	reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					+∞	-∞		
	-Normal		fma (E	3, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11	ND 7	+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
±0	+Normal					+∞	-∞		
	-Normal		fma (E	8, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11	ND 7	+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
+∞	+Normal					+∞	Q-NaN[V]		
	-Normal		+	.00		Q-NaN[V]	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	Q-NaN[V]	0.11		+∞	Q-NaN[V]		
	-∞	Q-NaN[V]	+∞	Q-Na	aN[V]	Q-NaN[V]	+∞		
-∞	+Normal					Q-NaN[V]	-∞		
	-Normal		-	∞		-∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	-∞		aN[V]	Q-NaN[V]	-∞		
	-∞	-∞	Q-NaN[V]		מאנען	-∞	Q-NaN[V]		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								

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 the hardware manual of the product used.

[Supplement]The operation is executed as if it were of infinite accuracy and the result is rounded in<br/>accordance with the current rounding mode. The result therefore differs from the result<br/>obtained when using a combination of the ADDF and MULF instructions.



# 2.4.4.48 FMSF.S

<Floating-point instruction>

FMSF.S											F	loati	ng-p	oin	Fus	ed-	-Multip	oly-su	lbtrad	ct (Sir	igle)
							Floa	ting	g-poi	nt fu	sed	-mul	tiply	-sul	otrac	t or	peratio	n (si	ngle	precis	ion)
[Instruction format]	FMSF.S reg1,	reg2, reg3																			
[Operation]	reg3 ← fms (re	eg2, reg1, reg3	3)																		
[Format]	Format F: I																				
[Opcode]																					
	15 11	10	5	4			(	0 3	31			27	26	25		23	22 2	1 20		17	7 1
	rrrr	1 1 1 1 1	1	R	R	R	RI	R '	w v	7 W	W	W	1	0	0	1	1 1	0	0	0 1	C
	reg2				I	eg1				reg	13			Са	itego	ry	type		sub-	ор	
[Description]	This instructio purpose registe purpose registe general- purpo operation is ex is not rounded current roundin	er reg2 with th er reg1, subtra se register reg ecuted as if it , but the result	e s cts 3, a we	ing the and re	le- si sto of i	preo ngle ores nfii	eisio -pr the nite	on reci e re ac	floa isio esul cura	n fl t in acy	g-p oat ge . Ti	oin ing ner he r	t fc -po al-p esu	orm int our	at o for pos	con ma e re	ntents at con regist mult	s in nter er r iply	gen its in eg3. v ope	eral- n The eratio	e on
[Floating-point operation	Unimplemente	d operation ex	(ce	ptio	on	(E)															
exceptions]	Invalid operati	on exception (	(V)																		
	Inexact except	ion (I)																			
	Overflow exce	ption (O)																			
	Underflow exc	ception (U)																			



[Operation result]

	reg2(B)								
reg3(C)	reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					+∞	-∞		
	-Normal		fms (E	8, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11	ND 7	+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
±0	+Normal					+∞	-∞		
	-Normal		fms (E	8, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11	ND 7	+∞	-∞		
	-∞	- ∞	+∞	Q-Na	aN[V]	-∞	+∞		
+∞	+Normal					Q-NaN[V]	-∞		
	-Normal		-	×0		-∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	-∞	0.11		Q-NaN[V]	-∞		
	-∞	-∞	Q-NaN[V]	Q-INA	aN[V]	-∞	Q-NaN[V]		
-∞	+Normal					+∞	Q-NaN[V]		
	-Normal		+	~		Q-NaN[V]	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	Q-NaN[V]		aN[V]	+∞	Q-NaN[V]		
	-∞	Q-NaN[V]	+∞	Q-INA	an [v]	Q-NaN[V]	+∞		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								A

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 the hardware manual of the product used.

[Supplement] The operation is executed as if it were of infinite accuracy and the result 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.



# 2.4.4.49 FNMAF.S

<Floating-point instruction>

FNMAF.S							Floa	ting-p	oint	Fused	-Neg	jate-M	ultipl	y-add	(Sing	le)
FINIMAT.S					F	loating-	point f	used-	mult	iply-ad	ld op	eratior	ו (sir	ngle pr	ecisio	on)
[Instruction format]	FNMAF.S reg	1, reg2,	reg3													
[Operation]	$reg3 \leftarrow neg (f$	ma (reg2	2, reg1, re	eg3))												
[Format]	Format F: I															
[Opcode]																
	15 11	10	5	4	0	31		27	26	25	23	22 21	20		17	16
	rrrr	1 1 1	1 1 1	RR	RRR	w w	w v	w w	1	0 0	1	1 1	0	0 1	0	0
	reg2			n	eg1		reg3			categ	ory	type		sub-oj	c	
[Description]	This instruction purpose register purpose register purpose register The operation	er reg2 v er reg1, er reg3,	vith the s adds the s	ingle-p single-	precision precision	n float on floa	ting-p ting-	poin poir	t foi it fo	rmat orma	con t con	tents ntents	in g s in	genei gene	ral- eral-	
	operation is no the current rou	ot rounde	ed, but th	t were e resul	of infir t of the	ite ac add o	cura perat	cy. T ion i	The is ro	resul	t of	the r	nul	tiply	-	
[Floating-point operation	operation is no	ot rounde Inding m	ed, but the	t were e resul signs	of infir t of the are reve	ite ac add o	cura perat	cy. T ion i	The is ro	resul	t of	the r	nul	tiply	-	
[Floating-point operation exceptions]	operation is no the current rou	ot rounde inding m ed operat	ed, but the node. The tion exce	t were e resul signs	of infir t of the are reve	ite ac add o	cura perat	cy. T ion i	The is ro	resul	t of	the r	nul	tiply	-	
	operation is no the current rou Unimplemente	ot rounde inding m ed operation exce	ed, but the node. The tion exce	t were e resul signs	of infir t of the are reve	ite ac add o	cura perat	cy. T ion i	The is ro	resul	t of	the r	nul	tiply	-	
	operation is no the current rou Unimplemente Invalid operati	ot rounde inding m ed operation exception (I)	ed, but the node. The tion excer ption (V)	t were e resul signs	of infir t of the are reve	ite ac add o	cura perat	cy. T ion i	The is ro	resul	t of	the r	nul	tiply	-	



[Operation result]

$\overline{\ }$	reg2(B)								
reg3(C)	reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					-∞	+∞		
	-Normal		–fma (I	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11	ND 7	-∞	+∞		
	-∞	+∞	-∞	Q-Na	an[V]	+∞	-∞		
±0	+Normal					-∞	+∞		
	-Normal		–fma (I	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞			-∞	+∞		
	-∞	+∞	-∞	Q-Na	aN[V]	+∞	-∞		
+∞	+Normal					-∞	Q-NaN[V]		
	-Normal		-	∞		Q-NaN[V]	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	Q-NaN[V]	0.11	ND 7	-∞	Q-NaN[V]		
	-∞	Q-NaN[V]	-∞	Q-Na	aN[V]	Q-NaN[V]	-∞		
-∞	+Normal					Q-NaN[V]	+∞		
	-Normal		+	.00		+∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	+∞	0.14		Q-NaN[V]	+∞		
	-∞	+∞	Q-NaN[V]	Q-Na	an [v]	+∞	Q-NaN[V]		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								

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 the hardware manual of the product used.

[Supplement] The operation is executed as if it were of infinite accuracy and the result 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.



#### 2.4.4.50 FNMSF.S

<Floating-point instruction>

**FNMSF.S** 

Floating-point Fused-Negate-Multiply-subtract	(Sinale)
riouting point ruood riogato matiply oubtract	(Chingle)

Floating-point fused-multiply-subtract operation (single precision)

[Instruction format]	FNMSF.S reg1, reg2, reg3
[Operation]	$reg3 \leftarrow neg (fms (reg2, reg1, reg3))$
[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	R	R	R	R	R	w	w	w	w	W	1	0	0	1	1	1	0	0	1	1	0
ļ			reg2	2										reg1	I				reg3	3			ca	tego	ory	ty	pe		sub	-ор		

[Description] This instruction multiplies the single-precision floating-point format contents in generalpurpose register reg2 with the single-precision floating-point format contents in purpose register reg1, subtracts the single-precision floating-point format contents in general- purpose register reg3, inverts the sign, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy. The result of the multiply operation is not rounded, but the result of the subtract operation is rounded, in accordance with the current rounding mode. The signs are reversed after rounding.

[Floating-point operationUnimplemented operation exception (E)exceptions]Invalid operation exception (V)Inexact exception (I)Overflow exception (O)Underflow exception (U)Overflow exception (U)



[Operation result]

	reg2(B)								
reg3(C)	reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					-∞	+∞		
	-Normal		–fms (E	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-∞	Q-Na	aN[V]	+∞	-∞		
±0	+Normal					-∞	+∞		
	-Normal		–fms (E	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-∞	Q-Na	aN[V]	+∞	-∞		
+∞	+Normal			+∞					
	-Normal		+	.00		+∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	+∞	Q-Na	N ID /1	Q-NaN[V]	+∞		
	-∞	+∞	Q-NaN[V]	Q-Na	an [v]	+∞	Q-NaN[V]		
-∞	+Normal					-∞	Q-NaN[V]		
	-Normal		-	-00		Q-NaN[V]	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	Q-NaN[V]		aN[V]	-∞	Q-NaN[V]		
	-∞	Q-NaN[V]	-∞	Q-No	an [v]	Q-NaN[V]	-∞		
Q-NaN	±Normal								
	±0			Q-1	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								

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 the hardware manual of the product used.

[Supplement] The operation is executed as if it were of infinite accuracy and the result 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.



## 2.4.4.51 MAXF.D

<Floating-point instruction>

MAXF.D																	Floa	ating	-po	int N	laxi	mum	n (Do	uble
												F	Floa	ating	-poi	nt n	naxi	mun	n va	alue	(do	uble	preci	sior
[Instruction format]	MAXF.D reg1	, reg2	2, reg	3																				
[Operation]	$reg3 \leftarrow max (r$	eg2, 1	reg1)																					
[Format]	Format F: I																							
[Opcode]																								
	15 11	10		5	4				0	31				27	26	25		23	22	21	20		1	7 1
	rrrr O	1 1	1 1	1 1	R	R	R	R	0	W	W	W	W	0	1	0	0	0	1	1	1	1	0	0 0
	reg2						reg1					reg	3			C	ateg	ory	ty	pe		sub	-ор	
[Description]	This instruction format data in stores it in the If one of the so	the re regist	egiste ter pa	r pair ir spe	spe cifi	ecif ied	ied by g	by gen	gei era	ner 11-p	al oui	-pu po	ırp se	ose reg	registe	gis er 1	ter: eg	s re 3.	g1	anc	l re	eg2,	and	
	detected. If invoccurs.		-															-						
[Floating-point operation exceptions]	Invalid operati	on ex	cepti	on (V	)																			
[Supplement]	When both reg reg3.	1 and	l reg2	l is eit	hei	r +0	) or	-0,	it	is	un	def	in	ed v	wh	eth	er -	+0 (	or -	–0 i	s s	tore	ed ir	1



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0								
-0		reg1(A)						
+∞								
-∞								
Q-NaN			reg	2(B)			Q-NaN	
S-NaN								Q-NaN[V]

**Note:** [ ] indicates an exception that must occur.


# 2.4.4.52 MAXF.S

<Floating-point instruction>

MAXF.S														Floa	ting-	poir	nt M	axir	num (	Sing	gle)
										FI	patin	g-po	int n	naxir	num	val	ue (	sinę	gle pre	ecisi	on)
[Instruction format]	MAXF.S reg1	l, reg2, reg3																			
[Operation]	$reg3 \leftarrow max$ (	$eg3 \leftarrow max (reg2, reg1)$																			
[Format]	Format F: I	Format F: I																			
[Opcode]																					
	15 11	10	5	4			0	31			27	26	25		23 2	2 2	21 2	20		17	16
	rrrrr	1 1 1 1 1	1	R	R R	R	R	W	W	w	v w	1	0	0	0	1	1	0	1 0	0	0
	reg2				reg	1		1	r	eg3		Ì	са	tego	ry	type	e	ę	sub-op	)	
[Description]	data in genera reg3. If one of	on extracts the al-purpose regis f the source op letected. If inva occurs.	ster: erai	s re 1ds	eg1 a is S-	nd re NaN	eg2 N, a	2, a an 1	ind IEE	stor EE7:	es i 54-d	t in lefii	gei ned	nera inv	al-p valio	urp 1 o	oose per	e ro ati	egist on	er	
[Floating-point operation exceptions]	Invalid operat	tion exception	(V)																		
[Supplement]	When both reg reg3.	g1 and reg2 is	eith	er	+0 or	<sup>.</sup> –0,	it	is ı	und	efir	ed v	whe	the	r +	0 01	: _(	) is	ste	ored	in	
	A subnormal i	input will not b	oe fl	lus	hed e	ven	if	the	FS	bit	of t	he	FPS	SR	regi	ste	r is	1.			



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0								
-0			IVIAX	(B, A)			reg1(A)	
+∞								
-∞								
Q-NaN			reg	2(A)			Q-NaN	
S-NaN								Q-NaN[V]

**Note:** [ ] indicates an exception that must occur.



## 2.4.4.53 MINF.D

<Floating-point instruction>

	Floating-point Minimum (Double)
MINF.D	Floating-point minimum value (double precision)
[Instruction format]	MINF.D reg1, reg2, reg3
[Operation]	$reg3 \leftarrow min (reg2, reg1)$
[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 R R R R 0 w w w w 0 1 0 0 0 1 1 1 1 0 1 0
	reg2 reg1 reg3 category type sub-op
[Description]	This instruction extracts the minimum value from the double-precision floating-point format data in the register pair specified by general-purpose registers reg1 and reg2, and stores it in the register pair specified by general-purpose register reg3. If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is
	detected. If invalid operation exceptions are not enabled, Q-NaN is stored and no exception occurs.
[Floating-point operation exceptions]	Invalid operation exception (V)
[Supplement]	When both reg1 and reg2 is either $+0$ or $-0$ , whether $+0$ or $-0$ is stored in reg3 is undefined.
	A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN			
+Normal											
-Normal											
+0			MINT								
-0			MIN	(B, A)			reg1(A)				
+∞											
-∞											
Q-NaN		reg2(B) Q-NaN									
S-NaN								Q-NaN[V]			

**Note:** [ ] indicates an exception that must occur.



## 2.4.4.54 MINF.S

<Floating-point instruction>

		Floating-point Minimum (Single)							
MINF.S	Floating	g-point minimum value (single precision)							
[Instruction format]	MINF.S reg1, reg2, reg3								
[Operation]	$reg3 \leftarrow min (reg2, reg1)$	$eg3 \leftarrow min (reg2, reg1)$							
[Format]	Format F: I	ormat F: I							
[Opcode]									
	<u>15 11 10 5 4 0 31 27</u>	26 25 23 22 21 20 17 16							
	r r r r r 1 1 1 1 1 1 R R R R R w w w w	1 0 0 0 1 1 0 1 0 1 0							
	reg2 reg1 reg3	category type sub-op							
[Description]	This instruction extracts the minimum value from the single- data in general-purpose registers reg1 and reg2, and stores it reg3. If one of the source operands is S-NaN, an IEEE754-de exception is detected. If invalid operation exceptions are not no exception occurs.	in general-purpose register efined invalid operation							
[Floating-point operation exceptions]	Invalid operation exception (V)								
[Supplement]	When both reg1 and reg2 is either $+0$ or $-0$ , whether $+0$ or $-0$	0 is stored in reg3 is undefined.							
	A subnormal input will not be flushed even if the FS bit of the	ne FPSR register is 1.							



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0			MIN					
-0			IVIIN	(B, A)			reg1(A)	
+∞								
-∞								
Q-NaN			reg	2(B)			Q-NaN	
S-NaN								Q-NaN[V]

**Note:** [ ] indicates an exception that must occur.



# 2.4.4.55 MULF.D

<Floating-point instruction>

MULF.D		Floating-point Multiply (Double)						
	Floating-	-point multiplication (double precision)						
[Instruction format]	MULF.D reg1, reg2, reg3							
[Operation]	$reg3 \leftarrow reg2 \times reg1$	$g3 \leftarrow reg2 \times reg1$						
[Format]	Format F: I	ormat F: I						
[Opcode]								
	15 11 10 5 4 0 31 27 26							
	r r r r 0 1 1 1 1 1 1 R R R R 0 w w w w 0 1	0 0 0 1 1 1 0 1 0 0						
	reg2 reg1 reg3	category type sub-op						
[Description]	This instruction multiplies the double-precision floating-point register pair specified by general-purpose register reg2 by the opoint point format contents in the register pair specified by general-purpose stores the results in the register pair specified by general-purpose	double-precision floating- purpose register reg1 and						
[Floating-point operation	Unimplemented operation exception (E)							
exceptions]	Invalid operation exception (V)							
	Inexact exception (I)							
	Overflow exception (O)							
	Underflow exception (U)							



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-%		
-Normal					-∞	+∞		
+0		B	×A		0.11			
-0					Q-Na	aniv		
+∞	+∞	-8			+∞	-8		
-∞	-∞	+∞	Q-IN6	aN[V]	-∞	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



# 2.4.4.56 MULF.S

<Floating-point instruction>

MULF.S													Float	ing-poir	nt Mi	ultip	ly (Sii	ngle)
										Floa	ting-	poin	t multip	olicatior	n (sir	igle	preci	sion)
[Instruction format]	MULF.S reg1	l, reg2, reg3																
[Operation]	$reg3 \leftarrow reg2$	$rg3 \leftarrow reg2 \times reg1$																
[Format]	Format F: I	ormat F: I																
[Opcode]																		
	15 11	10	5	4		0	31			27	26	25	23	22 21	20		1	7 16
	rrrrr	1 1 1 1 1	1	R	RR	R R	w	W	w v	w w	1	0	0 0	1 1	0	0	1 (	0
	reg2				reg1			I	reg3			cat	egory	type		sub	-ор	
[Description]	This instruction purpose registry purpose registry	ter reg2 by the	e sing	gle	-precis	ion f	floa	tin	g-po	int 1	forn	nat	conte	ents of		-		-
[Floating-point operation	Unimplement	ted operation e	excej	ptio	on (E)													
exceptions]	Invalid operat	tion exception	(V)															
	Inexact excep	Inexact exception (I)																
	Overflow exc	eption (O)																
	Underflow ex	ception (U)																



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-∞		
-Normal		-			-∞	+∞		
+0		B:	×A		0.11			
-0					Q-Na	aN[V]		
+∞	+∞	-8			+∞	-8		
-∞	- 8	+∞	Q-Na	an[v]	-∞	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.57 NEGF.D

<Floating-point instruction>

	Floating-point Negate (Double)					
NEGF.D	Floating-point sign inversion (double precision)					
[Instruction format]	NEGF.D reg2, reg3					
[Operation]	$reg3 \leftarrow neg reg2$					
[Format]	rmat F: I					
[Opcode]						
	15       11       10       5       4       0       31       27       26       25       23       22       21       20       17       16         rrrr0       1       1       1       1       1       0       0       0       1       wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww					
	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 results in the register pair specified by the 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.					



#### 2.4.4.58 NEGF.S

<Floating-point instruction>

NEGF.S	Floating-point Negate (Single)
	Floating-point sign inversion (single precision)
[Instruction format]	NEGF.S 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 r 1 1 1 1 1 1 0 0 0 0 1 w w w w w 1 0 0 0 1 0 0 1 0 0 0
	reg2 reg3 category type sub-op
[Description]	This instruction inverts the sign of the single-precision floating-point format contents of 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.



# 2.4.4.59 RECIPF.D

oint ir 

RECIPF.D	Reciprocal of a Floating-point Value (Double)
REGIFF.D	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 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 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	Unimplemented operation exception (E)
exceptions]	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/A	[]	+∞[Z]	–∞[Z]	+0	-0	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

### CAUTION

The results obtained fall within the 1ULP error range against the results of computing 1/x. ULP: Unit in the Least-significant Place



**Note 2.** When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

## 2.4.4.60 RECIPF.S

<Floating-point instruction>

<b>RECIPF.S</b>													F	leci	oroc	al of	a Flo	ating-p	oint	Value	(Sin	igle)
NLUIFF.J																	Re	ciproca	al (si	ngle pr	ecisi	ion
[Instruction format]	RECIP	F.S reg	g2, reg	g3																		
[Operation]	reg3 ←	- 1 ÷ re	g2																			
[Format]	Format	F: I																				
[Opcode]																						
	15	11	10			5	4			0 3	31			27	26 2	25	23	22 21	20		17	7 1
	rrı	rr	1 1	1	1 1	1	0 0	0	0	1	wv	v w	w	w	1	0	0 0	1 0	0	1 1	. 1	
	re	g2								Ì		reg	3			cate	egory	type	Ì	sub-o	р	Ī
[Description]	This in content reg3. T	s of ge	eneral	purp	ose	regi	ster	reg2	2, ar	d s	stor	es tl	he re	su	t ir	ı ge	enera	ıl-pur	pos			
[Floating-point operation	Unimp	lement	ed op	erati	on ey	ссер	tion	(E)														
exceptions]	Invalid	operat	ion ex	kcept	tion (	(V)																
	Inexact	excep	tion (	I)																		
	Divisio	n by z	ero ex	cept	ion (	Z)																
	Underf																					

[Operation result]

reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	1/A	[]	+∞[Z]	-∞[Z]	+0	-0	Q-NaN	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 the hardware manual of the product used.

### CAUTION

The results obtained fall within the 1ULP error range against the results of computing 1/x. ULP: Unit in the Least-significant Place



17 16

#### 2.4.4.61 **ROUNDF.DL**

**ROUNDF.DL** 

----

<Floating-point instruction>

Floating-point Convert Double to Long, round to nearest (Double)

Convert to integer format (double-precision)

[Instruction format]	ROUNDF.DL reg2, reg3										
[Operation]	reg3 $\leftarrow$ round reg2 (double $\rightarrow$ long-word)										
[Format]	Format F: I										
[Opcode]											
	15 11 10 5 4 0 31 27 <u>26 25 23 22 21 20 17 16</u>										
	r r r r 0 1 1 1 1 1 1 0 0 0 0 0 w w w w 0 1 0 0 0 1 0 1										
	reg2 reg3 category type sub-op										
[Description]	<ul> <li>This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg2 to 64-bit integer format and stores the result in the register pair specified by general-purpose register reg3.</li> <li>The result is rounded to the nearest value or an even number regardless of the current rounding mode.</li> <li>If the source operand is an infinite number or not-a-number or if the rounded result is outside the range of 2<sup>63</sup> – 1 to – 2<sup>63</sup>, an IEEE754-defined invalid operation exception is detected.</li> <li>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.</li> <li>Source is a positive number or +∞: 2<sup>63</sup> – 1 is returned.</li> <li>Source is a negative number, not-a-number, or –∞: –2<sup>63</sup> is returned.</li> </ul>										



[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	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.62 **ROUNDF.DUL**

<Floating-point instruction>

**ROUNDF.DUL** 

Floating-point Convert Double to Unsigned-Long, round to nearest (Double)

Convert to unsigned integer format (double-precision)

[Instruction format]	RO	UND	F.DI	JL	reg2	l, reg	g3																				
[Operation]	reg	3 ← ı	round	l re	g2 (d	doul	ble -	→ ι	un	nsig	ned	lor	ıg-	wc	ord)	)											
[Format]	For	mat H	F: I																								
[Opcode]																											
	15		1'	1 10				5	4	1			0	31				27	26	25	23	2	2 21	20		17	<b>7</b> 16
	r	r r	r 0	1	1	1 1	L 1	1	1	L 0	0	0	0	w	W	W	w	0	1	0 0	0	-	1 0	1	0	1 0	0
		reg2	2													reg3	3			cate	gory		type		sub-	ор	
[Description]	of t forr	s inst he reg mat a e resu	giste nd st	r pa ores	ir sp s the	ecif res	fied ult i	by n tł	ge he	ene e reş	ral- giste	pur er p	po: air	se i sp	reg eci	iste fie	er r d b	eg2 yg	to ene	uns eral-	igne purj	ed po	64-l se re	bit egis	inte ster	ger reg3	
	rou	nding	g moo	le.																							
	rou	he sou nded eptio	resu	lt is	outs	side														-							
	regi	nvalio ister i ows,	is set	as a	an ir	ıval	id oj	per	at	ion	and	d no	o ez	xce	epti						`						8

- Source is a positive number outside the range of  $2^{64} 1$  to 0 or  $+\infty$ :  $2^{64} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 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.



## 2.4.4.63 ROUNDF.DUW

**ROUNDF.DUW** 

<Floating-point instruction>

Floating-point Convert Double to Unsigned-Word, round to nearest (Double)

Convert to unsigned integer format (double-precision)

[Instruction format]	ROUNDF.DUW reg2, reg3
[Operation]	reg3 $\leftarrow$ round reg2 (double $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
1	2	r	r	r	0	1	1	1	1	1	1	1	0	0	0	0	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
		I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents<br/>of the register pair specified by general-purpose register reg2 to unsigned 32-bit integer<br/>format and stores the result in the general-purpose register reg3.

The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number, not-a-number, or negative number, or if the rounded result is outside the range of  $2^{32} - 1$  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  $2^{32} 1$  to 0 or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 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.



## 2.4.4.64 ROUNDF.DW

**ROUNDF.DW** 

<Floating-point instruction>

Floating-point Convert Double to Word, round to nearest (Double)

Convert to integer format (double-precision)

[Instruction format]	ROUNDF.DW reg2, reg3
[Operation]	reg3 $\leftarrow$ round reg2 (double $\rightarrow$ word)
[Format]	Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
ı	2	r	r	r	0	1	1	1	1	1	1	0	0	0	0	0	w	w	w	w	W	1	0	0	0	1	0	1	0	0	0	0
			reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	o-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents<br/>of the register pair specified by general-purpose register reg2 to 32-bit integer format and<br/>stores the result in the general-purpose register reg3.

The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number or not-a-number or if the rounded result is outside the range of  $2^{31} - 1$  to  $-2^{31}$ , 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  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  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 (Int	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.65 **ROUNDF.SL**

**ROUNDF.SL** 

<Floating-point instruction>

Floating-point Convert Single to Long, round to nearest (Single)

27 26 25

1

0 0 0

category

w w w O

reg3

Convert to integer format (single precision)

23 22 21 20

1 0

type

0 0 1

sub-op

17 16

0 0

	reg2
	r r r r r 1 1 1 1 1 1 0 0 0 0 0 v
	15 11 10 5 4 0 3
[Opcode]	
[Format]	Format F: I
[Operation]	reg3 $\leftarrow$ round reg2 (single $\rightarrow$ long-word)
[Instruction format]	ROUNDF.SL reg2, reg3

[Description] This instruction arithmetically converts the single-precision floating-point format contents of the general-purpose register reg2 to 64-bit integer format and stores the result in the general-purpose register reg3.

> The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number or not-a-number or if the rounded result is outside the range of  $2^{63} - 1$  to  $-2^{63}$ , 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  $+\infty$ :  $2^{63} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{63}$  is returned.



[Floating-point operationUnimplemented operation exception (E)exceptions]Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	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.



## 2.4.4.66 ROUNDF.SUL

**ROUNDF.SUL** 

<Floating-point instruction>

Floating-point Convert Single to Unsigned-Long, round to nearest (Single)

27 26 25

1

0 0 0

category

0

Convert to unsigned integer format (single precision)

23 22 21 20

1 0

type

17 16

0

0 0 1 0

sub-op

	reg2	reg3
	r r r r r 1 1 1 1 1 1	1 0 0 0 0 w w w w
	15 11 10 5	4 0 31
[Opcode]		
[Format]	Format F: I	
[Operation]	$reg3 \leftarrow round reg2 (single \rightarrow ur$	nsigned long-word)
[Instruction format]	ROUNDF.SUL reg2, reg3	

[Description] This instruction arithmetically converts the single-precision floating-point format contents of the general-purpose register reg2 to unsigned 64-bit integer format and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number, not-a-number, or negative number, or if the rounded result is outside the range of  $2^{64} - 1$  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  $2^{64} 1$  to 0 or  $+\infty$ :  $2^{64} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 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.



## 2.4.4.67 ROUNDF.SUW

**ROUNDF.SUW** 

<Floating-point instruction>

Floating-point Convert Single to Unsigned-Word, round to nearest (Single)

Convert to unsigned integer format (single precision)

[Instruction format]	ROUNDF.SUW reg2, reg3
[Operation]	reg3 $\leftarrow$ round reg2 (single $\rightarrow$ unsigned

[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	0	0	0	0	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
	I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	-op		$\left  \right $

word)

[Description] This instruction arithmetically converts the single-precision floating-point format contents of the general-purpose register reg2 to unsigned 32-bit integer format and stores the result in the general-purpose register reg3.

The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number, not-a-number, or negative number, or if the rounded result is outside the range of  $2^{32} - 1$  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  $2^{32} 1$  to 0 or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 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.



## 2.4.4.68 ROUNDF.SW

**ROUNDF.SW** 

<Floating-point instruction>

Floating-point Convert Single to Word, round to nearest (Single)

Convert to integer format (single precision)

[Instruction format]	ROUNDF.SW reg2, reg3
[Operation]	reg3 $\leftarrow$ round reg2 (single $\rightarrow$ 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	0	0	0	0	0	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
		reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	o-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of the general-purpose register reg2 to 32-bit integer format and stores the result in the general-purpose register reg3.

The result is rounded to the nearest value or an even number regardless of the current rounding mode.

If the source operand is an infinite number or not-a-number or if the rounded result is outside the range of  $2^{31} - 1$  to  $-2^{31}$ , 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  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  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 (Int	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.69 RSQRTF.D

**RSQRTF.D** 

<Floating-point instruction>

Reciprocal of the Square Root of a Floating-point Value (Double)

Reciprocal of square root (double precision)

[Instruction format]	RSQRTF.D r	eg2, reg	3																		
[Operation]	$reg3 \leftarrow 1 \div (s)$	sqrt reg2	2)																		
[Format]	Format F: I																				
[Opcode]																					
	15 11	I 10	5	4	Ļ		0	31			27	26	25	23	22	21	20			17	16
	rrrr O	1 1 1	1111	0	0 0	0 1	0	w	ww	w w	0	1	0	0 0	1	0	1	1	1	1	0
	reg2								reç	g3			cate	egory	ty	ype		sub	-ор		
[Description]	point format of	contents the reci urpose re fers fror	of the reg procal of egister reg	thi 3.	ter p is res	air sp sult a	nd	fied	d by res th	ger ne re	nera esu	al-p lt ii	urpo n the	ose r e reg	egi jisto	istei er p	r re oair	eg2, • spe	, the ecif	en Tied	-
[Floating-point operation	Unimplement	ted operation	ation exce	pt	ion (	(E)															
exceptions]	Invalid operation	tion exc	eption (V)	)												recision floating ster reg2, then er pair specified					
	Inexact excep	otion (I)																			
	Division by z	ero exce	eption (Z)																		



[Operation result]

reg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result exception]	1/√A[1]	Q-NaN[V]	+∞[Z]	-∞[Z]	+0	Q-NaN[V]	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

### CAUTION

The results obtained fall within the 2ULP error range against the results of computing  $1/\sqrt{x}$ . ULP: Unit in the Least-significant Place



**Note 2.** When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

0

Reciprocal of the Square Root of a Floating-point Value (Single)

#### 2.4.4.70 **RSQRTF.S**

<Floating-point instruction>

[Operation]

[Format]

[Opcode]

**RSQRTF.S** Reciprocal of square root (single precision) [Instruction format] RSQRTF.S reg2, reg3  $reg3 \leftarrow 1 \div (sqrt reg2)$ Format F: I 11 10 54 0 31 27 26 25 23 22 21 20 15 17 16 1 1 1 1 1 1 0 0 0 1 0 w w w w w rrrrr 1 0 0 0 1 0 0 1 1 1 reg2 reg3 category type sub-op

[Description] This instruction obtains the arithmetic positive square root of the single-precision floatingpoint format contents of general-purpose register reg2, then approximates the reciprocal of this result and stores it in general-purpose register reg3. The result differs from the result obtained when using a combination of the SQRTF and DIVF instructions.

[Floating-point operation Unimplemented operation exception (E) exceptions] Invalid operation exception (V) Inexact exception (I) Division by zero exception (Z)

[Operation result]

reg2	(A) +Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Opera resi [excep	lt 1/√A[1]	Q-NaN[V]	+∞[Z]	-∞[Z]	+0	Q-NaN[V]	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized Note 2. numbers shown in the hardware manual of the product used.

#### CAUTION

The results obtained fall within the 2ULP error range against the results of computing  $1/\sqrt{x}$ . ULP: Unit in the Least-significant Place



# 2.4.4.71 SQRTF.D

<Floating-point instruction>

Square root (double precision)
SQRTF.D reg2, reg3
$reg3 \leftarrow sqrt reg2$
Format F: I
15       11       10       5       4       0       31       27       26       25       23       22       21       20       17       16         r       r       r       r       r       r       1       1       1       1       0       0       0       w       w       w       0       1       0       1       1       1       1       1       0         reg2       reg3       reg3       category       type       sub-op       1
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 $-0$ .
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	Q-NaN[V]	+0	-0	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



# 2.4.4.72 SQRTF.S

<Floating-point instruction>

SQRTF.S		Floating-point Square Root (Single								
		Square root (single precision								
[Instruction format]	SQRTF.S reg2, reg3									
[Operation]	reg3 ← sqrt reg2									
[Format]	Format F: I									
[Opcode]										
	15 11 10 5 4 0 31 27	26 25 23 22 21 20 17 1								
	r r r r r 1 1 1 1 1 1 0 0 0 0 0 w w w w w	1 0 0 0 1 0 0 1 1 1								
	reg2 reg3	category type sub-op								
[Description]	This instruction obtains the arithmetic positive square root of point format contents of general-purpose register reg2, and stregister reg3. The operation is executed as if it were of infinit rounded in accordance with the current rounding mode. Whet $-0$ , the result becomes $-0$ .	stores it in general-purpose ite accuracy, and the result is								
[Floating-point operation	Unimplemented operation exception (E)									
exceptions]	Invalid operation exception (V)									

[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.


## 2.4.4.73 SUBF.D

<Floating-point instruction>

														Floati	ng-p	oint	Sub	tract (	Doub	ole)
											Floa	ating	-poi	nt sub	tract	tion	(dou	ble pr	ecisio	on)
SUBF.D reg1,	reg2,	reg3																		
$reg3 \leftarrow reg2 -$	reg1																			
Format F: I																				
15 11	10			54			0	31			27	26	25	23	3 22	21	20		17	16
rrrr O	1 1	1 1	1	1 R	RF	RR	0	w	w w	w	0	1	0	0 0	1	1	1	0 0	1	0
reg2					re	g1			reg	3			cat	egory	ty	pe		sub-op	)	
pair specified format content the result in the executed as if	by gen ts of tl e regi it wer	neral- he reg ster p re of i	-purp giste pair s	pose r pai speci	regis r spe fied l	ter re cified by ge	eg l d b ene	l fro y ge ral-	om tl enera purp	ne d al-p ose	ou urp re	ble- oose gist	pre e re er 1	ecisio giste reg3	on f er re . Th	loa g2, ie o	ting , an per	g-poi d sto atior	nt res	
Invalid operati Inexact except	ion ex	ception)		_	ion (E	E)														
	-		)																	
	reg3 $\leftarrow$ reg2 – Format F: I 15 11 r r r r r 0 reg2 This instruction pair specified format content the result in the executed as if current roundi Unimplemented Invalid operation Inexact except Overflow exception	reg3 $\leftarrow$ reg2 – reg1 Format F: I $15 \qquad 11 \ 10$ $\boxed{r \ r \ r \ r \ 0} \ 1 \ 1$ $\boxed{reg2}$ This instruction sub pair specified by gen format contents of the the result in the regi executed as if it wer current rounding models Unimplemented open Invalid operation ex Inexact exception (I Overflow exception	reg3 $\leftarrow$ reg2 – reg1 Format F: I $15 \qquad 11 \ 10$ $\boxed{r \ r \ r \ r \ r \ 0} \ 1 \ 1 \ 1 \ 1 \ 1}$ reg2 This instruction subtracts pair specified by general- format contents of the reg the result in the register p executed as if it were of it current rounding mode. Unimplemented operation Invalid operation exception Inexact exception (I) Overflow exception (O)	Format F: I 15   11   10 $x   r   r   r   0   1   1   1   1   1$ $reg2$ This instruction subtracts the pair specified by general-purp format contents of the register the result in the register pair s executed as if it were of infin current rounding mode. Unimplemented operation exception (1)	$reg3 \leftarrow reg2 - reg1$ Format F: I $15  11  10 \qquad 5  4$ $\boxed{r  r  r  r  0}  1  1  1  1  1  1  1  1$ $\boxed{reg2}$ This instruction subtracts the doul pair specified by general-purpose format contents of the register pair the result in the register pair specified as if it were of infinite a current rounding mode. Unimplemented operation exception (V) Inexact exception (I) Overflow exception (O)	reg3 $\leftarrow$ reg2 – reg1 Format F: I $15 \qquad 11 \ 10 \qquad 5 \ 4$ $\boxed{r \ r \ r \ r \ r \ 0} \qquad 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ F}$ $\boxed{reg2} \qquad reg$ This instruction subtracts the double-pr pair specified by general-purpose regiss format contents of the register pair specified to executed as if it were of infinite accura current rounding mode. Unimplemented operation exception (R Invalid operation exception (V) Inexact exception (I) Overflow exception (O)	reg3 $\leftarrow$ reg2 – reg1 Format F: I $15  11 \ 10 \qquad 5 \ 4$ $\boxed{r \ r \ r \ r \ r \ 0} \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R$	reg3 $\leftarrow$ reg2 – reg1 Format F: I $15  11 \ 10 \qquad 5 \ 4 \qquad 0$ $\boxed{r \ r \ r \ r \ r \ 0} \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R$	$reg3 \leftarrow reg2 - reg1$ Format F: I $\frac{15  11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31}{ r \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R$	$reg3 \leftarrow reg2 - reg1$ Format F: I $\frac{15  11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31}{ r \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R$	SUBF.D reg1, reg2, reg3 reg3 $\leftarrow$ reg2 - reg1 Format F: I $15 \qquad 11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31$ $\boxed{r \ r \ r \ r \ 0}$ 1 1 1 1 1 1 1 $\boxed{R \ R \ R \ 0}$ wwww reg2 reg1 reg3 This instruction subtracts the double-precision floating-popair specified by general-purpose register reg1 from the d format contents of the register pair specified by general-purpose executed as if it were of infinite accuracy, and the result i current rounding mode. Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I) Overflow exception (O)	SUBF.D reg1, reg2, reg3 reg3 $\leftarrow$ reg2 - reg1 Format F: I $\frac{15 \qquad 11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31 \qquad 27}{ r \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R$	SUBF.D reg1, reg2, reg3 reg3 $\leftarrow$ reg2 – reg1 Format F: I $15 \qquad 11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31 \qquad 27 \ 26$ $\boxed{r \ r \ r \ r \ 0} \qquad 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R \$	SUBF.D reg1, reg2, reg3 reg3 $\leftarrow$ reg2 - reg1 Format F: I $15  11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31 \qquad 27 \ 26 \ 25 \ \hline x \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R \ 0 \ w \ w \ w \ 0 \ 1 \ 0 \ reg1 \qquad reg3 \ cat$ This instruction subtracts the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register pair specified by general-purpose register reg1 from the double-precision floating-point formar pair specified by general-purpose register pair specified by general-pu	SUBF.D reg1, reg2, reg3 reg3 $\leftarrow$ reg2 - reg1 Format F: I $15  11 \ 10 \qquad 5 \ 4 \qquad 0 \ 31 \qquad 27 \ 26 \ 25 \qquad 23$ $r \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R$	Floating-point subtract         SUBF.D reg1, reg2, reg3         reg3 $\leftarrow$ reg2 – reg1         Format F: I         15 11 10 5 4 0 31 27 26 25 23 22 $x \ r \ r \ r \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R \ 0 \ w \ w \ w \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ reg3 \ category \ ty$ This instruction subtracts the double-precision floating-point format content pair specified by general-purpose register reg1 from the double-precision f format contents of the register pair specified by general-purpose register reg3. The executed as if it were of infinite accuracy, and the result is rounded in accoccurrent rounding mode.         Unimplemented operation exception (E)         Invalid operation exception (V)         Inexact exception (I)         Overflow exception (O)	Floating-point subtraction         SUBF.D reg1, reg2, reg3         reg3 $\leftarrow$ reg2 - reg1         15 11 10 5 4 0 31 27 26 25 23 22 21          r r r r 0 1 1 1 1 1 1 R R R R 0 w w w 0 1 0 0 0 1 1         reg1         This instruction subtracts the double-precision floating-point format contents pair specified by general-purpose register reg1 from the double-precision floa format contents of the register pair specified by general-purpose register reg2, the result in the register pair specified by general-purpose register reg3. The o executed as if it were of infinite accuracy, and the result is rounded in accorda current rounding mode.         Unimplemented operation exception (E)         Invalid operation exception (V)         Inexact exception (I)         Overflow exception (O)	SUBF.D reg1, reg2, reg3         reg3 $\leftarrow$ reg2 - reg1         Format F: I         15       11 10       5 4       0 31       27 26 25       23 22 21 20 $x \ x \ x \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ R \ R \ R$	Floating-point subtraction (double pr         SUBF.D reg1, reg2, reg3         reg3 $\leftarrow$ reg2 – reg1         Format F: I         15 11 10 5 4 0 31 27 26 25 23 22 21 20 $r r r r 0 1 1 1 1 1 1 1 1 R R R R 0 w w w w 0 1 0 0 0 1 1 1 0 0         reg1 reg3 category type sub-of         This instruction subtracts the double-precision floating-point format contents of the regain specified by general-purpose register reg1 from the double-precision floating-point format contents of the register pair specified by general-purpose register reg3. The operation exception the result is rounded in accordance wit current rounding mode.         Unimplemented operation exception (E)         Invalid operation exception (V)         Inexact exception (I)         Overflow exception (O)   $	$reg3 \leftarrow reg2 - reg1$ Format F: I $\frac{15  11 \ 10  5 \ 4  0 \ 31  27 \ 26 \ 25  23 \ 22 \ 21 \ 20  17}{                                    $



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		5						
+0		B	– A		+∞	-∞		
-0								
+∞		-	-00		Q-NaN[V]			
-∞			+∞			Q-NaN[V]		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.74 SUBF.S

<Floating-point instruction>

SUBF.S											FI	oatir	ng-poir	nt Su	btract	(Sing	le)
									Flo	atin	g-point	sub	tractio	n (sir	ngle pr	ecisio	on)
[Instruction format]	SUBF.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
	rrrr	1 1 1 1	1 1	R	RRR	. R	w	w w w	W	1	0 0	0	1 1	0	0 0	1	0
	reg2				reg1			reg3			categ	ory	type		sub-op	þ	
[Description]	This instruction purpose regist purpose regist is executed as the current roo	er reg1 from er reg2, and if it were o	n the s l store f infin	ing s th	gle-prec ne result	isio in	n fl gen	oating-p eral-pur	ooir pos	t fo e re	ormat egiste	con r re	ntents g3. T	s of The	gene opera	ral- atior	1
[Floating-point operation exceptions]	Unimplemento Invalid operat	-		otio	on (E)												
	Inexact except	tion (I)															
	Overflow exce	eption (O)															
	Underflow ex	ception (U)															



[Operation result]

reg2(B) reg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		5						
+0		B	- A		+∞	-∞		
-0								
+∞		-	-00		Q-NaN[V]			
-∞			+∞			Q-NaN[V]		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.75 TRFSR

<Floating-point instruction>

TRFSR											Trar	isfei	s spe	cifie	d CC I	oit to	Zerc	o flag	, in	PSW	(Sing	jle)
																				Flag	trans	fer
[Instruction format]	TRFSR fcbit TRFSR																					
[Operation]	PSW.Z ← fct	oit																				
[Format]	Format F: I																					
[Opcode]	15 11	10			5	4			0	31			27	′26	25	22	22	21	20		17	16
	0 0 0 0 0		. 1 :	1 1			0 0	0	0	1	0	0		1	0 0		0			f f		0
	Remark: fcbit	:fff													cate	gory	typ	be		sub-c	φ	
[Description]	This instruction register species transfers the O	fied b	y fcb	it to	the																	
[Floating-point operation exceptions]	None																					



## 2.4.4.76 TRNCF.DL

**TRNCF.DL** 

<Floating-point instruction>

Floating-point Convert Double to Long, round toward zero (Double)

Conversion to fixed-point format (double precision)

[Instruction format]	TR	NC	F.D	L re	eg2,	reg	;3																									
[Operation]	reg	3 ←	– trı	ınc	reg2	2 (d	ou	ble		→ lo	ng	g-v	vor	d)																		
[Format]	For	ma	t F:	Ι																												
[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	0	1	0	0
		re	eg2							ĺ									reg3	;			са	iteg	ory	ty	'ne		sul	o-op		
[Description]	the stor The Wh ran If in reg foll	reg rest e rest nen ge ( nva iste	stru siste the 1 sult the s of 2 <sup>o</sup> lid c r is s, ac urce	r pa resu is re soun <sup>53</sup> – oper set a	ir sp lt in ce c 1 to atio as a ding	peci i the ded pper -2 on ex n in g to	fie re in can <sup>63</sup> , xce val dif	ed b egis the d is an epti lid	y § ster e ze s in IE ion op ren	gen r pæ ero nfin ZEE s a era ces	di di di ettic s a	al- sp irec e o 54- nc on mc	pu eci ctic r no de de ano ong	rpc fie on, ot-a fin nal d na g so	ose d re a-n ed ble o e	e re by ga nui l in ed, exc exc	egi ge urd mb nva , th cep es.	iste ene les oer dic e p	er re eral s o: , or l op pres	eg2 -pt f th wh pera ser pcc	2 to urpone c nen nen atic	64 ose curr the on e	-bi rej en erc exc bit	t fi gis t ro oun ept	ter oun dec tior	d-p reg dir d re n is	ooir g3. ng 1 esu s de of t	nt fo moo lt is stec he	de. s ou cted	nat, utsi I. SR	, an de	d
	•	Sou	irce	is a	neg	gativ	ve	nuı	nb	er,	no	ot-a	ı-n	um	ıbe	er,	or	-0	o: –	-26	<sup>3</sup> is	ret	urr	ned								



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	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.77 **TRNCF.DUL**

<Floating-point instruction>

TRNCF.DU	IL			Flo	ating	g-poi	int Cor	nvert	Do	uble to	Un	signe	d-Lo	ng, ro	und tov	vard	zero (	Doul	ble)
							0	Conv	ersi	on to ı	Insig	ned	fixed	-point	format	(dou	ble p	ecisi	ion)
[Instruction format]	TRNCF.DUL	reg2, reg	g3																
[Operation]	reg3 ← trunc	reg2 (dou	$able \rightarrow$	unsi	igne	ed l	ong-	WO	rd)										
[Format]	Format F: I																		
[Opcode]																			
	15 11	10	5	4			0	31			27	26	25	23	22 21	20		17	16
	rrrr O	1 1 1	1 1 1	1	0	0	0 1	w	W	w w	0	1	0	0 0	1 0	1	0 1	. 0	0
	reg2								I	reg3			cate	egory	type	Ì	sub-o	р	
[Description]	This instruction the register participation format, and st	ir specifi	ed by ge	ener	al-p	our	pose	reg	iste	r reg	2 to	o un	sign	ned 6	4-bit	fixe	d-po	int	
	The result is r	ounded in	n the zer	o di	irec	tio	n, reg	gard	lles	s of t	he	curr	ent	roun	ding	mod	e.		
	When the sou result is outsic detected.	-									-								
	If invalid oper register is set follows, accor	as an inva	alid oper	ratio	on a	nd	no e	xce	-										
	• Source is a	a positive	number	out	tsid	e tł	ne rai	nge	of	2 <sup>64</sup> –	1 to	o 0,	or +	⊦∞: 2	$2^{64} - 1$	is r	etur	ned.	•

• Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.78 TRNCF.DUW

<Floating-point instruction>

TRNCF.DUW

Floating-point Convert Double to Unsigned-Word, round toward zero (Double)

Conversion to unsigned fixed-point format (double precision)

[Instruction format]	TRNCF.DUW reg2, reg3
[Operation]	reg3 $\leftarrow$ trunc reg2 (double $\rightarrow$ unsigned word)
[Format]	Format F: I

[Opcode]

1	5				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	1	0	0	0	1	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
			reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to unsigned 32-bit fixed-point<br/>format, and stores the result in general-purpose register reg3.

The result is rounded in the zero 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  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.79 TRNCF.DW

**TRNCF.DW** 

<Floating-point instruction>

Floating-point Convert Double to Word, round toward zero (Double)

Conversion to fixed-point format (double precision)

TRNCF.DW reg2, reg3
reg3 $\leftarrow$ trunc reg2 (double $\rightarrow$ word)

[Format] Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
r	2	r	r	r	0	1	1	1	1	1	1	0	0	0	0	1	w	w	w	w	w	1	0	0	0	1	0	1	0	0	0	0
l		I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	o-op		$\overline{ }$

[Description]This instruction arithmetically converts the double-precision floating-point format contents of<br/>the register pair specified by general-purpose register reg2 to 32-bit fixed-point format, and<br/>stores the result in general-purpose register reg3.

The result is rounded in the zero 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  $2^{31} - 1$  to  $-2^{31}$ , 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  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  is returned.



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	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



#### 2.4.4.80 TRNCF.SL

TRNCF.SL

<Floating-point instruction>

Floating-point Convert Single to Long, round toward zero (Single)

Conversion to fixed-point format (single precision)

[Instruction format]	TRNCF.SL reg2, reg3
[Operation]	$reg3 \leftarrow trunc reg2 (single \rightarrow long-word)$

[Format] Format F: I

[Opcode]

15	,				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
r	J	r	r	r	r	1	1	1	1	1	1	0	0	0	0	1	w	w	w	w	0	1	0	0	0	1	0	0	0	1	0	0
		r	eg2	2															reg3	3			ca	teg	ory	ty	pe		sub	o-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point format contents of 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 zero 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  $2^{63} - 1$  to  $-2^{63}$ , 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  $+\infty$ :  $2^{63} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{63}$  is returned.



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	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.4.4.81 TRNCF.SUL

**TRNCF.SUL** 

<Floating-point instruction>

Floating-point Convert Single to Unsigned-Long, round toward zero (Single)

Conversion to unsigned fixed-point format (single precision)

[Instruction format]	TRNCF.SUL reg2	2, reg3									
[Operation]	reg3 ← trunc reg2	$2 \text{ (single} \rightarrow$	unsigne	ed long-w	vord)						
[Format]	Format F: I										
[Opcode]											
	15 11 10		54	0	31	27 26	6 25 23	22 21	20	1	7 16
	rrrr 1 1	1 1 1 1	1 1 0	0 0 1	w w w w	0 1	0 0 0	1 0	0	0 1 0	0 0
	reg2				reg3		category	type	:	sub-op	
[Description]	This instruction ari general-purpose re the register pair sp	egister reg2	to unsig	gned 64-b	oit fixed-poi	nt for	• •				
	The result is round	led in the z	ero direo	ction, reg	ardless of th	ne cui	rent rour	iding 1	nod	e.	
	When the source operand is infinite, not-a-number, or negative value, or when the rounded result is outside the range of $2^{64} - 1$ 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 posi	itive numb	er outsic	le the ran	ge of $2^{64} - 1$	l to 0	, or $+\infty$ : 2	$2^{64} - 1$	is r	eturned	l.
	• Source is a nega	ative numb	er, not-a	a-number	, or –∞: 0 is	retur	ned.				



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.



#### 2.4.4.82 TRNCF.SUW

<Floating-point instruction>

TRNCF.SUW

Floating-point Convert Single to Unsigned-Word, round toward zero (Single)

Conversion to unsigned fixed-point format (single precision)

[Instruction format]	TRNCF.SUW reg2, reg3
[Operation]	$reg3 \leftarrow trunc reg2 (single \rightarrow unsigned word)$
[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	0	0	0	1	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
Ī		I	reg2	2															reg3	3			ca	tego	ory	ty	pe		sub	-op		$\overline{ }$

[Description]This instruction arithmetically converts the single-precision floating-point number format<br/>contents of general-purpose register reg2 to unsigned 32-bit fixed-point format, and stores the<br/>result in general-purpose register reg3.

The result is rounded in the zero 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  $2^{32} - 1$  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  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.



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.



#### 2.4.4.83 TRNCF.SW

**TRNCF.SW** 

<Floating-point instruction>

Floating-point Convert Single to Word, round toward zero (Single)

Conversion to fixed-point format (single precision)

[Instruction format]	TRNCF.SW reg2, reg3
[Operation]	reg3 $\leftarrow$ trunc reg2 (single $\rightarrow$ word)

[Format] Format F: I

[Opcode]

1	5				11	10					5	4				0	31				27	26	25		23	22	21	20			17	16
ı	2	r	r	r	r	1	1	1	1	1	1	0	0	0	0	1	w	w	w	w	w	1	0	0	0	1	0	0	0	0	0	0
		I	reg2	2															reg3	3			ca	teg	ory	ty	pe		sub	-op		$\overline{ }$

[Description] This instruction arithmetically converts the single-precision floating-point number format contents of general-purpose register reg2 to 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the zero 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  $2^{31} - 1$  to  $-2^{31}$ , 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  $+\infty$ :  $2^{31} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ :  $-2^{31}$  is returned.



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	teger)	0 (Int	eger)	Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.



## 2.5 Extended Floating-point Instructions

Extended floating-point instructions comprise the following two groups of instructions:

• Extended floating-point vector arithmetic instructions

A group of instructions that perform arithmetic operations on vector data.

• Extended floating-point vector manipulation instructions

A group of instructions that load, store, and move vector data.

## 2.5.1 Instruction Format

Extended floating-point instructions are represented in both 32- and 48-bit formats. They are allocated in memory as follows:

- Lower-order part of the instruction in both formats (including bit  $0) \rightarrow On$  the lower-order side of the address
- Higher-order part of the instruction in both formats (including bits 15, 31 or 47) → On the higher-order side of the address

#### (1) Format M: 2OP

A 32-bit instruction format that has a 6-bit opcode field and 2 vector register specification fields and that combines the other bits in the sub-opcode field.



### (2) Format M: 3OP

A 32-bit instruction format that has a 6-bit opcode field and 3 vector register specification fields and that combines the other bits in the sub-opcode field.



#### (3) Format M: 4OP

A 48-bit instruction format that has a 6-bit opcode field and 4 vector register specification fields and that combines the other bits in the sub-opcode field.





#### (4) Format M: imm12

A 48-bit instruction format that has a 6-bit opcode field, 3 vector register specification fields, and a 12-bit immediate field and that combines the other bits in the sub-opcode field.

The highest-order bit of the immediate field is allocated to the sub-opcode field.



### (5) Format M: D

A 48-bit instruction format that has a 6-bit opcode field, 2 general-purpose register specification fields, a vector register specification field, and a 16-bit displacement field and that combines the other bits in the sub-opcode field.





## 2.5.2 Extended Floating-point Instruction Set

This section describes the following items in each instruction (based on alphabetical order of instruction mnemonics).

- Instruction format: Indicates the formats of the instruction and its operand(s) (see **Table 2.16** for symbols).
- Operation: Indicates the function of the instruction (see **Table 2.17** for symbols).
- Format: Indicates the instruction format of the instruction by instruction format name (see **Section 2.5.1, Instruction Format**).
- Opcode: Indicates the bit fields of the instruction opcode (see **Table 2.18** for symbols).
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.

Symbol	Meaning
reg n	General-purpose register
imm ×	×-bit immediate data
disp ×	×-bit displacement data
wreg1	Vector register (used as source register)
wreg2	Vector register (used as source register)
wreg3	Vector register (primarily used as the destination register; also as the source register in some instructions.)
wreg4	Vector register (Used as the destination register.)

#### Table 2.16 Instruction Format Legends



### Table 2.17 Operation Legends

Symbol	Meaning
←	Assignment
GR [a]	Value stored in general-purpose register a
WR [a]	Value stored in vector register a
CheckException (a)	Checks the conditions for generating the exception <i>a</i> and, if one is detected, suspends the instruction execution and performs exception processing.
result	Reflect result in flags
zero-extend (n)	Zero-extends <i>n</i> to word
sign-extend (n)	Sign-extends <i>n</i> to word
load-memory (a, b)	Reads data of size <i>b</i> from address <i>a</i>
store-memory (a, b, c)	Writes data b of size c to address a
abs ( <i>n</i> )	Absolute value of n
ceil (n)	Rounds <i>n</i> toward + $\infty$
cvt ( <i>n</i> )	Converts type of <i>n</i> according to rounding mode
floor ( <i>n</i> )	Rounds <i>n</i> toward $-\infty$
max ( <i>a, b</i> )	Maximum value of a and b
min ( <i>a, b</i> )	Minimum value of <i>a</i> and <i>b</i>
neg ( <i>n</i> )	Sign inversion of <i>n</i>
round (n)	Rounds <i>n</i> to closest value
sqrt (n)	Square root of <i>n</i>
trunc (n)	Rounds <i>n</i> to zero
fma ( <i>a, b, c</i> )	Result of multiplying a and b and then adding c
fms ( <i>a, b, c</i> )	Result of multiplying a and b and then subtracting c
Halfword	Halfword (16 bits)
Word	Word (32 bits)
Double-word	Double-word (64 bits)
Quad-word	Quad-word (128 bits)
+	Add
-	Subtract
×	Multiply
÷	Divide
==	Match (true upon a match)
!=	Mismatch (true upon a mismatch)
( <i>n</i> : <i>m</i> ) or ( <i>n</i> )	Bit selection
(h <i>x</i> )	x'th halfword element selected from vector data ( $x = 0$ to 7). h7 = (127:112), h6 = (111:96), h5 = (95:80), h4 = (79:64), h3 = (63:48), h2 = (47:32), h1 = (31:16) h0 = (15:0)
(w <i>x</i> )	x'th word element selected from vector data ( $x = 0$ to 3). w3 = (127:96), w1 = (95:64), w1 = (63:32), w0 = (31:0)
(dw <i>x</i> )	x'th double-word element selected from vector data ( $x = 0$ to 1). dw1 = (127:64), dw0 = (63:0)

Table 2.18	Legends for Opcodes
Symbol	Meaning
R	1-bit data of code specifying wreg1 or regID
r	1-bit data of code specifying wreg2
W	1-bit data of code specifying wreg3
W	1-bit data of code specifying wreg4
d	1-bit data of displacement
1	1-bit data of immediate (indicates higher bits of immediate)
i	1-bit data of immediate

## Table 2.18 Legends for Opcodes



## 2.5.3 Overview of the Extended Floating-point Vector Manipulation Instructions

#### (1) Vector Data Copy Instruction

• MOVV.W4: Move vector register to vector register

#### (2) Data Rearrangement Instructions

- FLPV.S4: Floating-point SIMD Flip (single)
- SHFLV.W4: Vector Shuffle

#### (3) Load to Vector Register Instructions

- LDV.DW: Load Vector (Double-Word)
- LDV.QW: Load Vector (Quad-Word)
- LDV.W: Load Vector (Word)
- LDVZ.H4: Load Vector at Even Halfword field

#### (4) Store from Vector Register Instructions

- STV.DW: Store Vector (Double-Word)
- STV.QW: Store Vector (Quad-Word)
- STV.W: Store Vector (Word)
- STVZ.H4: Store Vector at Even Halfword field

#### (5) Comparison/Conditional Move Instructions

- CMOVF.W4: Floating-point SIMD Conditional Move
- TRFSRV.W4: Transfers compare result to PSW



## 2.5.4 Overview of the Extended Floating-point Vector Arithmetic Instructions

These instructions perform floating-point arithmetic operations on vector data.

The available instructions (mnemonics) are listed below.

#### (1) Basic Arithmetic Instructions

- ABSF.S4: Floating-point SIMD Absolute (single)
- ADDF.S4: Floating-point SIMD Add (single)
- DIVF.S4: Floating-point SIMD Divide (single)
- MAXF.S4: Floating-point SIMD Maximum (single)
- MINF.S4: Floating-point SIMD Minimum (single)
- MULF.S4: Floating-point SIMD Multiply (single)
- NEGF.S4: Floating-point SIMD Negative (single)
- RECIPF.S4: Floating-point SIMD Reciprocal (single)
- RSQRTF.S4: Floating-point SIMD Reciprocal Square-Root (single)
- SQRTF.S4: Floating-point SIMD Square-Root (single)
- SUBF.S4: Floating-point SIMD Subtract (single)

#### (2) Extended Basic Operation Instructions

- FMAF.S4: Floating-point SIMD Fused-Multiply-Add (Single)
- FMSF.S4: Floating-point SIMD Fused-Multiply-Subtract (Single)
- FNMAF.S4: Floating-point SIMD Fused-Negative-Multiply-Add (Single)
- FNMSF.S4 Floating-point SIMD Fused-Negative-Multiply-Subtract (Single)

#### (3) Compound Arithmetic Instructions

- ADDSUBF.S4: Floating-point SIMD Add/Subtract (single)
- ADDSUBNF.S4: Floating-point SIMD Add/Subtract Negative (single)
- SUBADDF.S4: Floating-point SIMD Subtract/Add (single)
- SUBADDNF.S4: Floating-point SIMD Subtract/Add Negative (single)

#### (4) Exchange Arithmetic Instructions

- ADDXF.S4: Floating-point SIMD Add Exchange (single)
- MULXF.S4: Floating-point SIMD Multiply Exchange (single)
- SUBXF.S4: Floating-point SIMD Subtract Exchange (single)



#### (5) Compound Exchange Arithmetic Instructions

- ADDSUBNXF.S4: Floating-point SIMD Add/Subtract Negative Exchange (single)
- ADDSUBXF.S4: Floating-point SIMD Add/Subtract Exchange (single)
- SUBADDNXF.S4: Floating-point SIMD Subtract/Add Negative Exchange (single)
- SUBADDXF.S4: Floating-point SIMD Subtract/Add Exchange (single)

#### (6) Reduction Arithmetic Instructions

- ADDRF.S4: Floating-point SIMD Add Reduction (single)
- MAXRF.S4: Floating-point SIMD Maximum Reduction (single)
- MINRF.S4: Floating-point SIMD Minimum Reduction (single)
- MULRF.S4: Floating-point SIMD Multiply Reduction (single)
- SUBRF.S4: Floating-point SIMD Subtract Reduction (single)

#### (7) Conversion Instructions

- CEILF.SUW4: Floating-point SIMD Convert Single to Unsigned Word, round toward positive (single)
- CEILF.SW4: Floating-point SIMD Convert Single to Word, round toward positive (single)
- CVTF.HS4: Floating-point SIMD Convert Half to Single (single)
- CVTF.SH4: Floating-point SIMD Convert Single to Half (single)
- CVTF.SUW4: Floating-point SIMD Convert Single to Unsigned Word (single)
- CVTF.SW4: Floating-point SIMD Convert Single to Word (single)
- CVTF.UWS4: Floating-point SIMD Convert Unsigned Word to Single (single)
- CVTF.WS4: Floating-point SIMD Convert Word to Single (single)
- FLOORF.SUW4: Floating-point SIMD Convert Single to Unsigned Word, round toward negative (single)
- FLOORF.SW4 Floating-point SIMD Convert Single to Word, round toward negative (single)
- ROUNDF.SUW4: Floating-point SIMD Convert Single to Unsigned Word, round to nearest (single)
- ROUNDF.SW4: Floating-point SIMD Convert Single to Word, round to nearest (single)
- TRNCF.SUW4: Floating-point SIMD Convert Single to Unsigned Word, round toward zero (single)
- TRNCF.SW4: Floating-point SIMD Convert Single to Word, round toward zero (single)

#### (8) Comparison Instruction

• CMPF.S4: Floating-point SIMD Comparison (single)



## 2.5.4.1 ABSF.S4

ABSF.S4

<Extended Floating-point Instructions>

Floating-point Absolute Value (Single)

Floating-point absolute value (single)

[Instruction format]	ABSF.S4 wreg2, wreg3							
[Operation]	$WR[wreg3](w3) \leftarrow abs(WR[wreg2](w3))$ $WR[wreg3](w2) \leftarrow abs(WR[wreg2](w2))$ $WR[wreg3](w1) \leftarrow abs(WR[wreg2](w1))$							
[Format]	WR[wreg3](w0) ← abs(WR[wreg2](w0)) Format M: 2OP							
[Opcode]								
	15         11         10         5         4         0         31         27         26         25         23         22         17         16							
	r r r r r 1 1 1 1 1 1 1 0 0 0 0 w w w w w 1 0 1 1 0 1 0							
	wreg2 sub-op wreg3 category sub-op							
[Descriptions]	Takes an absolute value of the single-precision floating-point number in the vector register wreg2 and stores the result in the vector register wreg3.							
[Floating-point operation exceptions]	None							
[Supplement]	The subnormal number input will not be flushed even if the FS bit of FXSR is set to 1.							



#### 2.5.4.2 ADDF.S4

ADDF.S4	Floating-point SIMD Add (single)											
ADDF.34	Extended floating-point add (single precision)											
[Instruction format]	ADDF.S4 wreg1, wreg2, wreg3											
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) + WR[wreg1](w3)$											
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) + WR[wreg1](w2)$											
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) + WR[wreg1](w1)$											
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) + WR[wreg1](w0)$											
[Format]	Format M: 3OP											
[Opcode]												
	15 11 10 5 4 0 31 27 26 25 23 22 17 16											
	r r r r r 1 1 1 1 1 1 R R R R R W W W W W 1 0 1 1 0 1 0 0 1 0 0											
	wreg2 wreg1 wreg3 category sub-op											
[Descriptions]	Adds the single-precision floating-point data elements in the vector register wreg2 and the single-precision floating point data elements in the vector register wreg1 and stores the results in the respective data elements of the vector register wreg3. The operation is executed as if it were of infinite precision and the result is rounded in accordance with the current rounding mode.											
[Floating-point operation	Unimplemented operation exception (E)											
exceptions]	Invalid operation exception (V)											
	Inexact exception (I)											
	Overflow exception (O)											
	Underflow exception (U)											

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		P	+Α			-∞		
+0		Б-	+ A			-80		
-0								
+∞					+∞	Q-NaN[V]		
-∞		-	∞		Q-NaN[V]	- 8		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When the FS bit of the FXSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



## 2.5.4.3 ADDRF.S4

<Extended Floating-point Instructions>

# ADDRF.S4

Floating-point SIMD Add Reduction (single)

Extended floating-point add reduction (single precision)

[Instruction format]	ADDRF.S4 wreg1, wreg2, wreg3										
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) + WR[wreg2](w2)$										
	$WR[wreg3](w2) \leftarrow WR[wreg1](w3) + WR[wreg1](w2)$										
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) + WR[wreg2](w0)$										
	$WR[wreg3](w0) \leftarrow WR[wreg1](w1) + WR[wreg1](w0)$										
[Format]	Format M: 3OP										
[Opcode]											
	15         11         10         5         4         0         31         27         26         25         23         22         17         16										
	rrrr 1 1 1 1 1 1 1 R R R R R W W W W W 1 0 1 1 0 1 1 0 1 0 0										
	wreg2 wreg1 wreg3 category sub-op										
[Descriptions]	Adds together the contents of the even-number elements and odd-number elements of the single-precision floating-point data in the vector register wreg2 and store the results in the odd-number element of the vector register wreg3. Adds together the contents of the even-number elements and odd-number elements of the										
	single-precision floating-point data in the vector register wreg1 and store the results in the even-number element of the vector register wreg3.										
	The operation is executed as if it were of infinite precision and the result is rounded in accordance with the current rounding mode.										
[Floating-point operation	Unimplemented operation exception (E)										
exceptions]	Invalid operation exception (V)										
	Inexact exception (I)										
	Overflow exception (O)										
	Underflow exception (U)										



#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

B*4 A*4	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		٨	. D			-∞		
+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.

**Note 2.** When the FS bit of the FXSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

**Note 4.** Refer to [Operation] for the operands A and B that are input to produce the output.



## 2.5.4.4 ADDSUBF.S4

<Extended Floating-point Instructions>

# ADDSUBF.S4

Floating-point SIMD Add/Subtract (single)

Extended floating-point add/subtract (single precision)

[Instruction format]	ADDSUBF.S4 wreg1, wreg2, wreg3										
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) + WR[wreg1](w3)$										
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) - WR[wreg1](w2)$										
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) + WR[wreg1](w1)$										
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) - WR[wreg1](w0)$										
[Format]	Format M: 3OP										
[Opcode]											
	15         11         10         5         4         0         31         27         26         25         23         22         17         16										
	r r r r r 1 1 1 1 1 1 R R R R R w w w w 1 0 1 1 1 0 1 0 0 0 0										
	wreg2 wreg1 wreg3 category sub-op										
[Descriptions]	Adds together the contents of single-precision floating-point data in the odd-number element of the vector register wreg2 and the contents of the single-precision floating-point data in the odd-number element of the vector register wreg1 and store the results in the odd-number element of the vector register wreg3.										
	Subtracts the contents of single-precision floating-point data in the even-number element of the vector register wreg1 from the contents of single-precision floating-point data in the even-number element of the vector register wreg2 and stores the results in the even-number element of the vector register wreg3.										
	The operation is executed as if it were of infinite precision and the result is rounded in accordance with the current rounding mode.										
[Floating-point operation	Unimplemented operation exception (E)										
exceptions]	Invalid operation exception (V)										
	Inexact exception (I)										
	Overflow exception (O)										
	Underflow exception (U)										



[Operation result]

<Odd-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		B-				-∞		
+0		DT	FA			-8		
-0								
+∞					+∞	Q-NaN[V]		
-∞		-	×		Q-NaN[V]	- 80		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

<Even-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal	B – A							
-Normal					+∞	-∞		
+0								
-0								
+∞	-∞				Q-NaN[V]			
-∞	+∞					Q-NaN[V]		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.


# 2.5.4.5 ADDSUBNF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Add/Subtract Negative (single)

Extended floating-point Add/Subtract Negative (single precision)

[Instruction format]	ADDSUBNF.S4 wreg1, wreg2, wreg3								
[Operation]	$WR[wreg3](w3) \leftarrow neg(WR[wreg2](w3) + WR[wreg1](w3))$ $WR[wreg3](w2) \leftarrow neg(WR[wreg2](w2) - WR[wreg1](w2))$ $WR[wreg3](w1) \leftarrow neg(WR[wreg2](w1) + WR[wreg1](w1))$ $WR[wreg3](w0) \leftarrow neg(WR[wreg2](w0) - WR[wreg1](w0))$	$R[wreg3](w2) \leftarrow neg(WR[wreg2](w2) - WR[wreg1](w2))$ $R[wreg3](w1) \leftarrow neg(WR[wreg2](w1) + WR[wreg1](w1))$							
[Format]	Format M: 3OP								
[Opcode]									
	15     11     10     5     4     0     31     27     26     25     23     22	17 16							
	rrrr111111RRRRRWWWW101111	0 1 1 0 0 0							
	wreg2 wreg1 wreg3 category	sub-op							
[Descriptions]	Adds together the contents of single-precision floating-point data in the odd element of the vector register wreg2 and the contents of single-precision flo in the odd-number element of the vector register wreg1 and stores the result number element of the vector register wreg3 with their sign inverted.	pating-point data							
	Subtracts the contents of single-precision floating-point data in the even-nu the vector register wreg1 from the contents of floating-point data in the even element of the vector register wreg2 and stores the results in the even-numb the vector register wreg3 with their sign inverted.	en-number							
	The operation is executed as if it were of infinite precision and the result is rounded in accordance with the current rounding mode.								
[Floating-point operation	Unimplemented operation exception (E)								
exceptions]	Invalid operation exception (V)								
	Inexact exception (I)								
	Overflow exception (O)								
	Underflow exception (U)								



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

#### <Odd-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(D						
+0		-(В	+ A)			+∞		
-0								
+∞					-∞	Q-NaN[V]		
-∞		+	×		Q-NaN[V]	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Even-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(D	•		-∞			
+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.

**Note 2.** When the FS bit of the FXSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.



# 2.5.4.6 ADDSUBNXF.S4

ADDSUBN	YE SA		Flo	eating-point SIME	) Add	/Sub	otract Ne	egative	Exchange	) (sing	jle)
ADDSUDIA	AI .34		Extended float	ing-point add/su	otract	neg	ative ex	chang	e (single p	recisi	on)
[Instruction format]	ADDSUBNXF.S4	wreg1, wreg	2, wreg3								
[Operation]	$WR[wreg3](w3) \leftarrow neg(WR[wreg2](w3) + WR[wreg1](w2))$										
	$WR[wreg3](w2) \leftarrow neg(WR[wreg2](w2) - WR[wreg1](w3))$										
	WR[wreg3](w1)	- neg(WR[w	reg2](w1) + W	/R[wreg1](w	((0						
	WR[wreg3](w0)	- neg(WR[w	reg2](w0) – W	/R[wreg1](w	1))						
[Format]	Format M: 3OP										
[Opcode]											
	15 11 10	5	4 0	31 :	27 26	5 25	23	22		17	16
	r r r r r 1	1 1 1 1 1	RRRRR	wwww	w 1	0	1 1	1 0	1 1 1	1 0	0
	wreg2		wreg1	wreg3		Ca	ategory		sub-op		
[Descriptions]	Adds together the element of the vec in the even-number number element o	etor register w er element of t	reg2 and the c the vector regi	ontents of sin ster wreg1 a	ngle nd st	-pre	ecisior s the 1	n floa	ting-poi		
	Subtracts the cont the vector register element of the vec the vector register	wreg1 from t tor register w	he contents of reg2 and store	floating-points the results	nt da	ata i	in the	even-	number	•	
	The operation is e accordance with the second secon			ite precision	and	the	e resul	t is ro	ounded i	n	
[Floating-point operation	Unimplemented o	peration exce	ption (E)								
exceptions]	Invalid operation	exception (V)									
	Inexact exception	(I)									
	Overflow exception	on (O)									



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

#### <Odd-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(D				1.00		
+0		-(Б	+ A)			+∞		
-0								
+∞					-∞	Q-NaN[V]		
-∞		+	∞		Q-NaN[V]	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Even-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(D	•		-∞			
+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.

**Note 2.** When the FS bit of the FXSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.



# 2.5.4.7 ADDSUBXF.S4

<Extended Floating-point Instructions>

# ADDSUBXF.S4

Floating-point SIMD Add/Subtract Exchange (single)

Extended floating-point add//subtract exchange (single precision)

[Instruction format]	ADDSUBXF.S4 wreg1, wreg2, wreg3							
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) + WR[wreg1](w2)$							
	$R[wreg3](w2) \leftarrow WR[wreg2](w2) - WR[wreg1](w3)$							
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) + WR[wreg1](w0)$							
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) - WR[wreg1](w1)$							
[Format]	ormat M: 3OP							
[Opcode]								
	15         11         10         5         4         0         31         27         26         25         23         22         17         16							
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 1 0 1 0 1 0 0							
	wreg2 wreg1 wreg3 category sub-op							
[Descriptions]	Adds together the single-precision floating-point format contents in the odd-number elements of vector register wreg2 and the single-precision floating-point format contents in the even-number elements of vector register wreg1 and stores the results in the odd-number elements of vector register wreg3. Subtract the single-precision floating-point format contents in the odd-number elements of vector register wreg1 from the single-precision floating-point format contents in the even- number elements of vector register wreg2 and stores the results in the even- number elements of vector register wreg3. 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)							



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

#### <Odd-number element>

	0.0.1.0.1.0							
wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		P						
+0		В-	+ A			-∞		
-0								
+∞					+∞	Q-NaN[V]		
-∞		-	~		Q-NaN[V]	- 8		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Even-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		B·						
+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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.8 ADDXF.S4

<Extended Floating-point Instructions>

# ADDXF.S4

Floating-point SIMD Add Exchange (single)

Extended floating-point add exchange (single precision)

[Instruction format]	ADDXF.S4 wreg1, wreg2, wreg3							
[Operation]	$R[wreg3](w3) \leftarrow WR[wreg2](w3) + WR[wreg1](w2)$ $R[wreg3](w2) \leftarrow WR[wreg2](w2) + WR[wreg1](w3)$ $R[wreg3](w1) \leftarrow WR[wreg2](w1) + WR[wreg1](w0)$ $R[wreg3](w0) \leftarrow WR[wreg2](w0) + WR[wreg1](w1)$							
[Format]	Format M: 3OP							
[Opcode]	15 11 10 5 4 0 31 27 26 25 23 22 17 16							
	rrrr111111RRRR     wwwww10111000100							
	wreg2         wreg1         wreg3         category         sub-op							
[Descriptions]	Adds together the single-precision floating-point format contents in the odd-number elements of vector register wreg2 and the single-precision floating-point format contents in the even-number elements of vector register wreg1 and stores the results in the odd-number elements of vector register wreg3.							
	Adds together the single-precision floating-point format contents in the even-number elements of vector register wreg2 and the single-precision floating-point format contents in the odd-number elements of vector register wreg1 and stores the results in the even-number elements of vector register wreg3.							
	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	Unimplemented operation exception (E)							
exceptions]	Invalid operation exception (V)							
	Inexact exception (I)							
	Overflow exception (O)							
	Underflow exception (U)							



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		Р	+ A			-∞		
+0		D-	+ A			-80		
-0								
+∞					+∞	Q-NaN[V]		
-∞		-	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		Q-NaN[V]	-∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.9 CEILF.SUW4

<Extended Floating-point Instructions>

	Floating-point SIMD Convert Single to Unsigned Word, round toward positive (single)											
CEILF.SUV												
	Extended floating-point type conversion (Single precision → unsigned integer)											
[Instruction format]	CEILF.SUW4 wreg2, wreg3											
[Operation]	$WR[wreg3](w3) \leftarrow ceil(WR[wreg2](w3) (single \rightarrow unsigned word))$											
	$WR[wreg3](w2) \leftarrow ceil(WR[wreg2](w2) (single \rightarrow unsigned word))$											
	$WR[wreg3](w1) \leftarrow ceil(WR[wreg2](w1) \text{ (single} \rightarrow unsigned word))$											
	$WR[wreg3](w0) \leftarrow ceil(WR[wreg2](w0) (single \rightarrow unsigned word))$											
[Format]	Format M: 3OP											
[Opcode]												
	15         11         10         5         4         0         31         27         26         25         23         22         17         16											
	rrrr         1         1         1         1         0         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0         1         0											
	wreg2 sub-op wreg3 category sub-op											
[Descriptions]	Arithmetically converts the single-precision floating-point format contents in the elements of vector register wreg2 to an unsigned 32-bit integer format and stores the results in the respective elements of vector register wreg3. The results are rounded toward $+\infty$ 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 $2^{32} - 1$ to 0, an IEE754-defined invalid operation exception is detected.											
	If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FXSR register is set as an invalid operation and no exception occurs. The return value differs according to the value of the source operand as follows:											
	• Source is a positive number outside the range of $2^{32} - 1$ to 0 or $+\infty$ : $2^{32} - 1$ is returned.											
	• Source is a negative number, not-a-number, or $-\infty$ : 0 is returned.											
[Floating-point operation	Unimplemented operation exception (E)											
exceptions]	Invalid operation exception (V)											
	Inexact exception (I)											



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

### [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.10 CEILF.SW4

CEILF.SW4	1				Flo	ating	J-point	SIM	DO	Convert S	Singl	le to	Wo	ord, ro	Jund	tow	ard	oosit	ive (	ingle)
	+					E	xtende	d flo	oati	ing-point	type	e cor	nvei	rsion	(Sin	gle p	oreci	sion	$\rightarrow$ lr	teger)
[Instruction format]	CEILF.SW4 v	vreg2, v	vreg3																	
[Operation]	WR[wreg3](w	3) ← c	eil(WF	R[wr	eg2	2](w	3) (si	ngl	e -	$\rightarrow$ wor	d))									
	WR[wreg3](w	2) ← c	eil(WF	R[wr	eg2	2](w	2) (si	ngl	e -	$\rightarrow$ wor	d))									
	WR[wreg3](w	1) ← c	eil(WF	R[wr	eg2	2](w	1) (si	ngl	e -	$\rightarrow$ wor	d))									
	WR[wreg3](w	0) ← c	eil(WF	R[wr	eg2	2](w	0) (si	ngl	e -	$\rightarrow$ wor	d))									
[Format]	Format M: 30	Р																		
[Opcode]																				
	15 11	10		5	4		C	) 31	1		27	26	25		23 2	22				17 16
	rrrr	1 1 1	1 1	1	0 (	0 1	0 0	) w	7 1	w w w	w	1	0	1	1	0 1	LO	0	0	0 0
	wreg2					sub-	ор			wreg3			Ca	ategoi	у		รเ	ıb-op		
[Descriptions]	This instruction arithmetically converts the single-precision floating-point format contents of vector register wreg2 to 32-bit integer format, and stores the results in vector register wreg3.																			
	The results are	rounde	ed tow	ard +	-00 ]	rega	rdles	s of	f tl	he curr	ent	rou	inc	ling	mc	ode.				
	When the sourthe range of $2^3$	-																		
	If invalid oper register is set a according to th	is an in	valid o	pera	tioı	n an	d no	exc	ep	otion of										
	• Source is a	positiv	e num	ber o	r+	∞: 2	$2^{31} - 1$	l is	re	eturned	•									
	• Source is a	negativ	ve num	ber,	not	t-a-r	umb	er, (	or		2 <sup>31</sup> i	is re	etu	rned	l.					
[Floating-point operation	Unimplemente	ed opera	ation e	хсер	tioı	n (E	)													
exceptions]	Invalid operat	on exce	eption	(V)																
	Invalid operation exception (V) Inexact exception (I)																			



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg	2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Oper res [exce		A (Int	eger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.11 CMOVF.W4

<Extended Floating-point Instructions>

# CMOVF.W4

Floating-point SIMD Conditional Move

Conditional move of vector register (Single precision)

[Instruction format]	CMOVF.W4 wreg4, wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow (WR[wreg4](w3) != 0) ? WR[wreg1](w3): WR[wreg2](w3)$
	$WR[wreg3](w2) \leftarrow (WR[wreg4](w2) != 0) ? WR[wreg1](w2): WR[wreg2](w2)$
	$WR[wreg3](w1) \leftarrow (WR[wreg4](w1) != 0) ? WR[wreg1](w1): WR[wreg2](w1)$
	$WR[wreg3](w0) \leftarrow (WR[wreg4](w0) != 0) ? WR[wreg1](w0): WR[wreg2](w0)$
[Format]	Format M: 4OP
[Opcode]	
	<u>15 11 10 5 4 0 31 27 26 17 16</u>
	0 0 0 0 0 1 1 1 1 0 1 R R R R R W W W W 1 1 0 0 0 0 1 1 1 0 1
	sub-op wreg1 wreg3 sub-op
	47 43 42 37 36 32
	rrrr 0 0 0 0 0 0 W W W W W
	wreg2 sub-op wreg4
[Descriptions]	This instruction stores each element of vector register wreg1 in the corresponding element of vector register wreg3 if the value of the corresponding element of vector register wreg4 is set to a nonzero value. If the each element of vector register wreg4 is set to 0, the corresponding element of vector
	register wreg2 is stored in the corresponding element of vector register wreg3.
[Floating-point operation exceptions]	None

# CAUTION

Even when a nonzero value is set in bits 37 to 42, the opcode functions as the CMOVF.W4 instruction. An RIE exception does not occur in such cases.



# 2.5.4.12 CMPF.S4

<Extended Floating-point Instructions>

CMPF.S4	Floating-point SIMD Comparison (single)
	Extended floating-point comparison (Single precision)
[Instruction format]	CMPF.S4 fcond, wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow cmpf (fcond, WR[wreg2](w3), WR[wreg1](w3))$
	$WR[wreg3](w2) \leftarrow cmpf (fcond, WR[wreg2](w2), WR[wreg1](w2))$
	$WR[wreg3](w1) \leftarrow cmpf (fcond, WR[wreg2](w1), WR[wreg1](w1))$
	$WR[wreg3](w0) \leftarrow cmpf (fcond, WR[wreg2](w0), WR[wreg1](w0))$
	$cmpf (fcond, A, B) =$ if isNaN(A) or isNaN(B) 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 $\leftarrow A < B$ result.equal $\leftarrow A == B$ result.equal $\leftarrow A == B$ result.unordered $\leftarrow 0$ endif result.equal $\leftarrow A == B$ result.e
	(fcond[0] & result.unordered) return (result == true) ? FFFF FFFF <sub>H</sub> : $0000 \ 0000_{\text{H}}$
	$\operatorname{feath}(\operatorname{feath}) = \operatorname{free}(\operatorname{feath}) = f$

[Format]

Format M: 3OP

[Opcode]

15				11	10					5	4				0	31				27	26	25		23	22					17	16
r r	C	r	r	r	1	1	1	1	1	1	R	R	R	R	R	w	w	w	w	w	1	0	1	1	0	0	F	F	F	F	0
	w	reg	2									v	vreg	1			v	vreg	3			са	iteg	ory			sub	o-op			

**Remark:** FFFF = fcond.



[Descriptions] This instruction compares the single-precision floating-point format contents of vector register wreg1 with the single-precision floating-point format contents in each element of vector register wreg2 based on the comparison condition fcond. The result of comparison is stored in vector register wreg3.

For a description of the comparison condition fcond, see **Table 2.19 Comparison Conditions**.

If one of the values is not-a-number and the MSB of the comparison condition fcond is set, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are enabled, the comparison result is not set and processing proceeds with the processing of the exception.

If no enable bit is set, no exception is generated, the preservation bit (bit 4) of the FXSR register is set, and the comparison result is undefined.

For floating-point arithmetic instructions including comparison instructions, any SignalingNaN (S-NaN) received as an operand value is regarded as an invalid operation condition. If a comparison that results in an invalid operation not on only an S-NaN but also on QuietNaN (Q-NaN) is used, the programming code for handling an error caused by NaN can be made simpler. In other words, this dispenses with the code for explicitly checking for Q-NaN which would lead to the "Unordered" result. Instead, make an arrangement so that an exception should be generated upon detection of an invalid operation and the error processing be handled by the exception processing system. Shown below is an example of comparison in which two numeric values are compared for a relationship and an error is signaled when an Unordered result is detected.

Comparia Condition				Detection of Invalid operation
	fcond	Definition	Description	Exception by Unordered
F	0	FALSE	Always false	No
UN	1	Unordered	One of wreg1 and wreg2 is not-a-number	No
EQ	2	wreg2 = wreg1	Ordered (both wreg1 and wreg2 is not not-a-number) and equal	No
UEQ	3	wreg2 ?= wreg1	Unordered (at least, one of wreg1 and wreg2 is not-a-number) or equal	No
OLT	4	wreg2 < wreg1	Ordered (both wreg1 and wreg2 are not not-a-number) and less than	No
ULT	5	wreg2 ?< wreg1	Unordered (one of wreg1 and wreg2 is not-a-number) or less than	No
OLE	6	wreg2 ≤ wreg1	Ordered (both wreg1 and wreg2 are not not-a-number) and less than or equal to	No
ULE	7	wreg2 ?≤ wreg1	Unordered (one of wreg1 and wreg2 is not-a-number) or less than or equal to	No
SF	8	FALSE	Always false	Yes
NGLE	9	Unordered	One of wreg1 and wreg2 is not-a-number	Yes
SEQ	10	wreg2 = wreg1	Ordered (both wreg1 and wreg2 are not not-a-number) and equal	Yes
NGL	11	wreg2 ?= wreg1	Unordered (one of wreg1 and wreg2 is not-a-number) or equal	Yes
LT	12	wreg2 < wreg1	Ordered (both wreg1 and wreg2 are not not-a-number) and less than	Yes
NGE	13	wreg2 ?< wreg1	Unordered (one of wreg1 and wreg2 is not-a-number) or less than	Yes
LE	14	wreg2 ≤ wreg1	Ordered (both wreg1 and wreg2 are not not-a-number) and less than or equal to	Yes
NGT	15	wreg2 ?≤ wreg1	Unordered (one of wreg1 and wreg2 is not-a-number) or less than or equal to	Yes

# Table 2.19 Comparison Conditions

Note: ?: Unordered (invalid comparison)

# Example of explicitly checking any Q-NaN in each element

```
CMPF.S4
            OLT, wr12, wr14, wr15 # Check wr14 < wr12 for each
                                    element.
CMPF.S4
            UN,wr12,wr14,wr16
                                   # Check for Unordered condition for
                                    each element.
TRFSRV.W4
            4,wr15
                                    # Branch to L2 if true.
BT
             L2
TRFSRV.W4 5,wr16
ΒT
             ERROR
                                    # Branch to error processing if true.
```

# Put the code to be executed if there is an element that is not Unordered and for which wr14 < wr12 is not established.</pre>

```
L2:
```

# Put the code to be executed when wr14 < wr12 is established for all elements.

# Example of using the comparison for notifying Q-NaN

TRFSRV.W4 4,wr15

CMPF.S4 LT,wr12,wr14,wr15 # Check wr14 < wr12 for each element.

BT L2 # Branch to L2 if true.

# Put the code to be executed if there is an element that wr14 < wr12 is not established. L2:

# Put the code to be executed when wr14 < wr12 is established for all elements.

[Floating-point Invalid operation exception (V) operation exceptions]

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

# [Condition code (fcond) = 0 to 7]

wreg1(B) wreg2(A)	+Normal	Q-NaN	S-NaN											
±Normal														
±0	Stored	Stored in WR[wreg3] according to the result of comparison based on comparison conditions (fcond).												
±∞														
Q-NaN	Unorderd													
S-NaN	Unorderd[V]													

# [Condition code (fcond) = 8 to 15]

wreg1(B) wreg2(A)	+Normal	-Normal	Q-NaN	S-NaN										
±Normal														
±0	Stored	Stored in WR[wreg3] according to the result of comparison based on comparison conditions (fcond).												
±∞														
Q-NaN														
S-NaN		Unorderd[V]												

Note: [ ] indicates an exception that must occur.



# 2.5.4.13 CVTF.HS4

**CVTF.HS4** 

<Extended Floating-point Instructions>

Floating-point SIMD Convert Half to Single (single)

Extended floating-point type conversion (Half-precision  $\rightarrow$  Single precision)

[Instruction format]	CVTF.HS4 wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow cvt(WR[wreg2](h6) (half \rightarrow single))$ $WR[wreg3](w2) \leftarrow cvt(WR[wreg2](h4) (half \rightarrow single))$ $WR[wreg3](w1) \leftarrow cvt(WR[wreg2](h2) (half \rightarrow single))$ $WR[wreg3](w0) \leftarrow cvt(WR[wreg2](h0) (half \rightarrow single))$
[Format]	Format M: 2OP
[Opcode]	15 11 10 5 4 0 31 27 26 25 23 22 17 16
	rrrr       1       1       1       1       0       1       1       0       1       1       0       1       1       0       1       1       0       1       1       0       1       1       0       1       1       0       1       1       0       1       0
[Descriptions]	This instruction arithmetically converts the half-precision floating-point format contents in the lower-order 16 bits of each element of vector register wreg2 to single-precision floating-point format in the current rounding mode and stores the result in the corresponding element of vector register wreg3.
[Floating-point operation exceptions]	Invalid operation exception (V)

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



# RH850G4MH Software

# [Operation result]

wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Si	ngle)	+0	-0	+∞	- 8	Q-NaN	Q-NaN[V]

Note: [ ] indicates an exception that must occur.

[Supplement] This instruction can accurately convert data in all half-precision floating-point formats except not-a-number to single-precision floating-point format. Any subnormal number input is not flushed even when FXSR.FS = 1.



# 2.5.4.14 CVTF.SH4

CVTF.SH4	Floating-point SIMD Convert Single to Half (single
6717.3114	Extended floating-point type conversion (Single precision $\rightarrow$ Half-precision
[Instruction format]	CVTF.SH4 wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow zero-extend(cvt(WR[wreg2](w3) (single \rightarrow half)))$
	$WR[wreg3](w2) \leftarrow zero-extend(cvt(WR[wreg2](w2) (single \rightarrow half)))$
	$WR[wreg3](w1) \leftarrow zero-extend(cvt(WR[wreg2](w1) (single \rightarrow half)))$
	$WR[wreg3](w0) \leftarrow zero-extend(cvt(WR[wreg2](w0) (single \rightarrow half)))$
[Format]	Format M: 2OP
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 17 16
	r r r r r 1 1 1 1 1 1 0 1 1 0 1 w w w w w 1 0 1 1 0 1 0
	wreg2 sub-op wreg3 category sub-op
[Descriptions]	This instruction arithmetically converts the single-precision floating-point format contents in each element of vector register wreg2 to half-precision floating-point format in the current rounding mode. The results are zero-extended to word length and stored in the corresponding element of vector register wreg3.
[Floating-point operation	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Overflow exception (O)
	Underflow exception (U)

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (H	Half)	+0	-0	+∞	-∞	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.15 CVTF.SUW4

CVTF.SUW	I A						F	loa	ting	-point S	IMD	Con	vert	Sinę	jle to	ა Un	ısigr	ned	Word	(sing	jle)
					Exte	ended f	loating	j-po	oint 1	type cor	nversi	on (	Sin	gle p	recis	sion	$\rightarrow l$	Jnsi	gned i	nteg	jer)
[Instruction format]	CVTF.SUW4	wreg	2, wre	eg3																	
[Operation]	WR[wreg3](w WR[wreg3](w WR[wreg3](w WR[wreg3](w	2) ← 1) ←	cvt(V cvt(V	VR[w VR[w	reg reg	2](w 2](w	2) (si 1) (si	ng ng	le - le -	$\rightarrow$ uns $\rightarrow$ uns	igne igne	ed v ed v	voi voi	d)) d))							
[Format]	Format M: 20	Р																			
[Opcode]	15 11	10		E	4			0 :	21		2	7 26	- 25		22	22				17	16
	rrrr		1 1		4	1 0				w w			Т	1	-		1	0	0 0		16 0
	wreg2					sub				wreg3				ateg		   		sub-			<u> </u> 
[Descriptions]	Arithmetically of vector regis respective eler	ter w	reg2 t	o an u	insi	gned	32-t	oit i													ts
	When the sour rounded result exception is de	is ou	tside								-										
	If invalid oper register is set a according to th	is an	invali	d ope	rati	on ar	ıd no	ex	ce	ption	occu										
	• Source is a					-						to (	), o	r +0	<b>∞:</b> Ω	232	- 1	is	retur	ned	I.
	• Source is a	nega	tive n	umbe	r, n	ot-a-i	numt	oer,	, 01	: –∞: (	) is 1	etu	rne	ed.							
[Floating-point operation	Unimplemente	1	eratio	n exce	epti	on (E	5)														
· 1	Unimplemented operation exception (E) Invalid operation exception (V)																				
exceptions]	Invalid operat	-			-	(-	,														



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.16 CVTF.SW4

CVTF.SW4

<Extended Floating-point Instructions>

Floating-point SIMD Convert Single to Word (single)

Extended floating-point type conversion (Single precision  $\rightarrow$  Integer)

[Instruction format]	CVTF.SW4 w	vreg2, wreg3																		
[Operation]	WR[wreg3](w	$v3) \leftarrow cvt(W)$	'R[wr	eg2	2](w3	5) (sii	ıgle	:	• wo	rd))										
	WR[wreg3](w	$(v2) \leftarrow cvt(W)$	'R[wr	eg2	2](w2	l) (sii	ıgle	$\rightarrow$	• wo	rd))										
	WR[wreg3](w	$(v1) \leftarrow cvt(W)$	'R[wr	eg2	2](w1	) (siı	ıgle	$\rightarrow$	• wo	rd))										
	WR[wreg3](w	$v(0) \leftarrow cvt(W)$	'R[wr	reg2	2](w0	)) (sii	ıgle	: →	• wo	rd))										
[Format]	Format M: 2C	)P																		
[Opcode]																				
	-	10		1						27	1	-			22					16
	rrrr wreg2	1 1 1 1	1 1	0	1 0	0 0	) w	W	W	w w	1	0	1	1	0	1	0	0 0	0	0
	wreg2				sub-	ор		,	wreg3	3		С	atego	y		s	sub-	ор		
[Descriptions]	This instruction in vector regist wreg3. When the source	ster wreg2 to	32-b	it iı	ntege	r fori	nat	and	d sto	ores ti	he 1	es	sults	in	ve	ctor	re	giste	er	
	the range of 2	-																		
	If invalid oper register is set according to the	as an invalid	oper	atic	on and	d no	exce	ept	ion	occui										
	• Source is a	n positive nu	nber	or -	+∞: 2	<sup>31</sup> –	l is	ret	urne	ed.										
	• Source is a	negative nu	mber	, no	ot-a-n	umb	er, o	or –	-∞: -	-2 <sup>31</sup> i	s re	etu	irnec	1.						
[Floating-point operation	Unimplement	ed operation	excej	ptic	on (E)	)														
exceptions]	Invalid operat	ion exceptio	n (V)																	
	Inexact except	tion (I)																		



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg	2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Oper res [exce		A (Int	eger)	0 (Int	eger)	+Max Int[V]		-Max Int[V]	

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.17 CVTF.UWS4

<extended floating-point="" inst<="" th=""><th></th></extended>	
	Floating-point SIMD Convert Unsigned Word to Single (single)
CVTF.UWS	54
	Extended floating-point type conversion (Unsigned integer $\rightarrow$ Single precision)
[Instruction format]	CVTF.UWS4 wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow cvt(WR[wreg2](w3) \text{ (unsigned word} \rightarrow single))$
	$WR[wreg3](w2) \leftarrow cvt(WR[wreg2](w2) \text{ (unsigned word} \rightarrow single))$
	$WR[wreg3](w1) \leftarrow cvt(WR[wreg2](w1) \text{ (unsigned word} \rightarrow single))$
	$WR[wreg3](w0) \leftarrow cvt(WR[wreg2](w0) \text{ (unsigned word} \rightarrow single))$
[Format] [Opcode]	Format M: 2OP
	15     11     10     5     4     0     31     27     26     25     23     22     17     16
	r r r r r 1 1 1 1 1 1 0 1 0 1 1 w w w w w 1 0 1 1 0 1 0
	wreg2 sub-op wreg3 category sub-op
[Descriptions]	This instruction arithmetically converts the unsigned 32-bit integer format contents in each element of vector register wreg2 to single-precision floating-point format and stores the results in the corresponding element of vector register wreg3. The results are rounded in accordance with the current rounding mode.
[Floating-point operation exceptions]	Inexact exception (I)
NOTE	

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

wreg2(A)	+Integer	-Integer	0 (Integer)
Operation result [exception]	A (No	ormal)	+0



# 2.5.4.18 CVTF.WS4

CVTF.WS4

<Extended Floating-point Instructions>

Floating-point SIMD Convert Word to Single (single)

Extended floating-point type conversion (Integer  $\rightarrow$  Single precision)

[Instruction format]	CVTF.WS4 wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow cvt(WR[wreg2](w3) (word \rightarrow single))$ $WR[wreg3](w2) \leftarrow cvt(WR[wreg2](w2) (word \rightarrow single))$ $WR[wreg3](w1) \leftarrow cvt(WR[wreg2](w1) (word \rightarrow single))$ $WR[wreg3](w0) \leftarrow cvt(WR[wreg2](w0) (word \rightarrow single))$
[Format]	Format M: 2OP
[Opcode]	<u>15 11 10 5 4 0 31 27 26 25 23 22 17 16</u>
	r r r r r       1       1       1       1       1       0       1       0       1       0       1       0       1       0       1       0       1       0       1       0
[Descriptions]	This instruction arithmetically converts the 32-bit integer format contents in vector register wreg2 to single-precision floating-point format and stores the results in vector register wreg3. The results are rounded in accordance with the current rounding mode.
[Floating-point operation exceptions]	Inexact exception (I)

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

wreg2(A)	+Integer	-Integer	0 (Integer)
Operation result [exception]	A (No	ormal)	+0



# 2.5.4.19 DIVF.S4

<Extended Floating-point Instructions>

DIVF.S4	Floating-point Divide (Single)
DIVF.34	Extended floating-point divide (Single precision)
[Instruction format]	DIVF.S4 wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) \div WR[wreg1](w3)$
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) \div WR[wreg1](w2)$
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) \div WR[wreg1](w1)$
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) \div WR[wreg1](w0)$
[Format]	Format M: 2OP
[Opcode]	
	15 11 10 5 4 0 31 27 26 25 23 22 17 16
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 0 1 0 1 1 1 0
	wreg2 wreg1 wreg3 category sub-op
[Descriptions]	This instruction divides the single-precision floating-point format contents in each element of vector register wreg2 by the single-precision floating-point format contents in vector register wreg1 and stores the results in vector register wreg3. The operation is executed as if it were of infinite accuracy and the result is rounded in accordance with the current rounding mode.
	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Division by zero exception (Z)
	Overflow exception (O)
	Underflow exception (U)

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal		D			+∞	-8		
-Normal		В-	÷A		-∞	+∞		
+0		[7]	0.11		+∞	-∞		
-0	±∞	·[۷]	Q-Na	aN[V]	-∞	+∞		
+∞	+0	-0	+0	-0				
-∞	-0	+0	-0	+0	Q-Na	antvi		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.20 FLOORF.SUW4

<Extended Floating-point Instructions>

<extended floating-point<="" th=""><th>Instructions&gt;</th></extended>	Instructions>
FLOORF.	Floating-point SIMD Convert Single to Unsigned Word, round toward negative (single)
FLOOKF.	<b>SUVV4</b> Extended floating-point type conversion (Single precision → Unsigned integer)
[Instruction format]	FLOORF.SUW4 wreg2, wreg3
[Operation]	WR[wreg3](w3) $\leftarrow$ floor WR[wreg2](w3) (single $\rightarrow$ unsigned word)
	$WR[wreg3](w2) \leftarrow floor WR[wreg2](w2) (single \rightarrow unsigned word)$
	WR[wreg3](w1) $\leftarrow$ floor WR[wreg2](w1) (single $\rightarrow$ unsigned word)
	$WR[wreg3](w0) \leftarrow floor WR[wreg2](w0) (single \rightarrow unsigned word)$
[Format]	Format M: 2OP
[Opcode]	

54 0 31 15 11 10 27 26 25 23 22 17 16 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 rrrr W www W wreg2 sub-op wreg3 category sub-op

[Descriptions] Arithmetically converts the single-precision floating-point format contents in the elements of vector register wreg2 to an unsigned 32-bit integer format and stores the results in the respective elements of vector register wreg3.

The results are rounded toward  $-\infty$  regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or a negative number, or when the rounded result is outside the range of  $2^{32} - 1$  to 0, an IEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FXSR register is set as an invalid operation and no exception occurs. The return value differs according to the value of the source operand as follows:

- Source is a positive number outside the range of  $2^{32} 1$  to 0, or  $+\infty$ :  $2^{32} 1$  is returned.
- Source is a negative number, not-a-number, or  $-\infty$ : 0 is returned.

[Floating-point operationUnimplemented operation exception (E)exceptions]Invalid operation exception (V)Inexact exception (I)



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



#### 2.5.4.21 FLOORF.SW4

<extended floating-point="" inst<="" th=""><th>tructions&gt;</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></extended>	tructions>												
			Floati	ing-point SI	MD Convert Sing	e to	Word, roun	d toward	negative	(single)			
FLOORF.S	vv4												
				Extended	floating-point typ	e coi	nversion (S	ingle pre	cision $\rightarrow$	Integer)			
[Instruction format]	FLOORF.SW	4 wreg2, wreg3											
[Operation]	WR[wreg3](w	v3) ← floor WR[v	vreg2	](w3) (sii	ngle $\rightarrow$ word	)							
	$WR[wreg3](w2) \leftarrow floor WR[wreg2](w2) (single \rightarrow word)$												
	WR[wreg3](w	$(1) \leftarrow \text{floor WR}[w]$	vreg2	](w1) (sin	$ngle \rightarrow word$	)							
	$WR[wreg3](w0) \leftarrow floor WR[wreg2](w0) (single \rightarrow word)$												
[Format]	Format M: 2C	P											
[Opcode]													
	15 11	10 5	4	0	31 2	7 26	25 23	22		17 16			
	rrrr	1 1 1 1 1 1	0 0	1 1 0	wwww	1	0 1 1	0 1	0 0 0	0 0			
	wreg2		s	sub-op	wreg3		category		sub-op				
[Descriptions]		on arithmetically of the ter wreg2 to 32-b											
	The results are	e rounded toward	-∞ re	gardless	of the current	rou	inding m	ode.					
		the operand is infinite $a^{31} - 1$ to $-2^{31}$ , and											
	If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FXSR register is set as an invalid operation and no exception occurs. The return value differs according to the value of the source operand as follows:												
	• Source is a positive number or $+\infty$ : $2^{31} - 1$ is returned.												
	• Source is a	negative number	, not-	a-numbei	r, or $-\infty: -2^{32}$	is r	eturned.						
[Floating-point operation	Unimplemente	ed operation exce	ption	(E)									
exceptions]	Invalid operat	ion exception (V)											
	Inexact except	tion (I)											



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

# [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



#### FLPV.S4 2.5.4.22

<Extended Floating-point Instructions>

# **FLPV.S4**

Floating-point SIMD Flip (single)

Extended floating-point data flip (Single precision)

[Instruction format]	FLPV.S4 imm2, wreg2, wreg3
[Operation]	if (imm2 == 0) then WR[wreg3](w3) $\leftarrow$ WR[wreg2](w3) WR[wreg3](w2) $\leftarrow$ neg(WR[wreg2](w2)) WR[wreg3](w1) $\leftarrow$ WR[wreg2](w1) WR[wreg3](w0) $\leftarrow$ neg(WR[wreg2](w0))
	else if (imm2 == 1) then WR[wreg3](w3) ← WR[wreg2](w2) WR[wreg3](w2) ← neg(WR[wreg2](w3)) WR[wreg3](w1) ← WR[wreg2](w0) WR[wreg3](w0) ← neg(WR[wreg2](w1))
	else if (imm2 == 2) then WR[wreg3](w3) ← neg(WR[wreg2](w3)) WR[wreg3](w2) ← WR[wreg2](w2) WR[wreg3](w1) ← neg(WR[wreg2](w1)) WR[wreg3](w0) ← WR[wreg2](w0)
	else $WR[wreg3](w3) \leftarrow neg(WR[wreg2](w2))$ $WR[wreg3](w2) \leftarrow WR[wreg2](w3)$ $WR[wreg3](w1) \leftarrow neg(WR[wreg2](w0))$ $WR[wreg3](w0) \leftarrow WR[wreg2](w1)$
[Format]	Format M: 20P

[Format]

[Opcode]

	15				11	10					5	4				0	31				27	26	25		23	22					17	16
	r	r	r	r	r	1	1	1	1	1	1	1	1	0	i	i	w	w	W	w	W	1	0	1	1	0	1	0	0	0	0	0
Ī		v	vreg	2								sub-op				wreg3					са	iteg	ory			sub	o-op					

[Descriptions] This instruction exchanges between the even- and odd-number elements of vector register wreg2 according to the 2-bit immediate value and flips the sign of the even- or odd-number elements according to the 2-bit immediate value.

[Floating-point operation None exceptions]



#### 2.5.4.23 FMAF.S4

<extended floating-point="" inst<="" th=""><th></th></extended>												
FMAF.S4	Floating-point SIMD Fused-Multiply-Add (Single)											
	Extended floating-point fused-multiply-add operation (Single precision)											
[Instruction format]	FMAF.S4 wreg1, wreg2, wreg3											
[Operation]	$WR[wreg3](w3) \leftarrow fma(WR[wreg2](w3), WR[wreg1](w3), WR[wreg3](w3))$											
	$WR[wreg3](w2) \leftarrow fma(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2))$											
	$WR[wreg3](w1) \leftarrow fma(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1))$											
	$WR[wreg3](w0) \leftarrow fma(WR[wreg2](w0), WR[wreg1](w0), WR[wreg3](w0))$											
[Format]	Format M: 3OP											
[Opcode]												
	15 11 10 5 4 0 31 27 26 25 23 22 17 16											
	r r r r r 1 1 1 1 1 1 R R R R R W W W W W 1 0 0 1 1 0 0 0 0 0 0											
	wreg2 wreg1 wreg3 category sub-op											
[Descriptions]	This instruction multiplies the single-precision floating-point format contents in each element of vector register wreg2 by the single-precision floating-point format contents in											
	vector register wreg1, adds the results to the single-precision floating-point format contents in											
	in vector register wreg3, and stores the results in the corresponding element of vector											
	register wreg3. The results of the multiply operation is not rounded, but the results of the add operation is rounded in accordance with the current rounding mode.											
	and operation is founded in accordance with the current founding mode.											
[Floating-point operation	Unimplemented operation exception (E)											
exceptions]	Invalid operation exception (V)											
	Inexact exception (I)											
	Overflow exception (O)											
	Underflow exception (U)											

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.


	wreg2(B)								
wreg3(C)	wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal			•		+∞	-∞		
	-Normal		fma (E	3, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11		+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
±0	+Normal					+∞	-∞		
	-Normal		fma (E	3, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞	0.11		+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
+∞	+Normal					+∞	Q-NaN[V]		
	-Normal		+	.00		Q-NaN[V]	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	Q-NaN[V]			+∞	Q-NaN[V]		
	-∞	Q-NaN[V]	+∞	Q-INA	aN[V]	Q-NaN[V]	+∞		
-∞	+Normal					Q-NaN[V]	-∞		
	-Normal		-	-00		-∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	-∞	Q-Na		Q-NaN[V]	-∞		
	-∞	-∞	Q-NaN[V]	Q-INA	an [v]	-∞	Q-NaN[V]		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								

Note 1. [ ] indicates an exception that must occur.

Note 2. When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

[Supplement] The results of the fused-multiply-add operation are rounded in accordance with the current rounding mode. Consequently, the results of operation differ from the results of executing a combination of the ADDF.S4 and MULF.S4 instructions.



#### 2.5.4.24 FMSF.S4

				Floating-pc	oint S	MD Fused	Multiply-Subtrac	ct (Sin	gle)						
FMSF.S4			Extended floatin	ng-point fused-mu	ıltiply	-subtract op	peration (Single	precis	ion)						
[Instruction format]	FMSF.S4 wreg1	, wreg2, wreg3													
[Operation]	WR[wreg3](w3)	$\phi \leftarrow \text{fms}(\text{WR}[\text{wr}])$	reg2](w3), WI	R[wreg1](w3)	), W	R[wreg3	5](w3))								
	WR[wreg3](w2)	$\leftarrow$ fms(WR[wi	reg2](w2), WI	R[wreg1](w2)	), W	R[wreg3	5](w2))								
	WR[wreg3](w1)	$\leftarrow$ fms(WR[wi	reg2](w1), WI	R[wreg1](w1]	), W	R[wreg3	5](w1))								
	WR[wreg3](w0)	$\leftarrow$ fms(WR[w	reg2](w0), WI	R[wreg1](w0)	), W	R[wreg3	5](w0))								
[Format]	Format M: 3OP														
[Opcode]															
	15 11 10				7 26		22		/ 16						
	r r r r r 1		RRRRR	w w w w v	v T	001	1 0 0 0	0 1	0						
	wreg2		wreg1	wreg3		category	sub-op								
[Descriptions]	This instruction element of vector vector register w register wreg3, a wreg3. The resu operation is rour	or register wreg2 yreg1, subtracts and stores the re lts of the multip	2 by the single the single-precessults in the con- sults operation is	-precision flo cision floatin rresponding e s not rounded	oatin g-po elem , bu	g-point f bint form ents of v t the resu	format conter at contents in ector registe	nts in n veo r	ctor						
[Floating-point operation	Unimplemented	operation excep	ption (E)												
exceptions]	Invalid operation														
	Invalid operation exception (V) Inexact exception (I)														
	Inexact exceptio	1 . /													
	Inexact exception Overflow except	n (I)													

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



	wreg2(B)								
wreg3(C)	wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					+∞	-∞		
	-Normal		fms (E	3, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞		10.0	+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
±0	+Normal					+∞	-∞		
	-Normal		fms (E	3, A, C)		-∞	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	-∞		10.0	+∞	-∞		
	-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
+∞	+Normal					Q-NaN[V]	-∞		
	-Normal		-	·∞		-∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	-∞	0.11	N ID /1	Q-NaN[V]	-∞		
	-∞	-∞	Q-NaN[V]	Q-Na	aN[V]	-∞	Q-NaN[V]		
-∞	+Normal					+∞	Q-NaN[V]		
	-Normal		+	.00		Q-NaN[V]	+∞		
	±0					Q-Na	aN[V]		
	+∞	+∞	Q-NaN[V]			+∞	Q-NaN[V]		
	-∞	Q-NaN[V]	+∞	Q-INA	aN[V]	Q-NaN[V]	+∞		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								₩[v]

Note 1. [] indicates an exception that must occur.

Note 2. When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

[Supplement] The results of the fused-multiply-subtract operation are rounded in accordance with the current rounding mode. Consequently, the results of operation differ from the results of executing a combination of the SUBF.S4 and MULF.S4 instructions.



#### 2.5.4.25 FNMAF.S4

<extended floating-point="" inst<="" th=""><th>ructions&gt;</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></extended>	ructions>																		
FNMAF.S4								Flo	ating	J-point	SIN	1D F	us	ed-Ne	gat	ve-N	lultip	y-Add	Single
						Exte	ended f	loati	ng-po	oint fue	sed-	mult	tipl	y-add	оре	eratio	n (Si	ngle pr	ecision
									0.					<u> </u>				0.	
[Instruction format]	FNMAF.S4	ł w	reg1, wreg	g2, wre	eg3	;													
[Operation]	WR[wreg3]	](w	/3) ← neg	(fma(V	VR	[wreg2	2](w3	), V	VR[	wreg	[1]	w3	),	WR	[WI	eg3	](w	3)))	
-	$WR[wreg3](w2) \leftarrow neg(fma(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2)))$																		
	$WR[wreg3](w1) \leftarrow neg(fma(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1)))$																		
	$WR[wreg3](w0) \leftarrow neg(fma(WR[wreg2](w0), WR[wreg1](w0), WR[wreg3](w0)))$																		
[Format]	Format M:	30	P																
[Opcode]																			
	15	11	10	5	4		0	31			27	26	25	5	23	22			17 1
	rrrr	r	1 1 1 1	1 1 1	R	RR	R R	W	W	w w	W	1	0	0	1	1 (	) ()	0 1	0 0
	wreg2					wreg	1		W	reg3			С	ategoi	у		sul	о-ор	
[Descriptions]	This instruc	etio	on multipli	es the	sin	gle-pr	ecisio	n fl	oati	ng-p	oin	t fo	orr	nat c	on	tent	s in	each	
[[]	element of		-							• •									s in
	vector regis		-					-	-					-	-				ntents
	in vector re register wre	-	-								-			-					led,
	but the resu	-		-										-					
	mode.																		
[Floating-point operation	Unimpleme	ente	ed operation	on exce	ept	ion (E)	)												
exceptions]	Unimplemented operation exception (E) Invalid operation exception (V)																		
	Invalid ope	rati	•		)														
	Invalid ope		ion except		)														
	-	ept	ion except tion (I)	tion (V	)														

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



	wreg2(B)								
wreg3(C)	wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal			L		-∞	+∞		
	-Normal		–fma (I	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-∞	Q-Na	an(v)	+∞	-∞		
±0	+Normal					-∞	+∞		
	-Normal		–fma (I	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-8	Q-Na	an[v]	+∞	-∞		
+∞	+Normal					-∞	Q-NaN[V]		
	-Normal		-	∞0		Q-NaN[V]	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	Q-NaN[V]	Q-Na		-∞	Q-NaN[V]		
	-∞	Q-NaN[V]	-∞		מאנען	Q-NaN[V]	-∞		
-∞	+Normal					Q-NaN[V]	+∞		
	-Normal		+	00		+∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	+∞	Q-Na		Q-NaN[V]	+∞		
	-∞	+∞	Q-NaN[V]	Q-No	מאנע	+∞	Q-NaN[V]		
Q-NaN	±Normal								
	±0			Q-1	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								

Note 1. [] indicates an exception that must occur.

Note 2. When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

[Supplement] The results of the fused-multiply-add operation are rounded in accordance with the current rounding mode. Consequently, the results of operation differ from the results of executing a combination of the ADDF.S4, MULF.S4, and NEGF.S4 instructions.



# 2.5.4.26 FNMSF.S4

<extended floating-point="" inst<="" th=""><th>ructions&gt;</th><th></th><th></th><th></th><th></th></extended>	ructions>				
		Flo	ating-point SIMD Fu	sed-Negative-Multiply-Subtract (Sin	gle)
FNMSF.S4					
		Extended floati	ng-point fused-multip	bly-subtract operation (Single precis	ion)
[Instruction format]	FNMSF.S4 wreg1, wreg2, w	vreg3			
[Operation]	$WR[wreg3](w3) \leftarrow neg(fms(V))$	WR[wreg2](w3)	), WR[wreg1](w	v3), WR[wreg3](w3)))	
	$WR[wreg3](w2) \leftarrow neg(fms(V))$	WR[wreg2](w2]	), WR[wreg1](w	v2), WR[wreg3](w2)))	
	$WR[wreg3](w1) \leftarrow neg(fms(V))$	WR[wreg2](w1]	), WR[wreg1](w	v1), WR[wreg3](w1)))	
	$WR[wreg3](w0) \leftarrow neg(fms(V))$	WR[wreg2](w0)	), WR[wreg1](w	v0), WR[wreg3](w0)))	
[Format]	Format M: 3OP				
[Opcode]					
	15 11 10 5	5 4 0	31 27 2	26 25 23 22 17	7 16
	rrrr 1 1 1 1 1 1	RRRRR	wwww	1 0 0 1 1 0 0 0 1 1	0
	wreg2	wreg1	wreg3	category sub-op	
[Descriptions]	This instruction multiplies the	• •	• •		
	element of vector register wre vector register wreg1, subtract		-	• •	
	register wreg3, and stores the			-	
	with its sign inverted. The resu				s of
	the subtract operation is round	ed in accordance	ce with the curre	ent rounding mode.	
Electing point operation	Unimplemented operation ave	option (E)			
[Floating-point operation exceptions]	Unimplemented operation exc	•			
	Invalid operation exception (V	)			
	Inexact exception (I)				
	Overflow exception (O)				
	Underflow exception (U)				

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



	wreg2(B)								
wreg3(C)	wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
±Normal	+Normal					-∞	+∞		
	-Normal		–fms (E	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-∞	Q-Na	aN[V]	+∞	-∞		
±0	+Normal					-∞	+∞		
	-Normal		–fms (E	B, A, C)		+∞	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	+∞	0.11		-∞	+∞		
	-∞	+∞	-∞	Q-Na	aN[V]	+∞	-∞		
+∞	+Normal					Q-NaN[V]	+∞		
	-Normal		+	.00		+∞	Q-NaN[V]		
	±0					Q-Na	aN[V]		
	+∞	Q-NaN[V]	+∞	0.14	aN[V]	Q-NaN[V]	+∞		
	-∞	+∞	Q-NaN[V]	Q-INA	an [v]	+∞	Q-NaN[V]		
-∞	+Normal					-∞	Q-NaN[V]		
	-Normal		-	-00		Q-NaN[V]	-∞		
	±0					Q-Na	aN[V]		
	+∞	-∞	Q-NaN[V]			-∞	Q-NaN[V]		
	-∞	Q-NaN[V]	-∞		aN[V]	Q-NaN[V]	-∞		
Q-NaN	±Normal								
	±0			Q-N	laN				
	±∞								
Not S-NaN	Q-NaN							Q-NaN	
Don't care	S-NaN								Q-NaN[V]
S-NaN	Don't care								A

Note 1. [ ] indicates an exception that must occur.

Note 2. When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

[Supplement] The results of the fused-multiply-subtract operation are rounded in accordance with the current rounding mode. Consequently, the results of operation differ from the results of executing a combination of the SUBF.S4, MULF.S4 and NEGF.S4 instructions.



# 2.5.4.27 LDV.DW

<Extended Floating-point Instructions>

	Load Vector (Double-Word)
LDV.DW	Load double-word data to vector register
[Instruction format]	LDV.DW imm2, disp16[reg1], wreg3
[Operation]	adr $\leftarrow$ GR[reg1] + sign-extend (disp16) <sup>Note 1</sup>
	CheckException(MAE)
	CheckException(MDP)
	val ← Load-memory(adr, Double-word)
	$WR[wreg3](dw1) \leftarrow (imm2[1] == 1) ? val: WR[wreg3](dw1)$
	$WR[wreg3](dw0) \leftarrow (imm2[0] == 1) ? val: WR[wreg3](dw0)$
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	
	15 11 10 5 4 0 31 27 26 17 16
	0 0 0 0 0 1 1 1 1 0 1 R R R R R w w w w w 0 1 1 0 i i 1 1 1 0 1
	sub-op reg1 wreg3 sub-op
	47 35 34 33 32
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	disp16

Where ii = 2-bit immediate data ddddddddddd = Higher-order 13 bits of disp16



[Descriptions] This instruction adds together the word data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length to generate a 32-bit address. The instruction reads double-word data from the generated 32-bit address. Each element of vector register wreg3 is loaded with the read data when the corresponding bit in the 2-bit immediate data is set to 1.

#### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Since an arbitrary value can be set as the 2-bit immediate data, data can be stored in four ways. The read doubleword data can be stored in both elements of the vector register at the same time by setting both bits of the 2-bit immediate data to 1. If both bits of the 2-bit immediate data are set to 0, the read double-word data is stored in none of the elements of the vector register.
- 3. Even when a nonzero value is set in bits 32 to 34, the opcode functions as the LDV.DW instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the three lower-order bits were set to 0.



# 2.5.4.28 LDV.QW

<Extended Floating-point Instructions>

LDV.QW	Load Vector (Quad-Word)
	Load quad-word data to vector register
[Instruction format]	LDV.QW disp16[reg1], wreg3
[Operation]	adr ← GR[reg1] + sign-extend (disp16) <sup>Note 1</sup> CheckException(MAE) CheckException(MDP) WR[wreg3] ← Load-memory(adr, Quad-word)
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	
	<u>15 11 10 5 4 0 31 27 26 17 16</u>
	0 0 0 0 0 1 1 1 1 0 1 R R R R R W W W W 0 1 0 1 0 1 1 1 0 1
	sub-op reg1 wreg3 sub-op
	47 35 34 33 32
	a a a a a a a a a a a o o o o
	disp16
	dddddddddd = Higher-order 12 bits of disp16
[Descriptions]	This instruction adds together the word data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length to generate a 32-bit address. The instruction reads quad-word data from the generated 32-bit address and stores it in vector register wreg3.

# CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Even when a nonzero value is set in bits 32 to 35, the opcode functions as the LDV.QW instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the four lower-order bits were set to 0.



# 2.5.4.29 LDV.W

<Extended Floating-point Instructions>

LDV.W	Load Vector (Word)
	Load word data to vector register
[Instruction format]	LDV.W imm4, disp16[reg1], wreg3
[Operation]	adr ← GR[reg1] + sign-extend (disp16) <sup>Note 1</sup> CheckException(MAE) CheckException(MDP) val ← Load-memory(adr, Word) WR[wreg3](w3) (imm4[3] == 1) ? val: WR[wreg3](w3) WR[wreg3](w2) (imm4[2] == 1) ? val: WR[wreg3](w2) WR[wreg3](w1) (imm4[1] == 1) ? val: WR[wreg3](w1) WR[wreg3](w0) (imm4[0] == 1) ? val: WR[wreg3](w0) Note 1. An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	
	15     11     10     5     4     0     31     27     26     17     16       0     0     0     0     1     1     1     0     1     R     R     R     R     W     W     W     0     0     0     1     1     1     0     1

15				11	10					5	4				0	31				27	26									17	16
0	0	0	0	0	1	1	1	1	0	1	R	R	R	R	R	w	w	W	w	W	0	0	i	i	i	i	1	1	1	0	1
	S	ub-a	р										reg1				v	vreg	13						sub	o-op	1				
47												35	34	33	32																
d	d	d	d	d	d	d	d	d	d	d	d	d	d	0	0																
							dis	p16																							

Where iiii = 4-bit immediate data dddddddddddd = Higher-order 14 bits of disp16



# [Descriptions] This instruction adds together the word data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length to generate a 32-bit address. The instruction reads word data from the generated 32-bit address and stores it in vector register wreg3. Each element of vector register wreg3 is loaded with the read data when the corresponding bit in the 4-bit immediate data is set to 1.

#### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Since an arbitrary value can be set as the 4-bit immediate data, data can be stored in 16 ways. The read word data can be stored in all elements of the vector register at the same time by setting all bits of the 4bit immediate data to 1. If all bits of the 4-bit immediate data are set to 0, the read word data is stored in none of the elements of the vector register.
- 3. Even when a nonzero value is set in bits 32 and 33, the opcode functions as the LDV.W instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the two lower-order bits were set to 0.



# 2.5.4.30 LDVZ.H4

<Extended Floating-point Instructions>

<extended floating-point<="" th=""><th></th></extended>	
	Load Vector at Even Halfword field
LDVZ.H4	
	Interleaved load of halfword to vector register
[Instruction format]	LDVZ.H4 disp16[reg1], wreg3
[Operation]	adr $\leftarrow$ GR[reg1] + sign-extend (disp16) <sup>Note 1</sup>
	CheckException(MAE)
	CheckException(MDP)
	$val[63:0] \leftarrow Load-memory(adr, Double-word)$
	$WR[wreg3](h7) \leftarrow 0$
	$WR[wreg3](h6) \leftarrow val[63:48]$
	$WR[wreg3](h5) \leftarrow 0$
	$WR[wreg3](h4) \leftarrow val[47:32]$
	$WR[wreg3](h3) \leftarrow 0$
	$WR[wreg3](h2) \leftarrow val[31:16]$
	$WR[wreg3](h1) \leftarrow 0$
	$WR[wreg3](h0) \leftarrow val[15:0]$
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D

# [Opcode]

15				11	10					5	4				0	31				27	26									17	16
0	0	0	0	0	1	1	1	1	0	1	R	R	R	R	R	w	w	w	W	W	0	1	1	1	1	0	1	1	1	0	1
	S	ub-o	р										reg1	l			v	vreg	3						sub	о-ор					
47												35	34	33	32																
d	d	d	d	d	d	d	d	d	d	d	d	d	0	0	0																
							dis	p16																							

Where dddddddddd = Higher-order 13 bits of disp16



[Descriptions]This instruction reads double-word data from the 32-bit address which is generated by<br/>adding together the data in general-purpose register reg1 and the 16-bit displacement data<br/>that is sign-extended to word length. The read double-word data is zero-extended in 16-bit<br/>units and stored in the respective elements of vector register wreg3.

#### CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Even when a nonzero value is set in bits 32 to 34, the opcode functions as the LDVZ.H4 instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the three lower-order bits were set to 0.



# 2.5.4.31 MAXF.S4

<Extended Floating-point Instructions>

MAXF.S4	Floating-point SIMD Maximum (single)
	Extended floating-point maximum value (Single precision)
[Instruction format]	MAXF.S4 wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow max (WR[wreg2](w3), WR[wreg1](w3))$
	$WR[wreg3](w2) \leftarrow max (WR[wreg2](w2), WR[wreg1](w2))$
	$WR[wreg3](w1) \leftarrow max (WR[wreg2](w1), WR[wreg1](w1))$
	$WR[wreg3](w0) \leftarrow max (WR[wreg2](w0), WR[wreg1](w0))$
[Format]	Format M: 3OP
[Opcode]	
	<u>15 11 10 5 4 0 31 27 26 25 23 22 17 16</u>
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1
	wreg2 wreg1 wreg3 category sub-op
[Descriptions]	This instruction stores the maximum values of the single-precision floating-point format contents in each element of vector registers wreg1 and wreg2 in the corresponding elements of vector register wreg3. If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are not enabled, Q-NaN is stored and no exception is generated.
[Floating-point operation	Invalid operation exception (V)

exceptions]

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-8	Q-NaN	S-NaN
+Normal								
-Normal								
+0								
-0			max	(B, A)			wreg1(A)	
+∞								
-∞								
Q-NaN			wreg	j2(B)			Q-NaN	
S-NaN								Q-NaN[V]

Note: [ ] indicates an exception that must occur.

[Supplement]

If both of vector registers wreg1 and wreg2 are set to either +0 or -0, whether wreg3 is to be loaded with either +0 or -0 is not defined.

The subnormal number input is not flushed even if FXSR.FS = 1.



# 2.5.4.32 MAXRF.S4

**MAXRF.S4** 

<Extended Floating-point Instructions>

Floating-point SIMD Maximum Reduction (single)

Extended floating-point maximum value reduction (Single precision)

[Instruction format]	MAXRF.S4 w	vreg1, wreg2, wre	g3							
[Operation]	WR[wreg3](w WR[wreg3](w	73) ← max (WR[v 72) ← max (WR[v 71) ← max (WR[v 70) ← max (WR[v	wreg1](w3), W wreg2](w1), W	/R[wreg1](w2 /R[wreg2](w0	)) ))					
[Format]	Format M: 30	P								
[Opcode]										
					26		3 22		17	
	rrrr	1 1 1 1 1 1	RRRRR	wwww	1	0 1 1	0 :	1 1 1 0	1	0
	wreg2		wreg1	wreg3		category		sub-op		
[Descriptions]	contents in the number eleme This instructio	on stores the maximum e odd- and even-ments of vector registion on stores the maximum odd, and even m	umber element ster wreg3. mum values of	ts of vector reg	gist ecis	ers wreg sion floa	g1 in ting-	the even-	nat	oor
		e odd- and even-meter wreg		is of vector reg	gist	ers wreg	g2 111	the odd-n	umo	ber
		ource operands is valid operation ex				-		-		
[Floating-point operation exceptions]	Invalid operat	ion exception (V)								

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



B*2 A*2	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0				(4 D)			٨	
-0			max	(А, В)			A	
+∞								
-∞								
Q-NaN			E	3			Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** Refer to [Operation] for the operands A and B that are input to produce the output.

[Supplement]

If both of vector registers wreg1 and wreg2 are set to either +0 or -0, whether wreg3 is to be loaded with either +0 or -0 is not defined.

The subnormal number input is not flushed even if FXSR.FS = 1.



# 2.5.4.33 MINF.S4

<Extended Floating-point Instructions>

<extended floating-point="" inst<="" th=""><th>ructions&gt; Floating-point SIMD Minimum (single)</th></extended>	ructions> Floating-point SIMD Minimum (single)
MINF.S4	
	Extended floating-point minimum value (Single precision)
[Instruction format]	MINF.S4 wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow min (WR[wreg2](w3), WR[wreg1](w3))$
	$WR[wreg3](w2) \leftarrow min (WR[wreg2](w2), WR[wreg1](w2))$
	$WR[wreg3](w1) \leftarrow min (WR[wreg2](w1), WR[wreg1](w1))$
	$WR[wreg3](w0) \leftarrow min (WR[wreg2](w0), WR[wreg1](w0))$
[Format]	Format M: 3OP
[Opcode]	
	15         11         10         5         4         0         31         27         26         25         23         22         17         16
	rrrr11111RRRRR         wwwww10110110101100
	wreg2 wreg1 wreg3 category sub-op
[Descriptions]	This instruction stores the minimum values of the single-precision floating-point format contents in each element of vector registers wreg1 and wreg2 in the corresponding elements of vector register wreg3.
	If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are not enabled, Q-NaN is stored and no exception is generated.
[Floating-point operation exceptions]	Invalid operation exception (V)

## NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-8	Q-NaN	S-NaN
+Normal								
-Normal								
+0								
-0			min (	B, A)			wreg1(A)	
+∞								
-∞								
Q-NaN			wreg	j2(B)			Q-NaN	
S-NaN								Q-NaN[V]

Note: [ ] indicates an exception that must occur.

[Supplement]

If both of vector registers wreg1 and wreg2 are set to either +0 or -0, whether wreg3 is to be loaded with either +0 or -0 is not defined.

The subnormal number input is not flushed even if FXSR.FS = 1.



# 2.5.4.34 MINRF.S4

<extended floating-point="" inst<="" th=""><th>tructions&gt; Floating-point SIMD Minimum Reduction (single)</th></extended>	tructions> Floating-point SIMD Minimum Reduction (single)
MINRF.S4	
	Extended floating-point minimum value reduction (Single precision)
[Instruction format]	MINRF.S4 wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow min (WR[wreg2](w3), WR[wreg2](w2))$
	$WR[wreg3](w2) \leftarrow min (WR[wreg1](w3), WR[wreg1](w2))$
	$WR[wreg3](w1) \leftarrow min (WR[wreg2](w1), WR[wreg2](w0))$
	$WR[wreg3](w0) \leftarrow min (WR[wreg1](w1), WR[wreg1](w0))$
[Format]	Format M: 3OP
[Opcode]	
	15         11         10         5         4         0         31         27         26         25         23         22         17         16
	r r r r r 1 1 1 1 1 1 R R R R R w w w w 1 0 1 1 0 1 1 1 0 0
	wreg2 wreg1 wreg3 category sub-op
[Descriptions]	This instruction stores the minimum value of the single-precision floating-point format contents in the odd- and even-number elements of vector registers wreg1 in the even-number elements of vector register wreg3.
	This instruction stores the minimum value of the single-precision floating-point format contents in the odd- and even-number elements of vector registers wreg2 in the odd-number elements of vector register wreg3.
	If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are not enabled, Q-NaN is stored and no exception is generated.
[Floating-point operation exceptions]	Invalid operation exception (V)

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



B*2 A*2	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0								
-0			min (	А, В)			A	
+∞								
-∞								
Q-NaN			E	3			Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** Refer to [Operation] for the operands A and B that are input to produce the output.

[Supplement]

If both of vector registers wreg1 and wreg2 are set to either +0 or -0, whether wreg3 is to be loaded with either +0 or -0 is not defined.

The subnormal number input is not flushed even if FXSR.FS = 1.



# 2.5.4.35 MOVV.W4

<Extended Floating-point Instructions>

Move vector register to vector register

# MOVV.W4

Move vector register

[Instruction format]	MOVV.W4 wreg2, wreg3
[Operation]	$WR[wreg3] \leftarrow WR[wreg2]$
[Format]	Format M: 2OP
[Opcode]	

_	15				11	10					5	4				0	31				27	26	25		23	22					17	16
	r	r	r	r	r	1	1	1	1	1	1	1	1	1	1	0	w	w	W	w	w	1	0	1	1	0	1	0	0	0	0	0
		١	wreg	j2									s	ub-c	р			v	vreg	3			са	iteg	ory			sub	-op			

[Descriptions] This instruction stores the contents in each element of vector register wreg2 in the corresponding element of vector register wreg3.



# 2.5.4.36 MULF.S4

<Extended Floating-point Instructions>

	Floating-point SIMD Multiply (single)
MULF.S4	Extended floating-point multiplication (Single precision
[Instruction format]	MULF.S4 wreg1, wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) \times WR[wreg1](w3)$
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) \times WR[wreg1](w2)$
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) \times WR[wreg1](w1)$
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) \times WR[wreg1](w0)$
[Format]	Format M: 3OP
[Opcode]	
	15     11     10     5     4     0     31     27     26     25     23     22     17     16
	r r r r r 1 1 1 1 1 1 R R R R R W W W W W 1 0 1 1 0 1 0 1 0 0 0
	wreg2 wreg1 wreg3 category sub-op
[Descriptions]	This instruction multiplies the single-precision floating-point format contents in each element of vector wreg2 by the single-precision floating-point format contents in vector register wreg1 and stores the results in the corresponding element of vector register wreg3. 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	Unimplemented operation exception (E)
exceptions]	Invalid operation exception (V)
	Inexact exception (I)
	Overflow exception (O)

#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-8		
-Normal		D.			-∞	+∞		
+0		B×A				Q-NaN[V]		
-0						antai		
+∞	+∞	- 8			+∞	- 8		
-∞	-∞	+∞	Q-Na	an[v]	- 8	+∞		
Q-NaN						Q-NaN		
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



#### 2.5.4.37 MULRF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Multiply Reduction (single)

**MULRF.S4** 

Extended floating-point multiplication reduction (Single precision)

[Instruction format]	MULRF.S4 wreg1, wreg2, wreg3							
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) \times WR[wreg2](w2)$ $WR[wreg3](w2) \leftarrow WR[wreg1](w3) \times WR[wreg1](w2)$ $WR[wreg3](w1) \leftarrow WR[wreg2](w1) \times WR[wreg2](w0)$ $WR[wreg3](w0) \leftarrow WR[wreg1](w1) \times WR[wreg1](w0)$							
[Format]	Format M: 3OP							
[Opcode]								
	15         11         10         5         4         0         31         27         26         25         23         22         17         16							
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 0 1 1 1 0 0 0							
	wreg2 wreg1 wreg3 category sub-op							
[Descriptions]	The instruction multiples the single-precision floating-point contents in the even- and odd- number elements of vector register wreg1 and stores the results in the even-number elements of vector register wreg3.							
	The instruction multiples the single-precision floating-point contents in the even- and odd- number elements of vector register wreg2 and stores the results in the odd-number elements of vector register wreg3.							
	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	Unimplemented operation exception (E)							
exceptions]	Invalid operation exception (V)							
	Inexact exception (I)							
	Overflow exception (O)							
	Underflow exception (U)							



#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

B*4 A*4	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-∞		
-Normal			5		-∞	+∞		
+0	A×B				Q-NaN[V]			
-0					Q-Na	an[v]		
+∞	+∞	-∞	0.14	- N IG) (1	+∞	-∞		
-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

**Note 3.** The results of operations performed on the elements will not affect one another.

Note 4. Refer to [Operation] for the operands A and B that are input to produce the output.



# 2.5.4.38 MULXF.S4

<Extended Floating-point Instructions>

Μ	JLXF.S	54

Floating-point SIMD Multiply Exchange (single)

Extended floating-point multiply exchange (Single precision)

[Instruction format]	MULXF.S4 wreg1, wreg2, wreg3						
[Operation]	$VR[wreg3](w3) \leftarrow WR[wreg2](w3) \times WR[wreg1](w2)$						
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) \times WR[wreg1](w3)$						
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) \times WR[wreg1](w0)$						
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) \times WR[wreg1](w1)$						
[Format]	Format M: 3OP						
[Opcode]							
	15         11         10         5         4         0         31         27         26         25         23         22         17         16						
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 1 0 0 1 0 0 0						
	wreg2 wreg1 wreg3 category sub-op						
[Descriptions]	This instruction multiples the single-precision floating-point contents in the even-number elements of vector register wreg1 by the single-precision floating-point format contents in the odd-number elements of vector register wreg2 and stores the results in the odd-number elements of vector register wreg3.						
	This instruction multiples the single-precision floating-point contents in the odd-number elements of vector register wreg1 by the single-precision floating-point format contents in the even-number elements of vector register wreg2 and stores the results in the even-number elements of vector register wreg3.						
	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	Unimplemented operation exception (E)						
exceptions]	Invalid operation exception (V)						
	Inexact exception (I)						
	Overflow exception (O)						
	Underflow exception (U)						



#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal					+∞	-8		
-Normal		D.			-∞	+∞		
+0		B×A						
-0					Q-NaN[V]			
+∞	+∞	-∞			+∞	- 8		
-∞	-∞	+∞	Q-Na	aN[V]	-∞	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



#### 2.5.4.39 NEGF.S4

<extended floating-point="" inst<="" th=""><th></th></extended>	
NEGF.S4	Floating-point SIMD Negative (single)
NL01.04	Extended floating-point sign inversion (Single precision)
[Instruction format]	NEGF.S4 wreg2, wreg3
[Operation]	$WR[wreg3](w3) \leftarrow neg(WR[wreg2](w3))$
	$WR[wreg3](w2) \leftarrow neg(WR[wreg2](w2))$
	$WR[wreg3](w1) \leftarrow neg(WR[wreg2](w1))$
	$WR[wreg3](w0) \leftarrow neg(WR[wreg2](w0))$
[Format]	Format M: 2OP
[Opcode]	
	15         11         10         5         4         0         31         27         26         25         23         22         17         16
	r r r r r 1 1 1 1 1 1 1 1 w w w w w 1 0 1 1 0 1 0
	wreg2 sub-op wreg3 category sub-op
[Descriptions]	This instruction inverts the sign of the single-precision floating-point format contents in each element of vector register wreg2, and stores the results in the corresponding element of vector register wreg3.
[Floating-point operation exceptions]	None
[Supplement]	A subnormal input will not be flushed even if the FS bit of the FXSR register is 1.



# 2.5.4.40 RECIPF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Reciprocal (single)

RECIPF.S4

Extended floating-point reciprocal (Single precision)

[Instruction format]	RECIPF.S4 wreg2, wreg3					
[Operation]	$WR[wreg3](w3) \leftarrow 1 \div WR[wreg2](w3)$ $WR[wreg3](w2) \leftarrow 1 \div WR[wreg2](w2)$ $WR[wreg3](w1) \leftarrow 1 \div WR[wreg2](w1)$ $WR[wreg3](w0) \leftarrow 1 \div WR[wreg2](w0)$					
[Format]	Format M: 2OP					
[Opcode]	15       11       10       5       4       0       31       27       26       25       23       22       17       16         r       r       r       r       r       r       1					
[Descriptions]	This instruction approximates the reciprocal of the single-precision floating-point format contents of vector register wreg2 and stores the result in vector register wreg3. The results differs from the results obtained by using the DIVF.S4 instruction.					
[Floating-point operation exceptions]	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I) Division by zero exception (Z) Underflow exception (U)					

NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	1/A[1]		+∞[Z]	-∞[Z]	+0	-0	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

#### NOTE

The results fall within the error range of 1ULP against the results of calculating 1/x. ULP: Unit in the Least-significant Place



# 2.5.4.41 ROUNDF.SUW4

<extended floating-point="" inst<="" th=""><th>tructions&gt;</th></extended>	tructions>								
	Floating-point SIMD Convert Single to Unsigned Word, round to nearest (single)								
ROUNDF.S	5UW4								
	Extended floating-point type conversion (Single precision $\rightarrow$ Unsigned integer)								
[Instruction format]	ROUNDF.SUW4 wreg2, wreg3								
[Operation]	$WR[wreg3](w3) \leftarrow round WR[wreg2](w3) (single \rightarrow unsigned word)$								
	$WR[wreg3](w2) \leftarrow round WR[wreg2](w2) (single \rightarrow unsigned word)$								
	$WR[wreg3](w1) \leftarrow round WR[wreg2](w1) (single \rightarrow unsigned word)$								
	$WR[wreg3](w0) \leftarrow round WR[wreg2](w0) (single \rightarrow unsigned word)$								
[Format]	Format M: 2OP								
[Opcode]									
	<u>15 11 10 5 4 0 31 27 26 25 23 22 17 16</u>								
	r r r r r 1 1 1 1 1 1 0 0 0 0 1 w w w w w 1 0 1 1 0 1 0								
	wreg2 sub-op wreg3 category sub-op								
[Descriptions]	Arithmetically converts the single-precision floating-point format contents in the elements								
	of vector register wreg2 to unsigned 32-bit integer format and stores the results in the								
	respective elements of vector register wreg3.								
	The result is rounded to the nearest value or an even number regardless of the current bunding mode.								
	When the source operand is infinite, not-a-number, or negative number, or when the								
	rounded result is outside the range of $2^{32}$ –1 to 0, an IEE754-defined invalid operation								
	exception is detected.								
	If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FXSR								
	<ul> <li>register is set as an invalid operation and no exception occurs. The return value differs according to the value of the source operand as follows:</li> <li>Source is a positive number outside the range of 2<sup>32</sup> - 1 to 0, or +∞: 2<sup>32</sup> - 1 is returned.</li> <li>Source is a negative number, not-a-number, or -∞: 0 is returned.</li> </ul>								
	Source is a negative number, not a number, or w. o is retained.								
[Floating-point operation	Unimplemented operation exception (E)								
exceptions]	Invalid operation exception (V)								
	Inexact exception (I)								



#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# 2.5.4.42 ROUNDF.SW4

<Extended Floating-point Instructions>

	Floating-point SIMD Convert Single to Word, round toward negative (single)				
ROUNDF.S	5VV4				
	Extended floating-point type conversion (Single precision → Integer)				
[Instruction format]	ROUNDF.SW4 wreg2, wreg3				
[Operation]	WR[wreg3](w3) $\leftarrow$ round WR[wreg2](w3) (single $\rightarrow$ word)				
	$WR[wreg3](w2) \leftarrow round WR[wreg2](w2) (single \rightarrow word)$				
	$WR[wreg3](w1) \leftarrow round WR[wreg2](w1) (single \rightarrow word)$				
	$WR[wreg3](w0) \leftarrow round WR[wreg2](w0) (single \rightarrow word)$				
[Format]	Format M: 2OP				
[Opcode]					
	15         11         10         5         4         0         31         27         26         25         23         22         17         16				
	r r r r r 1 1 1 1 1 1 0 0 0 0 0 w w w w w 1 0 1 1 0 1 0				
	wreg2 sub-op wreg3 category sub-op				
[Descriptions]	Arithmetically converts the single-precision floating-point format contents in the elements of vector register wreg2 to 32-bit integer format and stores the results in the respective elements of vector register wreg3.				
	The result is rounded to the nearest value or an even number 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 $2^{31} - 1$ to $-2^{31}$ , an IEE754-defined invalid operation exception is detected.				
	If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FXSR register is set as an invalid operation and no exception occurs. The return value differs according to the value of the source operand as follows:				
	• Source is a positive number or $+\infty$ : $2^{31} - 1$ is returned.				
	• Source is a negative number, not-a-number, or $-\infty$ : $-2^{31}$ is returned.				
[Floating-point operation	Unimplemented operation exception (E)				
exceptions]	Invalid operation exception (V)				
	Inexact exception (I)				



#### NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)		0 (Integer)		Max Int[V]	-Max Int[V]		

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.


# 2.5.4.43 RSQRTF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Reciprocal Square-Root (single)

**RSQRTF.S4** 

Extended floating-point reciprocal square root (Single precision)

[Instruction format]	RSQRTF.S4 v	wreg2, wreg3															
[Operation]	WR[wreg3](w WR[wreg3](w	$WR[wreg3](w3) \leftarrow 1 \div (sqrt WR[wreg2](w3))$ $WR[wreg3](w2) \leftarrow 1 \div (sqrt WR[wreg2](w2))$ $WR[wreg3](w1) \leftarrow 1 \div (sqrt WR[wreg2](w1))$ $WR[wreg3](w0) \leftarrow 1 \div (sqrt WR[wreg2](w0))$															
[Format]	Format M: 20	OP															
[Opcode]			_														4.0
		1 10	5			31			26			22	1	0.0		17	
	rrrr wreg2			LUL sub-o		W	w w w wreg3	W			1 1 tegory			sub-o	0 (0	0	0
[Descriptions]	single-precisions stores the resu	on approximate on floating-poir ults in the corres lts obtained by e	nt fo spor	ormat co nding el	ntent	s in t of	each el	eme regi	ent stei	of <sup>.</sup> r w	vecto reg3.	r re Th	egist ie re	ter v esult	vreg	2 an	
[Floating-point operation exceptions]	Invalid operation	ted operation ex tion exception ( otion (I) ero exception (2	V)	tion (E)													

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
Operation result [exception]	1/√A[1]	Q-NaN[V]	+∞[Z]	-∞[Z]	+0	Q-NaN[V]	Q-NaN	Q-NaN[V]

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.

## NOTE

The results fall within the error range of 2ULP against the results of calculating  $1/\sqrt{x}$ . ULP: Unit in the Least-significant Place



# 2.5.4.44 SHFLV.W4

<Extended Floating-point Instructions>

		Vector Shuffle
SHFLV.W	4	Vector shuffle
[Instruction format]	SHFLV.W4 imm12, wreg1, wreg2, wreg3	
[Operation]	$\begin{split} & \text{Sel3}[2:0] \parallel \text{Sel2}[2:0] \parallel \text{Sel1}[2:0] \parallel \text{Sel0}[2:0] \leftarrow \text{imm12} \\ & \text{foreach N } (0, 1, 2, 3) \\ & \text{case} \\ & \text{SelN} = 7: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg2}](\text{w3}) \\ & \text{SelN} = 6: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg2}](\text{w2}) \\ & \text{SelN} = 5: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg2}](\text{w1}) \\ & \text{SelN} = 4: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg2}](\text{w0}) \\ & \text{SelN} = 3: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg1}](\text{w3}) \\ & \text{SelN} = 2: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg1}](\text{w2}) \\ & \text{SelN} = 1: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg1}](\text{w1}) \\ & \text{SelN} = 0: \text{WR}[\text{wreg3}](\text{wN}) \leftarrow \text{WR}[\text{wreg1}](\text{w0}) \end{split}$	
[Format]	Format M: Imm12	
[Opcode]		
	15 11 10 5 4 0 31 27	26 21 17 16
	0 0 0 0 0 1 1 1 1 0 1 R R R R R W W W W	1 1 0 1 0 I 1 1 1 0 1
	sub-op wreg1 wreg3	sub-op
	47 43 42 32	
	rrrrriiiiiiiiii	
	wreg2 imm12	
	Where $I =$ the higher-order 1 bit of 12-bit immediate data	
	iiiiiiiiii = The lower-order 11 bits of 12-bit immedia	ata data
	IIIIIIIIII = The lower-order 11 bits of 12-bit minedia	ale data
[Descriptions]	This instruction selects an arbitrary element of vector register the value of the 12-bit immediate data and updates each elem with the results. The 12-bit immediate data is aligned in 3 bit one element to be selected. For example, the element of vector register wreg1 or wreg2 of 12-bit immediate data is stored in the 0th element of vector re	ent of vector register wreg3 units, each of which specifies designated by bits 2-0 of the



# 2.5.4.45 SQRTF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Square-Root (single)

SQRTF.S4

Extended floating-point square-root (Single precision)

[Instruction format]	SQRTF.S4 wreg2, wreg3											
[Operation]	$VR[wreg3](w3) \leftarrow sqrt WR[wreg2](w3)$ $VR[wreg3](w2) \leftarrow sqrt WR[wreg2](w2)$ $VR[wreg3](w1) \leftarrow sqrt WR[wreg2](w1)$ $VR[wreg3](w0) \leftarrow sqrt WR[wreg2](w0)$											
[Format]	Format M: 2OP											
[Opcode]												
	15         11         10         5         4         0         31         27         26         25         23         22         17         16											
	r r r r r 1 1 1 1 1 1 1 1 0 0 1 0 w w w w w 1 0 1 1 0 1 0											
	wreg2 sub-op wreg3 category sub-op											
[Descriptions]	This instruction obtains the positive arithmetic square root of the single-precision floating- point format contents in each element of vector register wreg2 and stores the results in the corresponding element of vector register wreg3. The operation is executed as if it were of infinite accuracy and the result is rounded in accordance with the current rounding mode. If the source operand value is $-0$ , the results become $-0$ .											
[Floating-point operation exceptions]	Unimplemented operation exception (E) Invalid operation exception (V) Inexact exception (I)											

## NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# 2.5.4.46 STV.DW

<Extended Floating-point Instructions>

# STV.DW

Store Vector (Double-Word)

Store double-word data from vector register

[Instruction format] STV.DW imm1, wreg3, disp16[reg1]  $adr \leftarrow GR[reg1] + sign-extend (disp16)^{Note 1}$ [Operation] CheckException(MAE) CheckException(MDP)  $val \leftarrow (imm1 == 1)$ ? WR[wreg3](dw1) : WR[wreg3](dw0) Store-memory (adr, val, Double-word) Note 1. An MAE, or MDP exception might occur depending on the result of address calculation. [Format] Format M: D [Opcode] 15 11 10 54 0 31 27 26 17 16 0 0 0 0 0 1 1 1 1 0 1 RRRRR w w w w w 0 1 1 1 0 i 1 1 1 0 1 sub-op reg1 wreg3 sub-op 47 35 34 33 32 d d d d d d d d d d d 0 0 0

> Where i = 1-bit immediate data ddddddddddd = Higher-order 13 bit of disp16.

disp16



[Descriptions]This instruction adds together the data in general-purpose register reg1 and the 16-bit<br/>displacement data that is sign-extended to word length to generate a 32-bit address. The<br/>double-word data from an arbitrary element of vector register wreg3 is stored in the address<br/>that is generated. The element to be selected is specified by the 1-bit immediate value.

## CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Even when a nonzero value is set in bits 32 to 34, the opcode functions as the STV.DW instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the three lower-order bits were set to 0.



# 2.5.4.47 STV.QW

<Extended Floating-point Instructions>

	Store Vector (Quad-Word)
STV.QW	Store quad-word data from vector register
[Instruction format]	STV.QW wreg3, disp16[reg1]
[Operation]	$adr \leftarrow GR[reg1] + sign-extend (disp16)^{Note 1}$
	CheckException(MAE)
	CheckException(MDP)
	Store-memory (adr, WR[wreg3], Quad-word)
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	
	15 11 10 5 4 0 31 27 26 17 16
	0 0 0 0 0 1 1 1 1 0 1 R R R R R W W W W 0 1 0 1 0 1 1 1 1 0 1
	sub-op reg1 wreg3 sub-op
	47 36 35 34 33 32
	a a a a a a a a a a a a a a a a a a a
	disp16
	Where $dddddddddd = Higher-order 12$ bits of disp16.
[Descriptions]	This instruction adds together the data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length to generate a 32-bit address. The quad-word data from vector register wreg3 is stored in the address that is generated.

# CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- 2. Even when a nonzero value is set in bits 32 to 35, the opcode functions as the STV.QW instruction. An RIE exception does not occur in such cases. Moreover, the displacement operates as if the four lower-order bits were set to 0.



# 2.5.4.48 STV.W

<Extended Floating-point Instructions>

STV.W	Store Vector (Word)
	Store word data from vector register
[Instruction format]	STV.W imm2, wreg3, disp16[reg1]
[Operation]	$adr \leftarrow GR[reg1] + sign-extend (disp16)^{Note 1}$
	CheckException(MAE)
	CheckException(MDP)
	val $\leftarrow$ (imm2 == 0) ? WR[wreg3](w0):
	(imm2 == 1)? WR[wreg3](w1):
	(imm2 == 2) ? WR[wreg3](w2):
	WR[wreg3](w3)
	Store-memory (adr, val, Word)
	<b>Note 1.</b> An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	

15				11	10					5	4				0	31				27	26									17	16
0	0	0	0	0	1	1	1	1	0	1	R	R	R	R	R	w	w	W	w	W	0	1	0	0	i	i	1	1	1	0	1
	S	ub-c	р										reg1				v	vreg	3						sub	о-ор					
47													34	33	32																
d	d	d	d	d	d	d	d	d	d	d	d	d	d	0	0																
							dis	o16																							

Where ii = 2-bit immediate data dddddddddd = Higher-order 14 bits of disp16



[Descriptions]This instruction adds together the data in general-purpose register reg1 and the 16-bit<br/>displacement data that is sign-extended to word length to generate a 32-bit address. The<br/>word data from an arbitrary element of vector register wreg3 is stored in the address that is<br/>generated.

The element to be selected is specified by the 2-bit immediate value.

# CAUTIONS

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- Even when a nonzero value is set in bits 32 and 33, the opcode functions as the STV.W instruction. An RIE
  exception does not occur in such cases. Moreover, the displacement operates as if the two lower-order bits were
  set to 0.



# 2.5.4.49 STVZ.H4

<Extended Floating-point Instructions>

Store Vector at Even Halfword field

Interleaved store halfword from vector register

[Instruction format]	STVZ.H4 wreg3, disp16[reg1]
[Operation]	adr ← GR[reg1] + sign-extend (disp16) <sup>Note 1</sup> CheckException(MAE) CheckException(MDP) val ← WR[wreg3](h6)    WR[wreg3](h4)    WR[wreg3](h2)    WR[wreg3](h0) Store-memory(adr, val, Double-word) Note 1. An MAE, or MDP exception might occur depending on the result of address calculation.
[Format]	Format M: D
[Opcode]	15       11       10       5       4       0       31       27       26       17       16         0       0       0       0       1 <td< td=""></td<>
[Descriptions]	This instruction adds together the data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length to generate a 32-bit address. The lower-order 16 bits that are taken out of each element of vector register wreg3 are stored in the address that is generated.

- 1. A misalignment exception (MAE) will occur if the address calculation results in a misaligned access.
- Even when a nonzero value is set in bits 32 to 34, the opcode functions as the STVZ.H4 instruction. An RIE
  exception does not occur in such cases. Moreover, the displacement operates as if the three lower-order bits were
  set to 0.

# 2.5.4.50 SUBADDF.S4

<Extended Floating-point Instructions>

# SUBADDF.S4

Floating-point SIMD Subtract/Add (single)

Extended floating-point subtract/add (Single precision)

[Instruction format]	SUBADDF.S4 wreg1, wreg2, wreg3													
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) - WR[wreg1](w3)$ $WR[wreg3](w2) \leftarrow WR[wreg2](w2) + WR[wreg1](w2)$													
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) + WR[wreg1](w2)$													
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) - WR[wreg1](w1)$													
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) + WR[wreg1](w0)$													
[Format]	Format M: 3OP													
[Opcode]														
	15 11 10 5 4 0 31 27 26 25 23 22 17 16													
	r r r r r 1 1 1 1 1 1 R R R R R W W W W 1 0 1 1 1 0 1 0 0 1 0													
	wreg2 wreg1 wreg3 category sub-op													
[Descriptions]	This instruction subtracts the single-precision floating-point format contents in the odd- number elements of vector register wreg1 from the single-precision floating-point format contents in the odd-number elements of vector register wreg2 and stores the results in the odd-number elements of vector register wreg3.													
	This instruction also adds together the single-precision floating-point format contents in the even-number elements of vector register wreg1 and the single-precision floating-point format contents in the even-number elements of vector register wreg2 and stores the results in the even-number elements of vector register wreg3.													
	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	Unimplemented operation exception (E)													
exceptions]	Invalid operation exception (V)													
	Inexact exception (I)													
	Overflow exception (O)													
	Underflow exception (U)													



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

[Operation result]

#### <Even-number element>

wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		P	+Α			-∞		
+0		D-	+ A			-80		
-0								
+∞					+∞	Q-NaN[V]		
-∞		-	∞		Q-NaN[V]	- 8		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Odd-number element>

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		Р	- A		+∞			
+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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



#### 2.5.4.51 SUBADDNF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Subtract/Add Negative (single)

Extended floating-point subtract/add negative (Single precision)

[Instruction format]	SUBADDNF.S4 wreg1, wre	SUBADDNF.S4 wreg1, wreg2, wreg3										
[Operation]	$WR[wreg3](w2) \leftarrow neg(WFWR[wreg3](w1) \leftarrow neg(WFWR[wreg3](w1))$	$WR[wreg3](w3) \leftarrow neg(WR[wreg2](w3) - WR[wreg1](w3))$ $WR[wreg3](w2) \leftarrow neg(WR[wreg2](w2) + WR[wreg1](w2))$ $WR[wreg3](w1) \leftarrow neg(WR[wreg2](w1) - WR[wreg1](w1))$ $WR[wreg3](w0) \leftarrow neg(WR[wreg2](w0) + WR[wreg1](w0))$										
[Format]	Format M: 3OP	Format M: 3OP										
[Opcode]												
	15 11 10	5		31	27			22			17	
	rrrr 1 1 1 1 1	1	RRRRR	W V	www	1	0 1 1	1	0 1	1 0	1	0
	wreg2		wreg1		wreg3		category		sub-	∙ор		
[Descriptions]	This instruction subtracts the number elements of vector r contents in the odd-number odd-number elements of vec	regis eler	ster wreg1 fror nents of vector	n the r reg	e single-pro	eci 2 a	sion floa nd stores	ting	g-poin	t fori	mat	
	The instruction also adds tog even-number elements of ve format contents in the even- in the even-number element	ector num	r register wreg nber elements	2 an of ve	d the singl ector regist	e-p ter	precision wreg1 a	floa nd s	ating- tores	point	t	
	The operation is executed as accordance with the current			ite a	ccuracy ar	nd	the resul	t is 1	round	ed in		
[Floating-point operation exceptions]	Unimplemented operation e	-										
exceptionsj	Invalid operation exception (V)											
	Inexact exception (I)											
	Overflow exception (O)											
Underflow exception (U)												



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

[Operation result]

#### <Even-number element>

wreg2(B)	Normal	Normal	.0	0	1.00			C NoN
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(P	+ A)			1.00		
+0		-(В	+ A)			+∞		
-0								
+∞					_∞	Q-NaN[V]		
-∞		+	∞		Q-NaN[V]	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Odd-number element>

wreg2(B)									
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN	
+Normal									
-Normal		(D	•		_∞				
+0		–(B	– A)			+∞			
-0									
+∞		+	.00		Q-NaN[V]				
-∞			_∞		Q-NaN[V]				
Q-NaN				Q-NaN					
S-NaN									

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.52 SUBADDNXF.S4

SUBADDN	VE CA	Floating-point SIMD Su	ubtract/Add N	egative Exchange (sing	gle)				
JUDADDN	_	d floating-point subtract/add	ld negative ex	change (Single precisi	ion)				
[Instruction format]	SUBADDNXF.S4 wreg1, wreg2, wreg								
[Operation]	$WR[wreg3](w3) \leftarrow neg(WR[wreg2](w3))$	) - WR[wreg1](w2))	)						
	$WR[wreg3](w2) \leftarrow neg(WR[wreg2](w2) + WR[wreg1](w3))$								
	$WR[wreg3](w1) \leftarrow neg(WR[wreg2](w1))$	) - WR[wreg1](w0))	)						
	$WR[wreg3](w0) \leftarrow neg(WR[wreg2](w))$	) + WR[wreg1](w1))	)						
[Format]	Format M: 3OP								
[Opcode]									
	15 11 10 5 4	0 31 27 2	26 25 23	22 17	7 16				
	r r r r r 1 1 1 1 1 1 R R	R R W W W W I	1 0 1 1	1 0 1 1 1 1	0				
	wreg2 wr	1 wreg3	category	sub-op					
[Descriptions]	This instruction subtracts the single-pr number elements of vector register wr contents in the odd-number elements of odd-number elements of vector register	1 from the single-pre vector register wreg2	ecision floa 2 and store	ating-point format	t				
	The instruction also adds together the even-number elements of vector regist format contents in the odd-number ele in the even-number elements of vector	wreg2 and the single ents of vector register	e-precision r wreg1 an	floating-point					
	The operation is executed as if it were accordance with the current rounding	-	d the resul	t is rounded in					
[Floating-point operation	Unimplemented operation exception (								
exceptions]	Invalid operation exception (V)								
	Inexact exception (I)								
	Overflow exception (O)								
	Overflow exception (O)								



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

[Operation result]

#### <Even-number element>

wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		(D						
+0		-(Б	+ A)			+∞		
-0								
+∞					_∞	Q-NaN[V]		
-∞		+	∞		Q-NaN[V]	+∞		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Odd-number element>

wreg2(B)									
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN	
+Normal									
-Normal		(D	<b>A</b> )						
+0		-(B	– A)			+∞			
-0									
+∞		+	∞		Q-NaN[V]				
-∞			-∞	Q-NaN[V]					
Q-NaN				Q-NaN					
S-NaN									

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.53 SUBADDXF.S4

<Extended Floating-point Instructions>

ŀ

Floating-point SIMD Subtract/Add Negative Exchange (single)

Extended floating-point subtract/add exchange (Single precision)

[Instruction format]	SUBADDXF.S4 wreg1, wreg2, wreg3								
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) - WR[wreg1](w2)$								
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) + WR[wreg1](w3)$								
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) - WR[wreg1](w0)$								
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) + WR[wreg1](w1)$								
[Format]	Format M: 3OP								
[Opcode]									
	15 11 10 5 4 0 31 27 26 25 23 22 17 16								
	r r r r r 1 1 1 1 1 1 R R R R R w w w w 1 0 1 1 1 0 1 0 1 1 0								
	wreg2 wreg1 wreg3 category sub-op								
[Descriptions]	This instruction subtracts the single-precision floating-point format contents in the even- number elements of vector register wreg1 from the single-precision floating-point format contents in the odd-number elements of vector register wreg2 and stores the results in the odd-number elements of vector register wreg3.								
	The instruction also adds together the single-precision floating-point format contents in the even-number elements of vector register wreg2 and the single-precision floating-point format contents in the odd-number elements of vector register wreg1 and stores the results in the even-number elements of vector register wreg3.								
	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	Unimplemented operation exception (E)								
exceptions]	Invalid operation exception (V)								
	Inexact exception (I)								
	Overflow exception (O)								
	Underflow exception (U)								



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

[Operation result]

#### <Even-number element>

wreg2(B)	+Normal	-Normal	+0	-0	1.00	-∞	Q-NaN	S-NaN
wreg1(A)	+INUITIAI	-inormai	+0	-0	+∞		Q-INAIN	3-INAIN
+Normal								
-Normal		P	+Α			-∞		
+0		B-	+ A			-∞		
-0								
+∞					+∞	Q-NaN[V]		
-∞		_,	$\infty$		Q-NaN[V]	- 8		
Q-NaN							Q-NaN	
S-NaN								Q-NaN[V]

#### <Odd-number element>

wreg2(B)									
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN	
+Normal									
-Normal		Р	- A		1.00				
+0		B	- A		+∞	-∞			
-0									
+∞		-	×		Q-NaN[V]				
-∞		+∞							
Q-NaN							Q-NaN		
S-NaN									

Note 1. [ ] indicates an exception that must occur.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.



# 2.5.4.54 SUBF.S4

SUBF.S4	Floating-point SIMD Subtract (s	ingle
30DF.34	Extended floating-point subtract (Single prec	ision
[Instruction format]	SUBF.S4 wreg1, wreg2, wreg3	
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) - WR[wreg1](w3)$	
	$WR[wreg3](w2) \leftarrow WR[wreg2](w2) - WR[wreg1](w2)$	
	$WR[wreg3](w1) \leftarrow WR[wreg2](w1) - WR[wreg1](w1)$	
	$WR[wreg3](w0) \leftarrow WR[wreg2](w0) - WR[wreg1](w0)$	
[Format]	Format M: 3OP	
[Opcode]		
	15 11 10 5 4 0 31 27 26 25 23 22 ·	17 16
	rrrr111111RRRRR w w w w w 1 0 1 1 0 1 0 0 1	1 0
	wreg2 wreg1 wreg3 category sub-op	
[Descriptions]	This instruction subtracts the single-precision floating-point format contents in vector register wreg1 from the single-precision floating-point format contents in each element vector register wreg2 and stores the results in the corresponding element of vector register wreg3. The operation is executed as if it were of infinite accuracy and the result is roun in accordance with the current rounding mode.	ster
[Floating-point operation	Unimplemented operation exception (E)	
exceptions]	Invalid operation exception (V)	
	Inexact exception (I)	
	Overflow exception (O)	

# NOTE

The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.



[Operation result]

wreg2(B) wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		_						
+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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# 2.5.4.55 SUBRF.S4

<Extended Floating-point Instructions>

SUBRF.S4

Floating-point SIMD Subtract Reduction (single)

Extended floating-point subtract reduction (Single precision)

[Instruction format]	SUBRF.S4 wreg1, wreg2, wreg3	
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreg2](w3) - WR[wreg3](w2) \leftarrow WR[wreg1](w3) - WR[wreg3](w2) \leftarrow WR[wreg1](w3) - WR[wreg3](w1) \leftarrow WR[wreg2](w1) - WR[wreg3](w1) \leftarrow WR[wreg1](w1) - WR[wreg3](w1) \leftarrow WR[wreg1](w1) - WR[wreg3](w1) - WR[wreg3](w1) \leftarrow WR[wreg1](w1) - WR[wreg3](w1) - WR[wreg3](w1$	eg1](w2) eg2](w0)
[Format]	Format M: 3OP	
[Opcode]	15 11 10 5 4 0 31	11 27 26 25 23 22 17 16
	r r r r r 1 1 1 1 1 1 R R R R R W	w w w w w 1 0 1 1 0 1 1 0 1 1 0
	wreg2 wreg1	wreg3 category sub-op
[Descriptions]	This instruction subtracts the single-precision flo number elements of vector register wreg2 from t contents in the odd-number elements of vector re odd-number elements of vector register wreg3.	the single-precision floating-point format
	The instruction also subtracts the single-precisio even-number elements of vector register wreg1 f format contents in the odd-number elements of v in the even-number elements of vector register w	from the single-precision floating-point vector register wreg1 and stores the results
	The operation is executed as if it were of infinite accordance with the current rounding mode.	e accuracy and the result is rounded in
[Floating-point operation exceptions]	Unimplemented operation exception (E) Invalid operation exception (V)	
	Inexact exception (I)	
	Overflow exception (O)	
	Underflow exception (U)	



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

B*4 A*4	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal								
+0		B.	- 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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

**Note 3.** The results of operations performed on the elements will not affect one another.

Note 4. Refer to [Operation] for the operands A and B that are input to produce the output.



#### 2.5.4.56 SUBXF.S4

<Extended Floating-point Instructions>

# SUBXF.S4

Floating-point SIMD Subtract Exchange (single)

Extended floating-point subtract exchange (Single precision)

[Instruction format]	SUBXF.S4 wreg1, wreg2, wr	reg3							
[Operation]	$WR[wreg3](w3) \leftarrow WR[wreward] WR[wreg3](w2) \leftarrow WR[wreward] WR[wreg3](w1) \leftarrow WR[wreward] WR[w$	g2](w2) – WR[v g2](w1) – WR[v	vreg1](w3) vreg1](w0)						
[Format]	Format M: 3OP								
[Opcode]									
		5 4 0 1 R R R R R		26 1	25 23 0 1 1	22	0 0 0 2		0 16
	rrrr11111 wreg2	wreg1	wwwww wreg3	-	category		sub-op	L I	
			in ego	I	outogoly	I			1 1
[Descriptions]	This instruction subtracts the number elements of vector re contents in the odd-number e odd-number elements of vect	gister wreg1 fro lements of vecto	m the single-pr or register wreg	ec	ision floa	ting	-point fo	rmat	t
	The instruction also subtracts number elements of vector re contents in the even-number even-number elements of vec	gister wreg1 fro elements of vect	m the single-pr or register wre	ec	ision floa	ting	-point fo	rmat	t
	The operation is executed as accordance with the current r		nite accuracy a	nd	the resul	t is r	ounded i	n	
[Floating-point operation exceptions]	Unimplemented operation ex Invalid operation exception (	•							
	Inexact exception (I)	•)							
	Overflow exception (O)								
	Underflow exception (U)								



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(B)								
wreg1(A)	+Normal	-Normal	+0	-0	+∞	-∞	Q-NaN	S-NaN
+Normal								
-Normal		Р	- A					
+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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# 2.5.4.57 TRFSRV.W4

<Extended Floating-point Instructions>

# TRFSRV.W4

Transfers compare result to PSW

Vector register flag transfer

[Instruction format]	TRFSRV.W4 imm3, wreg2
[Operation]	$val0 \leftarrow WR[wreg2](w0) == 0 ? 0: 1$ $val1 \leftarrow WR[wreg2](w1) == 0 ? 0: 1$
	$val2 \leftarrow WR[wreg2](w2) == 0 ? 0: 1$ $val3 \leftarrow WR[wreg2](w3) == 0 ? 0: 1$
	$imm3 = 0$ : PSW.Z $\leftarrow$ val0
	$imm3 = 1: PSW.Z \leftarrow val1$ $imm3 = 2: PSW.Z \leftarrow val2$
	imm3 = 3: PSW.Z $\leftarrow$ val3
	$imm3 = 4: PSW.Z \leftarrow val3 \& val2 \& val1 \& val0$ $imm3 = 5: PSW.Z \leftarrow val3   val2   val1   val0$

[Format]

Format M: 2OP

## [Opcode]

_	15				11	10					5	4				0	31				27	26	25		23	22					17	16
	r	r	r	r	r	1	1	1	1	1	1	1	1	1	1	1	0	0	i	i	i	1	0	1	1	0	1	0	0	0	0	0
ĺ		v	vreg	12									s	ub-c	р								са	ateg	ory			sub	o-op			

Where iii = 3-bit immediate data

[Descriptions] This instruction generates a 0 if the value of each element of vector register wreg2 is 0 and a 1 if the value is not 0, performs the operation specified by the 3-bit immediate data, and transfers the result to the Z flag in the PSW.

[Floating-point operation None exceptions]

# NOTE

The value of PSW.Z becomes 0 if 6 or 7 is specified in imm3.



# 2.5.4.58 TRNCF.SUW4

TRNCF.SU	\\//			Floa	ting	J-poin	it SII	MD C	onve	ert Si	ngle	to Ur	sign	ned	Word	d, ro	und 1	owa	rd ze	ero (	single
	VV4			E	Exte	nded	floa	iting-p	point	type	e con	versi	on (S	Sing	gle pr	ecis	ion –	→ Un	sign	ed ir	nteger
[Instruction format]	TRNCF.SUW	l wreg2	2, wı	reg3																	
[Operation]	WR[wreg3](w	3) ← tı	runc	WR[۱	wre	eg2]	(w?	3) (s	ing	le –	→ ur	nsigi	ned	w	ord)	1					
	WR[wreg3](w	2) ← tı	runc	WR[۱	wre	eg2]	(w2	2) (s	ing	le –	→ ur	nsigi	ned	w	ord)	1					
	WR[wreg3](w	1) ← tı	runc	WR[v	wre	eg2]	(w.	l) (s	ing	le –	→ ur	nsign	ned	w	ord)	1					
	WR[wreg3](w	0) ← tı	runc	WR[v	wre	eg2]	(w(	)) (s	ing	le –	→ ur	nsigi	ned	w	ord)						
[Format]	Format M: 20	P																			
[Opcode]																					
	15 11	10		5	4				31				26			23					17 10
	rrrr	1 1 1	L 1	1 1	0	0	0	1 1	W	W	v v	w w	1	0	1	1	0 1	. 0	0	0	0 0
	wreg2					su	b-op	1		w	reg3			C	atego	ry		SL	ıb-op		
[Descriptions]	This instructio in each elemer results in the c	t of ve	ctor	regist	er	wreg	g2 t	to 32	2-bi	t un	sig	ned	inte								
	The results are	-		-					-			-		cur	rent	ro	und	ing	mo	de.	
	When the sour rounded result exception is de	is outs	ide t									-									on
	If invalid oper register is set a according to v	s an in	valic	d oper	ati	on a	nd	no e		-							<i>′</i>				
	• Source is a returned.	positiv	e nu	mber	ou	tside	e th	e va	lue	ran	ge (	of 2 <sup>3</sup>	2_	1 1	to 0,	or	$+\infty$	: 2 <sup>3</sup>	<sup>2</sup> –	l is	
	• Source is a	negativ	/e nu	ımber	, n	ot-a	-nu	mbe	r, o	r –	o: 0	is r	etur	me	ed.						
[Floating-point operation	Unimplemente	d oper;	ation	1 exce	pti	on (	E)														
exceptions]	Invalid operati	on exce	eptic	on (V)																	



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# 2.5.4.59 TRNCF.SW4

<Extended Floating-point Instructions>

				Float	ing-poir	nt SIN	ID Conve	rt Sin	gle	to Word	l, rou	nd to	ward	zero (	single)
TRNCF.SW	/4			E	xtended	l float	ting-point 1	уре о	con	version	(Sing	le pre	ecisio	ז → Ir	nteger)
[Instruction format]	TRNCF.SW4	wreg2, wreg3													
[Operation]	WR[wreg3](w	$3) \leftarrow trunc WF$	<b>R[</b> w:	reg2](v	v3) (si	ngle	$e \rightarrow wo$	rd)							
	WR[wreg3](w	2) ← trunc WI	R[w:	reg2](v	v2) (si	ngle	$e \rightarrow wo$	rd)							
	WR[wreg3](w	1) $\leftarrow$ trunc WI	R[w	reg2](v	v1) (si	ngle	$e \rightarrow wo$	rd)							
	WR[wreg3](w	0) ← trunc WI	<b>R[</b> w:	reg2](v	v0) (si	ngle	$e \rightarrow wo$	rd)							
[Format]	Format M: 2C	Р													
[Opcode]															
	15 11	10	5 4	4	0	31		27	26	25 2	23 22	2			17 16
	rrrrr	1 1 1 1 1	1 (	0 0 0	1 0	w	w w w	w	1	0 1	1 0	1	0 0	0 (	0 0
	wreg2			sub-o	р		wreg3		ĺ	categor	у		sub-c	р	
[Descriptions]	in each element	n arithmeticall nt of vector reg element of vec	ister	r wreg2	to 32	-bit	-								
	The results are	rounded in the	e zei	ro direc	ction, 1	rega	rdless o	f the	e c	urrent	rou	ndin	g m	ode.	
		the operand is in $1^{1} - 1$ to $-2^{31}$ , a													
	register is set a	ation exception as an invalid op alue of the sour	erat	tion and	d no e		-								ł
	<ul> <li>Source is a</li> </ul>	positive numb	er o	r +∞: 2	<sup>31</sup> – 1	is re	eturned.								
	• Source is a	negative numb	er,	not-a-n	umbe	r, or	: —∞: — 2	2 <sup>31</sup> is	s re	eturneo	1.				
[Floating-point operation	Unimplement	ed operation ex	cept	tion (E)	)										
exceptions]	Invalid operat	on exception (	V)												
	Inexact except	ion (I)													



The cause and preservation bit fields of the FXSR register are loaded with the ORs of the floating-point exceptions occurring in the individual elements, respectively.

#### [Operation result]

wreg2(A)	+Normal	-Normal	+0	-0	+∞	-∞	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.

**Note 2.** When FXSR.FS = 1, the subnormal number is flushed to the normalized number which is explained in the hardware manual of the product used.

Note 3. The results of operations performed on the elements will not affect one another.



# Appendix A Number of Instruction Execution Clocks

# A.1 Numbers of Clock Cycles for Execution

Numbers of clock cycles for execution are given in this section. Since the G4MH has a pipelined 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 instructions.



# A.2 Number of G4MH Instruction Execution Clocks

Legend 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

			Instruction	Nu	Imber of Execut	(1/9 ion Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Load instruction	LD.B	disp16[reg1],reg2	4	1	1	3* <sup>1</sup>
	LD.B	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.B	[reg1]+,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.B	[reg1]-,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.BU	disp16[reg1],reg2	4	1	1	3* <sup>1</sup>
	LD.BU	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.BU	[reg1]+,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.BU	[reg1]-,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.DW	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.H	disp16[reg1],reg2	4	1	1	3* <sup>1</sup>
	LD.H	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.H	[reg1]+,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.H	[reg1]-,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.HU	disp16[reg1],reg2	4	1	1	3* <sup>1</sup>
	LD.HU	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.HU	[reg1]+,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.HU	[reg1]-,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.W	disp16[reg1],reg2	4	1	1	3* <sup>1</sup>
	LD.W	disp23[reg1],reg3	6	1	1	3* <sup>1</sup>
	LD.W	[reg1]+,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	LD.W	[reg1]-,reg3	4	1	1	3* <sup>1</sup> , 1* <sup>19</sup>
	SLD.B	disp7[ep],reg2	2	1	1	3* <sup>1</sup>
	SLD.BU	disp4[ep],reg2	2	1	1	3* <sup>1</sup>
	SLD.H	disp8[ep],reg2	2	1	1	3* <sup>1</sup>
	SLD.HU	disp5[ep],reg2	2	1	1	3* <sup>1</sup>
	SLD.W	disp8[ep],reg2	2	1	1	3* <sup>1</sup>



	Mnemonics	onics Operand	Instruction	Num	ber of Executio	n Clocks
Types of Instructions			Length (Number of Bytes)	issue	repeat	latency
Store	ST.B	reg2,disp16[reg1]	4	1	1	1
instruction	ST.B	reg3,disp23[reg1]	6	1	1	1
	ST.B	reg3,[reg1]+	4	1	1	1, 1* <sup>19</sup>
	ST.B	reg3,[reg1]-	4	1	1	1, 1* <sup>19</sup>
	ST.DW	reg3,disp23[reg1]	6	1	1	1
	ST.H	reg2,disp16[reg1]	4	1	1	1
	ST.H	reg3,disp23[reg1]	6	1	1	1
	ST.H	reg3,[reg1]+	4	1	1	1, 1* <sup>19</sup>
	ST.H	reg3,[reg1]-	4	1	1	1, 1* <sup>19</sup>
	ST.W	reg2,disp16[reg1]	4	1	1	1
	ST.W	reg3,disp23[reg1]	6	1	1	1
	ST.W	reg3,[reg1]+	4	1	1	1, 1* <sup>19</sup>
	ST.W	reg3,[reg1]-	4	1	1	1, 1* <sup>19</sup>
	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
Bit manipulation	CLR1	bit#3,disp16[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
instruction	CLR1	reg2,[reg1]	4	1* <sup>22</sup>	1* <sup>22</sup>	8*2,*23
	NOT1	bit#3,disp16[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
	NOT1	reg2,[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
	SET1	bit#3,disp16[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
	SET1	reg2,[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
	TST1	bit#3,disp16[reg1]	4	1*22	1* <sup>22</sup>	8*2,*23
	TST1	reg2,[reg1]	4	1* <sup>22</sup>	1* <sup>22</sup>	8*2,*23
Special instruction	CAXI	[reg1],reg2,reg3	4	10 <sup>*24</sup>	10 <sup>*24</sup>	8 <sup>*2,*17</sup> , 10 <sup>*2,*17</sup>
	DISPOSE*4	imm5,list12	4	N+1* <sup>3</sup>	N+2*3	N+1*3
	LDL.BU	[reg1],reg3	4	12* <sup>24</sup>	12 <sup>*24</sup>	9 <sup>*2,*17</sup> , 12 <sup>*2,*17</sup>
	LDL.HU	[reg1],reg3	4	12* <sup>24</sup>	12*24	9 <sup>*2,*17</sup> , 12 <sup>*2,*17</sup>
	LDL.W	[reg1],reg3	4	12* <sup>24</sup>	12 <sup>*24</sup>	9 <sup>*2,*17</sup> , 12 <sup>*2,*17</sup>
	POPSP	rh-rt	4	N+1*5	N+2*5	N+1*5
	LDM.MP	[reg1], eh-et	4	N+8* <sup>14,*25</sup>	N+8* <sup>14,*25</sup>	N+8* <sup>14,*25</sup>
	PREPARE	list12,imm5	4	N+1* <sup>3</sup>	N+2*3	N+1* <sup>3</sup>
	PREPARE	list12,imm5,sp	4	N+2*3	N+3* <sup>3</sup>	N+2*3
	PREPARE	list12,imm5,imm16	6	N+2* <sup>3</sup>	N+3* <sup>3</sup>	N+2*3
	PREPARE	list12,imm5,imm16<<16	6	N+2* <sup>3</sup>	N+3* <sup>3</sup>	N+2*3
	PREPARE	list12,imm5,imm32	8	N+2*3	N+3* <sup>3</sup>	N+2*3
	PUSHSP	rh-rt	4	N+1*5	N+2*5	N+1*5
	RESBANK		4	25, 30* <sup>20</sup>	25, 30* <sup>20</sup>	25, 30* <sup>20</sup>
	STC.B	reg3,[reg1]	4	8, 12*18,*24	8, 12*18,*24	6, 10* <sup>18</sup>
	STC.H	reg3,[reg1]	4	8, 12*18,*24	8, 12*18,*24	6, 10* <sup>18</sup>



	Mnemonics		Instruction	Num	ber of Executio	n Clocks
Types of Instructions		Operand	Length (Number of Bytes)	issue	repeat	latency
Special	STC.W	reg3,[reg1]	4	8, 12*18,*24	8, 12* <sup>18,*24</sup>	6, 10* <sup>18</sup>
instruction	STM.MP	eh-et, [reg1]	4	N+2* <sup>14,*25</sup>	N+2* <sup>14,*25</sup>	N+2* <sup>14,*25</sup>
Multiplication	MUL	reg1,reg2,reg3	4	1	1	3
instruction	MUL	imm9,reg2,reg3	4	1	1	3
	MULH	reg1,reg2	2	1	1	3
	MULH	imm5,reg2	2	1	1	3
	MULHI	imm16,reg1,reg2	4	1	1	3
	MULU	reg1,reg2,reg3	4	1	1	3
	MULU	imm9,reg2,reg3	4	1	1	3
Multiply-	MAC		4	2	2	4
accumulate operation		reg1,reg2,reg3,reg4				
instruction	MACU	reg1,reg2,reg3,reg4	4	2	2	4
Arithmetic	ADD	reg1,reg2	2	1	1	1
instruction	ADD	imm5,reg2	2	1	1	1
	ADDI	imm16,reg1,reg2	4	1	1	1
	CMP	reg1,reg2	2	1	1	1
	CMP	imm5,reg2	2	1	1	1
	MOV	reg1,reg2	2	1	1	1
	MOV	imm5,reg2	2	1	1	1
	MOV	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
Conditional	ADF	cccc,reg1,reg2,reg3	4	1	1	1
operation instruction	SBF	cccc,reg1,reg2,reg3	4	1	1	1
Saturated	SATADD	reg1,reg2	2	1	1	1
operation	SATADD	imm5,reg2	2	1	1	1
instruction	SATADD	reg1,reg2,reg3	4	1	1	1
	SATSUB	reg1,reg2	2	1	1	1
	SATSUB	reg1,reg2,reg3	4	1	1	1
	SATSUBI	imm16,reg1,reg2	4	1	1	1
	SATSUBR	reg1,reg2	2	1	1	1
Logical	AND	reg1,reg2	2	1	1	1
instruction	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
			1-	1.	1.	1.



			Instruction	Nu	Number of Execution Clocks		
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency	
Data operation	BINS	reg1,pos,width,reg2	4	1	1	1	
nstruction	BSH	reg2,reg3	4	1	1	1	
	BSW	reg2,reg3	4	1	1	1	
	CLIP.B	reg1,reg2	4	1	1	1	
	CLIP.BU	reg1,reg2	4	1	1	1	
	CLIP.H	reg1,reg2	4	1	1	1	
	CLIP.HU	reg1,reg2	4	1	1	1	
	CMOV	cccc,reg1,reg2,reg3	4	1	1	1	
	CMOV	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	
	ROTL	reg1,reg2,reg3	4	1	1	1	
	SAR	reg1,reg2	4	1	1	1	
	SAR	imm5,reg2	2	1	1	1	
	SAR	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	
	SHL	imm5,reg2	2	1	1	1	
	SHL	reg1,reg2,reg3	4	1	1	1	
	SHR	reg1,reg2	4	1	1	1	
	SHR	imm5,reg2	2	1	1	1	
	SHR	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	SCH0L	reg2,reg3	4	1	1	1	
instruction	SCH0R	reg2,reg3	4	1	1	1	
	SCH1L	reg2,reg3	4	1	1	1	
	SCH1R	reg2,reg3	4	1	1	1	
Division	DIV	reg1,reg2,reg3	4	1	19	19	
instruction	DIVH	reg1,reg2	2	1	19	19	
	DIVH	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 operation	DIVQ	reg1,reg2,reg3	4	1	N+3*6	N+3*6	
instruction	DIVQU	reg1,reg2,reg3	4	1	N+3*6	N+3*6	



	Mnemonics	Operand	Instruction	Nun	nber of Execution	on Clocks
Types of Instructions			Length (Number of Bytes)	issue	repeat	latency
Branch	Bcond	disp9	2	2/6*8	2/6*8	2/6*8
instruction*7	Bcond	disp9 (Always)	2	2/3*8	2/3*8	2/3*8
	Bcond	disp17	4	2/6*8	2/6*8	2/6*8
	JARL	disp22,reg2	4	2/3*8	2/3*8	2/3*8
	JARL	disp32,reg1	6	2/3*8	2/3*8	2/3*8
	JARL	[reg1],reg3	4	2/6*8	2/6*8	2/6*8
	JMP	[reg1]	2	2/6*8	2/6*8	2/6*8
	JMP	disp32[reg1]	6	2/7*8	2/7*8	2/7*8
	JR	disp22	4	2/3*8	2/3*8	2/3*8
	JR	disp32	6	2/3*8	2/3*8	2/3*8
Loop instruction	LOOP	reg1,disp16	4	2/6*8	2/6*8	2/6*8
Special	CALLT	imm6	2	17	17	17
instruction	CTRET		4	8	8	8
(with branching)	DISPOSE	imm5,list12,[reg1]	4	N+1/+8*9	N+2/+8*9	N+1/+8*9
	EIRET	_	4	8	8	8
	FERET		4	8	8	8
	FETRAP	vector	2	8	8	8
	RIE		4	8	8	8
	TRAP	vector5	4	8	8	8
	SWITCH	reg1	2	11, 18* <sup>8</sup>	11, 18* <sup>8</sup>	11, 18* <sup>8</sup>
	SYSCALL	vector8	4	17	17	17
Special	DI		4	2	2	2
instruction	EI	_	4	2	2	2
	HALT	_	4	*21	*21	*21
	LDSR	reg2,regID,selID	4	<b>1</b> * <sup>10</sup>	<b>1</b> * <sup>10</sup>	1
	NOP	_	2	1	1	1
	SNOOZE	_	4	*11	*11	*11
	STSR	regID,reg2,selID	4	<b>1</b> * <sup>10</sup>	<b>1</b> * <sup>10</sup>	3
	SYNCE	_	2	1	1	1
	SYNCI	_	2	*12	*12	*12
	SYNCM	_	2	*13	*13	*13
	SYNCP	_	2	*14	*14	*14
Cache	CACHE	cacheop,[reg1]	4	*15	*15	*15
instruction	PREF	prefop,[reg1]	4	*15	*15	*15
Floating-point	ABSF.S	reg2,reg3	4	1	1	4
arithmetic	ADDF.S	reg1,reg2,reg3	4	1	1	4
operation (single	CEILF.SL	reg2,reg3	4	1	1	4
precision)	CEILF.SUL	reg2,reg3	4	1	1	4
	CEILF.SUW	reg2,reg3	4	1	1	4
	CEILF.SW	reg2,reg3	4	1	1	4
	CMOVF.S	cc,reg1,reg2,reg3	4	1	1	4
	CMPF.S	cond,reg1,reg2,cc	4	1	1	1



			Instruction	Nu	mber of Execut	ion Clocks
Types of			Length (Number of			
Instructions	Mnemonics	Operand	Bytes)	issue	repeat	latency
Floating-point	CVTF.HS	reg2,reg3	4	1	1	4
arithmetic operation	CVTF.LS	reg2,reg3	4	1	1	4
(single	CVTF.SH	reg2,reg3	4	1	1	4
precision)	CVTF.SL	reg2,reg3	4	1	1	4
	CVTF.SUL	reg2,reg3	4	1	1	4
	CVTF.SUW	reg2,reg3	4	1	1	4
	CVTF.SW	reg2,reg3	4	1	1	4
	CVTF.ULS	reg2,reg3	4	1	1	4
	CVTF.UWS	reg2,reg3	4	1	1	4
	CVTF.WS	reg2,reg3	4	1	1	4
	DIVF.S	reg1,reg2,reg3	4	8 <sup>*16</sup>	8	11
	FLOORF.SL	reg2,reg3	4	1	1	4
	FLOORF.SUL	reg2,reg3	4	1	1	4
	FLOORF.SUW	reg2,reg3	4	1	1	4
	FLOORF.SW	reg2,reg3	4	1	1	4
	FMAF.S	reg1,reg2,reg3	4	1	1	4
	FMSF.S	reg1,reg2,reg3	4	1	1	4
	FNMAF.S	reg1,reg2,reg3	4	1	1	4
	FNMSF.S	reg1,reg2,reg3	4	1	1	4
	MAXF.S	reg1,reg2,reg3	4	1	1	4
	MINF.S	reg1,reg2,reg3	4	1	1	4
	MULF.S	reg1,reg2,reg3	4	1	1	4
	NEGF.S	reg2,reg3	4	1	1	4
	RECIPF.S	reg2,reg3	4	8* <sup>16</sup>	8	11
	ROUNDF.SL	reg2,reg3	4	1	1	4
	ROUNDF.SUL	reg2,reg3	4	1	1	4
	ROUNDF.SUW	reg2,reg3	4	1	1	4
	ROUNDF.SW	reg2,reg3	4	1	1	4
	RSQRTF.S	reg2,reg3	4	21* <sup>16</sup>	21	24
	SQRTF.S	reg2,reg3	4	14* <sup>16</sup>	14	17
	SUBF.S	reg1,reg2,reg3	4	1	1	4
	TRFSR	cc	4	1	1	5
	TRNCF.SL	reg2,reg3	4	1	1	4
	TRNCF.SUL	reg2,reg3	4	1	1	4
	TRNCF.SUW	reg2,reg3	4	1	1	4
	TRNCF.SW	reg2,reg3	4	1	1	4



			Instruction	Nu	umber of Execut	ion Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Floating-point	ABSF.D	reg2,reg3	4	1	1	4
arithmetic operation	ADDF.D	reg1,reg2,reg3	4	1	1	4
(double	CEILF.DL	reg2,reg3	4	1	1	4
precision)	CEILF.DUL	reg2,reg3	4	1	1	4
	CEILF.DUW	reg2,reg3	4	1	1	4
	CEILF.DW	reg2,reg3	4	1	1	4
	CMOVF.D	cc,reg1,reg2,reg3	4	1	1	4
	CMPF.D	cond,reg1,reg2,cc	4	1	1	1
	CVTF.DL	reg2,reg3	4	1	1	4
	CVTF.DS	reg2,reg3	4	1	1	4
	CVTF.DUL	reg2,reg3	4	1	1	4
	CVTF.DUW	reg2,reg3	4	1	1	4
	CVTF.DW	reg2,reg3	4	1	1	4
	CVTF.LD	reg2,reg3	4	1	1	4
	CVTF.SD	reg2,reg3	4	1	1	4
	CVTF.ULD	reg2,reg3	4	1	1	4
	CVTF.UWD	reg2,reg3	4	1	1	4
	CVTF.WD	reg2,reg3	4	1	1	4
	DIVF.D	reg1,reg2,reg3	4	16* <sup>16</sup>	16	19
	FLOORF.DL	reg2,reg3	4	1	1	4
	FLOORF.DUL	reg2,reg3	4	1	1	4
	FLOORF.DUW	reg2,reg3	4	1	1	4
	FLOORF.DW	reg2,reg3	4	1	1	4
	MAXF.D	reg1,reg2,reg3	4	1	1	4
	MINF.D	reg1,reg2,reg3	4	1	1	4
	MULF.D	reg1,reg2,reg3	4	4	4	7
	NEGF.D	reg2,reg3	4	1	1	4
	RECIPF.D	reg2,reg3	4	16* <sup>16</sup>	16	19
	ROUNDF.DL	reg2,reg3	4	1	1	4
	ROUNDF.DUL	reg2,reg3	4	1	1	4
	ROUNDF.DUW	reg2,reg3	4	1	1	4
	ROUNDF.DW	reg2,reg3	4	1	1	4
	RSQRTF.D	reg2,reg3	4	45* <sup>16</sup>	45	48
	SQRTF.D	reg2,reg3	4	30* <sup>16</sup>	30	33
	SUBF.D	reg1,reg2,reg3	4	1	1	4
	TRNCF.DL	reg2,reg3	4	1	1	4
	TRNCF.DUL	reg2,reg3	4	1	1	4
	TRNCF.DUW	reg2,reg3	4	1	1	4
	TRNCF.DW	reg2,reg3	4	1	1	4



			Instruction	N	umber of Execut	ion Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Extended	ABSF.S4	wreg2,wreg3	4	1	1	1
floating-point arithmetic	ADDF.S4	wreg1,wreg2,wreg3	4	2	2	5
operation	ADDRF.S4	wreg1,wreg2,wreg3	4	2	2	5
	ADDSUBF.S4	wreg1,wreg2,wreg3	4	2	2	5
	ADDSUBNF.S4	wreg1,wreg2,wreg3	4	2	2	5
	ADDSUBNXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	ADDSUBXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	ADDXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	CEILF.SUW4	wreg2,wreg3	4	2	2	5
	CEILF.SW4	wreg2,wreg3	4	2	2	5
	CMOVF.W4	wreg4,wreg1,wreg2,wreg3	6	1	1	1
	CMPF.S4	fcond,wreg1,wreg2,wreg3	4	2	2	5
	CVTF.HS4	wreg2,wreg3	4	2	2	5
	CVTF.SH4	wreg2,wreg3	4	2	2	5
	CVTF.SUW4	wreg2,wreg3	4	2	2	5
	CVTF.SW4	wreg2,wreg3	4	2	2	5
	CVTF.UWS4	wreg2,wreg3	4	2	2	5
	CVTF.WS4	wreg2,wreg3	4	2	2	5
	DIVF.S4	wreg1,wreg2,wreg3	4	16	16	19
	FLOORF.SUW4	wreg2,wreg3	4	2	2	5
	FLOORF.SW4	wreg2,wreg3	4	2	2	5
	FLPV.S4	imm2,wreg2,wreg3	4	1	1	1
	FMAF.S4	wreg1,wreg2,wreg3	4	2	2	5
	FMSF.S4	wreg1,wreg2,wreg3	4	2	2	5
	FNMAF.S4	wreg1,wreg2,wreg3	4	2	2	5
	FNMSF.S4	wreg1,wreg2,wreg3	4	2	2	5
	LDV.DW	imm2,disp16[reg1],wreg3	6	1	1	4
	LDV.QW	disp16[reg1],wreg3	6	1	1	4
	LDV.W	imm4,disp16[reg1],wreg3	6	1	1	4
	LDVZ.H4	disp16[reg1],wreg3	6	1	1	4
	MAXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MAXRF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MINF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MINRF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MOVV.W4	wreg2,wreg3	4	1	1	1
	MULF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MULRF.S4	wreg1,wreg2,wreg3	4	2	2	5
	MULXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	NEGF.S4	wreg2,wreg3	4	1	1	1
	RECIPF.S4	wreg2,wreg3	4	16	16	19
	ROUNDF.SUW4	wreg2,wreg3	4	2	2	5
	ROUNDF.SW4	wreg2,wreg3	4	2	2	5
	RSQRTF.S4	wreg2,wreg3	4	42	42	45



(0/0)

			Instruction	Nu	mber of Execut	ion Clocks
Types of Instructions	Mnemonics	Operand	Length (Number of Bytes)	issue	repeat	latency
Extended	SHFLV.W4	imm12,wreg1,wreg2,wreg3	6	1	1	1
floating-point arithmetic	SQRTF.S4	wreg2,wreg3	4	28	28	31
operation	STV.DW	imm1,wreg3,disp16[reg1]	6	1	1	1
	STV.QW	wreg3,disp16[reg1]	6	1	1	1
	STV.W	imm2,wreg3,disp16[reg1]	6	1	1	1
	STVZ.H4	wreg3,disp16[reg1]	6	1	1	1
	SUBADDF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBADDNF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBADDNXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBADDXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBRF.S4	wreg1,wreg2,wreg3	4	2	2	5
	SUBXF.S4	wreg1,wreg2,wreg3	4	2	2	5
	TRFSRV.W4	imm3,wreg2	4	1	1	1
	TRNCF.SUW4	wreg2,wreg3	4	2	2	5
	TRNCF.SW4	wreg2,wreg3	4	2	2	5

Note 1. If there are no wait states (cycles of waiting) associated with the memory access

Note 2. If there are no wait states (cycles of waiting) associated with the memory access

Note 3. N depends on the total number of registers specified as list12. It is independent of the arrangement of the register numbers. Since up to two registers are handled in one cycle, the value if there are no wait states will be as follows.

PREPARE: The minimum value is 1, and the maximum value is 6

(one clock cycle is also added if the instruction includes updating of the EP register)

DISPOSE: The minimum value is 1, and the maximum value is 6 (two clock cycles are added for JMP)

- Note 4. This is the value of DISPOSE without branching. For details of DISPOSE with branching, refer to [Special instruction (with branching)].
- Note 5. N depends on the total number of registers specified as rh-rt.

Since up to two registers are handled in one cycle, the value if there are no wait states will be as follows.

- PUSHSP: The minimum value is 1, and the maximum value is 16
- POPSP: The minimum value is 1, and the maximum value is 16

Note 6. N = int (((Number of bits in the absolute value of the dividend) - (Number of bits in the absolute value of the divisor)) / 2) + 1. However, when N < 1, N becomes 1, except in the case of division by zero, where N becomes 0. The range of N is 0 to 16.

Note 7. The number to the left of "/" indicates the number of clock cycles for execution at the time of a match with the predicted branch destination, and the number to the right indicates the number of cycles for execution in the case of a non-match with the predicted branch destination. The number of cycles for execution for instructions that do not perform branch prediction is constant.

Note 8. If the branch prediction function is not used, whether it is or is not included, when the condition is met the number will be the same as that for a non-match with the predicted branch destination, and when the condition is not met, it will be the same as that for a match with the predicted branch destination. There is no change in the number of clock cycles even when the instruction is immediately preceded by an instruction to rewrite the contents of the PSW.

Note 9. In a DISPOSE instruction with JMP, the branch is predicted. In the case of a non-match with the predicted branch destination, the values for "issue" and "repeat" will be the same. Refer to Note 3 for the value of N.

Note 10. SellD = 0, 1, 2, 3, 5 (however, regID is 15 or less). Access to system register 10, 13 stops the issuing of subsequent instruction. Otherwise, operation is with "issue" = 1, "repeat" = 1.

Note 11. Depends on the setting for operation of the SNOOZE instruction.

Note 12. Performs processing to synchronize instruction fetching.



- Note 13. Performs processing to synchronize memory access.
- Note 14. Performs processing to synchronize pipeline.
- Note 15. The instruction execution is completed, but completion of the internal processing depends on the internal state of the instruction fetching unit.
- Note 16. The "issue" value for instructions other than those involving floating point division (DIVF, RECIPF, RSQRTF, SQRTF) will be 1.
- Note 17. The number of cycles differs according to whether the point for reference to the result of executing instruction is not or is the load store unit, and are shown in that order.
- Note 18. The number of cycles differs according to whether the STC instruction succeeds or fails. The order of the numbers is that for failure, and then that for success.
- Note 19. In the case of reference to the result of updating the base address.
- Note 20. The number of cycles depends on the RBCR0.MD value. The left value for RBCR0.MD = 0, and the right value for RBCR0.MD = 1.
- Note 21. After waiting like the one due to the SYNCM instruction has finished, the execution changes to a HALT state.
- Note 22. The subsequent instruction is accepted, but the blocking operation is performed for 6 cycles or more in memory access after storing data to the store buffer inside the load store unit.
- Note 23. To use the flag result, one-cycle penalty is always added to the latency.
- Note 24. The number of execution clock cycles for "issue" and "repeat" of these instructions differs greatly from that of the CPU of G3MH. This is because the pipelined architecture of the CPU of G4MH has been changed from that of G3MH, and therefore measurement of execution clock cycles for "issue" and "repeat" of these instructions requires inclusion of the number of bus access cycles.
- Note 25. N is included in the total number of MPU entries specified in eh-et. Each entry has 3 registers, and since up to two registers are processed in one cycle, the value if there are no wait states will be as follows.

N = int (Number of saved and restored MPU entries x 1.5 + 0.5); however, N is in the range of 0 to 32.



# **REVISION HISTORY**

# RH850G4MH User's Manual: Software

Page	Description	Classification
_	Unification of r01us0209ej0100 and r01us0431ej0050 with improvement Classification	_



# RH850G4MH User's Manual: Software

Publication Date:	Rev.1.00	Mar 30, 2018	
	Rev.2.20	December 20, 2023	
Published by:	Renesas Electronics Corporation		

# RH850G4MH

