User's Manual

32

RH850G4MH

User's Manual: Software

Renesas microcontroller

Renesas Electronics
www.renesas.com

Rev.2.20 Dec. 2023

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:
www.renesas.com/contact/

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 reaches the level at which resetting 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 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_H
RH850G4MH2	07_H

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	● High performance 32-bit architecture for embedded control ● 32-bit internal data bus ● Thirty-two 32-bit general-purpose registers ● RISC type instruction set (backward compatible with V850, V850E1, V850E2, RH850G3M, and RH850G3MH) 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	● A floating point operation coprocessor (FPU) can be installed. Supports single precision (32-bit) and double precision (64-bit) Supports IEEE754-compliant data types and exceptions Rounding modes: Nearest, 0 direction, +∞ direction, and −∞ direction Handling on non-normalized numbers: Truncated to 0, or an exception is reported to comply with IEEE754. ● An extended floating-point operation coprocessor (FXU) can be installed. Supports 4 single-precision (32-bit) parallel operations Supports IEEE754-compliant data types and exceptions Rounding modes: Nearest, 0 direction, +∞ direction, and −∞ direction Handling on non-normalized numbers: Truncated to 0, or an exception is reported to comply with IEEE754.

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

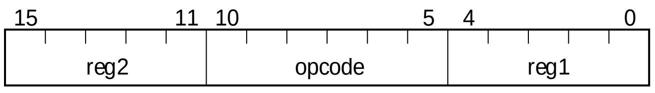
Instructions classified as CPU instructions are allocated in the opcode area other than the area used in the format of the coprocessor instructions shown in **Section 2.1.2, Coprocessor Instructions**.

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**.

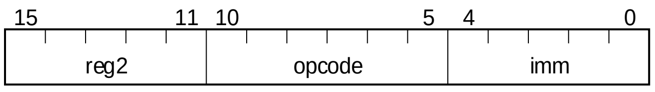
(1) reg-reg Instruction (Format I)

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



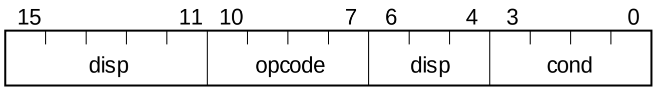
(2) imm-reg Instruction (Format II)

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



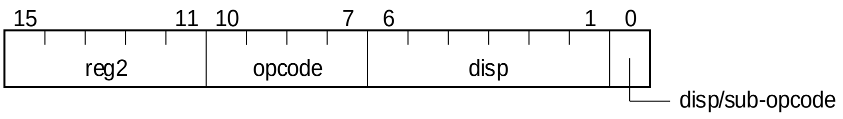
(3) Conditional Branch Instruction (Format III)

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

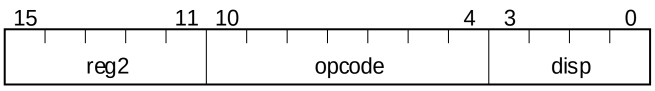


(4) 16-Bit Load/Store Instruction (Format IV)

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).

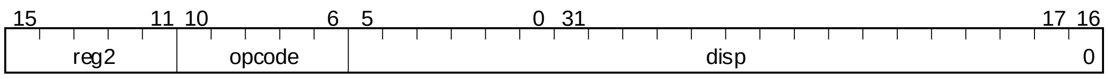


In addition, a 16-bit instruction format consists of a 7-bit opcode field, a general-purpose register specification field, and a 4-bit displacement field.



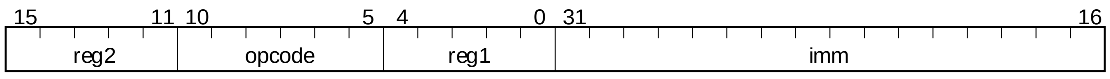
(5) Jump Instruction (Format V)

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



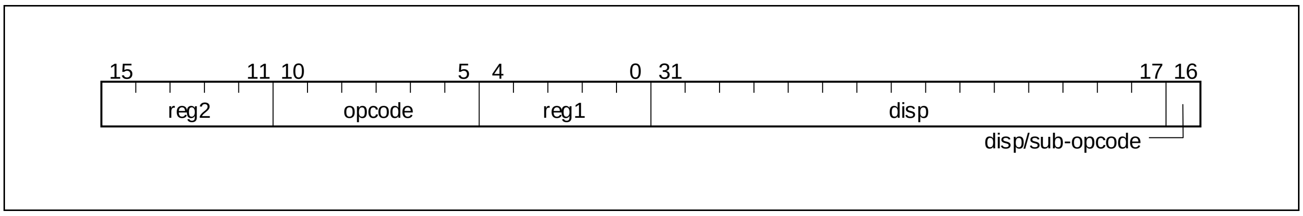
(6) 3-Operand Instruction (Format VI)

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



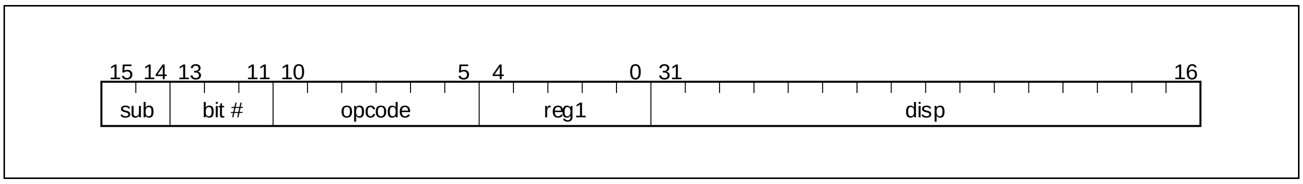
(7) 32-Bit Load/Store Instruction (Format VII)

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.

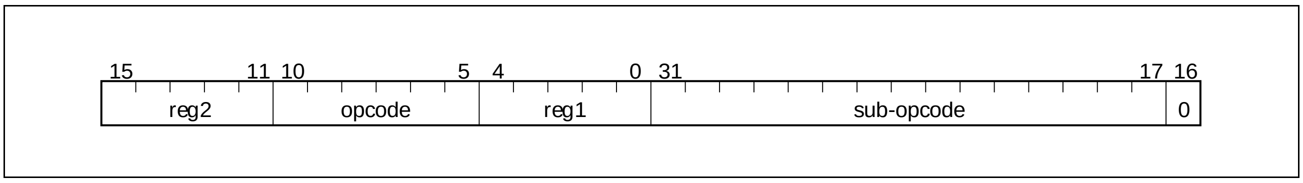


(9) Extended Instruction Format 1 (Format IX)

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**.

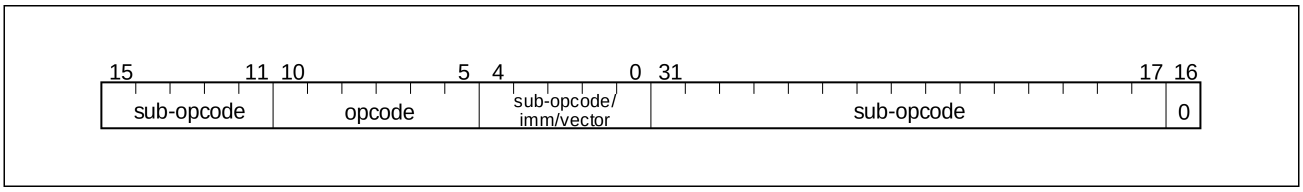


(10) Extended Instruction Format 2 (Format X)

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**.

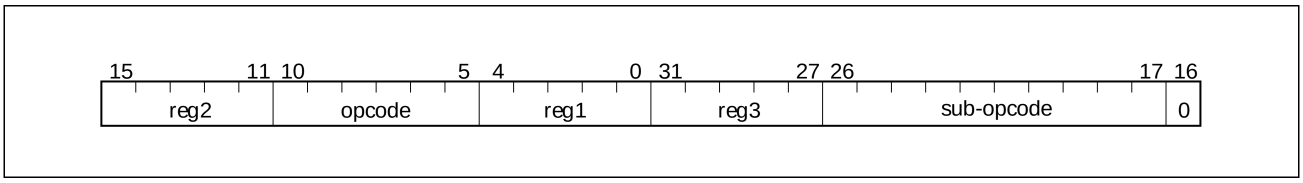


(11) Extended Instruction Format 3 (Format XI)

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**.

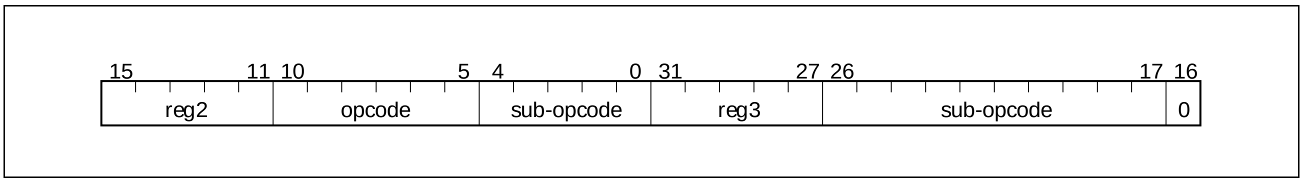


(12) Extended Instruction Format 4 (Format XII)

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

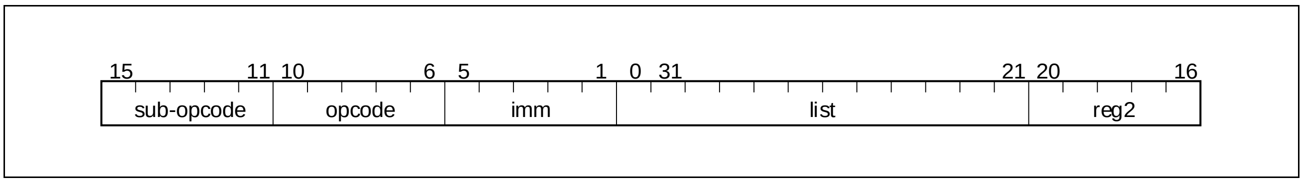
Extended instruction format 4 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in **Section 2.2.3, Basic Instruction Set**.



(13) Stack Manipulation Instruction Format (Format XIII)

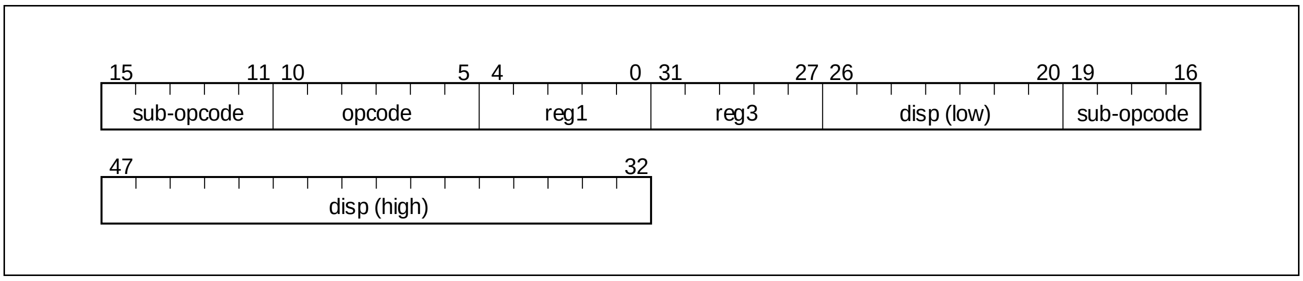
A 32-bit instruction format consists of a 5-bit opcode field, 5-bit immediate field, 12-bit register list field, 5-bit sub-opcode field, and one general-purpose register specification field (or 5-bit sub-opcode field).

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.

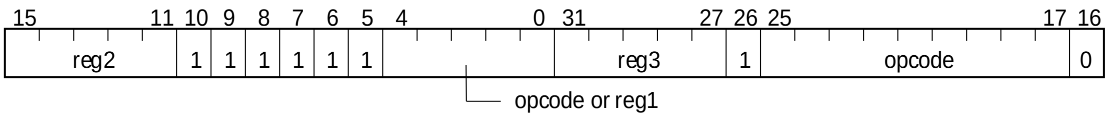


2.1.2 Coprocessor Instructions

Instructions in the following format are defined as coprocessor instructions.

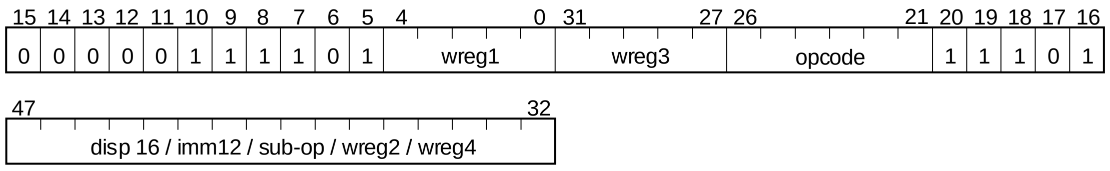
(1) Coprocessor Instruction Format 1

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.

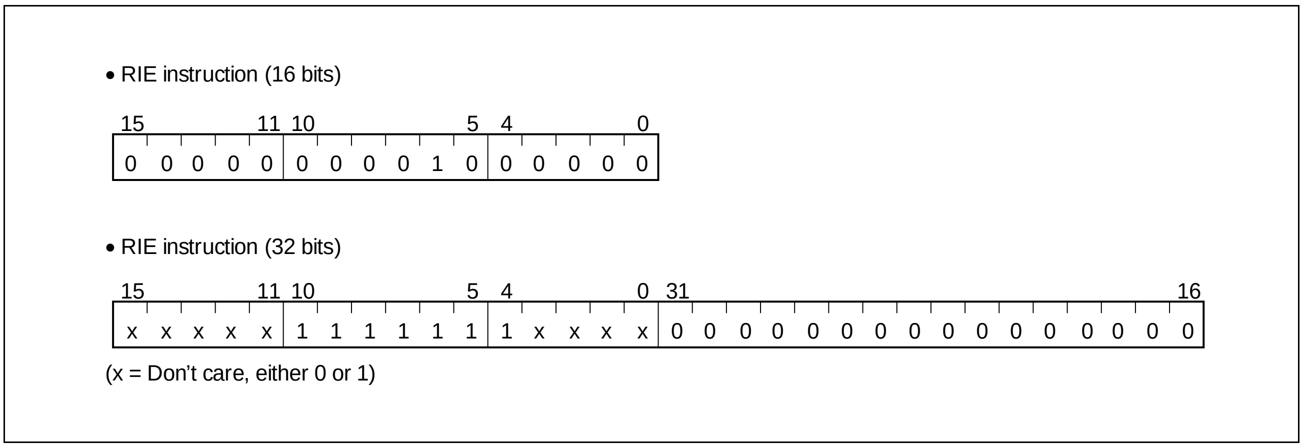
In addition, since instruction codes in a coprocessor instruction format 2 are all handled as an extended floating-point arithmetic instruction, the exception cause code will be 81_H.

Instruction Code						Corresponding Coprocessor	Exception Cause Code
Bit 26	Bit 25	Bit 24	Bit 23	Bit 22	Bit 21		
1	0	0	0	—	—	FPU	80_H
1	0	0	1	0	0	FPU	80_H
				0	1	FPU	80_H
				1	0	FXU	81_H
				1	1	FPU	80_H
1	0	1	0	—	—	Reserved	82_H
1	0	1	1	—	—	FXU	81_H
1	1	0	0	—	—	Reserved	82_H
1	1	0	1	—	—	Reserved	82_H
1	1	1	0	—	—	Reserved	82_H
1	1	1	1	—	—	Reserved	82_H

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:


• RIE instruction (16 bits)
15 11 10 5 4 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
• RIE instruction (32 bits)
15 11 10 5 4 0 31 16
x x x x x 1 1 1 1 1 1 1 x x x x 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(x = Don't care, either 0 or 1)

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_H), 7FFF FFFF_H returns. If the operation result exceeds the maximum negative value (8000 0000_H), 8000 0000_H returns. The following instructions (mnemonics) are provided.

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

(8) Logical Instructions

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

• AND:	AND
• ANDI:	AND immediate
• NOT:	NOT
• OR:	OR
• ORI:	OR immediate
• TST:	Test
• XOR:	Exclusive OR
• XORI:	Exclusive OR immediate

(9) Data Manipulation Instructions

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

• BINS:	Bitfield Insert
• BSH:	Byte swap halfword
• BSW:	Byte swap word
• 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

- 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_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 ×	×-bit immediate data
disp ×	×-bit displacement data
regID	System register number
selID	System register selection ID
vector ×	Data to specify vector (× 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)
ep	Element pointer (r30)
list12	Lists of registers
rh-rt	Indicates multiple general-purpose registers, from the general-purpose register indicated by *rh* to the general-purpose register indicated by *rt*.
eh-et	Indicates multiple system registers of MPU entry (MPLA, MPUA, MPAT), from the entry number indicated by *eh* to the entry number indicated by *et*.
[ ]+	Post increment addressing
[ ]−	Post decrement addressing

Table 2.2 Conventions of Operation (1/2)

Symbol	Meaning
←	Assignment
GR [a]	Value stored in general-purpose register *a*
SR [a, b]	Value stored in system register (RegID = *a*, SelID = *b*)
(n:m)	Bit selection. Select from bit *n* to bit *m*.
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 Conventions 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 *b* from address *a*
store-memory (a, b, c)	Writes data *b* of size *c* to address *a*
extract-bit (a, b)	Extracts value of bit *b* of data *a*
set-bit (a, b)	Sets value of bit *b* of data *a*
not-bit (a, b)	Inverts value of bit *b* of data *a*
clear-bit (a, b)	Clears value of bit *b* of data *a*
saturated (n)	Performs saturated processing of “n”. If n ≥ 7FFF FFFF$_H$, n = 7FFF FFFF$_H$. If n ≤ 8000 0000$_H$, n = 8000 0000$_H$.
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”. ● If the sign “b” is Sign and the size “c” is Byte: When 0000 007F$_H$ < a ≤ 7FFF FFFF$_H$, the result is 0000 007F$_H$. When 8000 0000$_H$ ≤ a < FFFF FF80$_H$, the result is FFFF FF80$_H$. ● If the sign “b” is Unsign and the size “c” is Byte: When 0000 00FF$_H$ < a, the result is 0000 00FF$_H$. ● If the sign “b” is Sign and the size “c” is Halfword: When 0000 7FFF$_H$ < a ≤ 7FFF FFFF$_H$, the result is 0000 7FFF$_H$. When 8000 0000$_H$ ≤ a < FFFF 8000$_H$, the result is FFFF 8000$_H$. ● If the sign “b” is Unsign and the size “c” is Halfword: When 0000 FFFF$_H$ < a, the result is 0000 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.

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
I	1-bit data of immediate (indicates higher bits of immediate)
i	1-bit data of immediate
V	1-bit data of code specifying vector (indicates higher bits of vector)
v	1-bit data of code specifying vector
cccc	4-bit data for condition code specification (See **Table 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
P	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	H	(CY or Z) = 0
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	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] ← GR[reg2] + GR[reg1]
	(2) GR[reg2] ← GR[reg2] + sign-extend (imm5)
[Format]	(1) Format I
	(2) Format II

[Opcode]

(1)

```
15              0
rrrrr001110RRRRR
```

(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] ← GR [reg1] + sign-extend (imm16)

[Format] Format VI

[Opcode]

15 0	31 16
rrrrr110000RRRRR	iiiiiiiiiiiiiiii

[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 word data of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.3 ADF

<Conditional Operation Instructions>

ADF	Add on condition flag Conditional add

[Instruction format] ADF cccc, reg1, reg2, reg3

[Operation]
```
if conditions are satisfied
then GR [reg3] ← GR [reg1] + GR [reg2] + 1
else GR [reg3] ← GR [reg1] + GR [reg2] + 0
```

[Format] Format XI

[Opcode]

15 0	31 16
rrrrr111111RRRRR	wwwww011101cccc0

[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 general-purpose 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
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	H	(CY or Z) = 0

Condition Code	Name	Condition Formula
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	Always (Unconditional)
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
(1101)	Setting prohibited	

2.2.3.4 AND

<Logical instruction>

AND	AND AND

[Instruction format]	AND reg1, reg2
[Operation]	GR[reg2] ← GR[reg2] AND GR[reg1]
[Format]	Format I
[Opcode]	

15	0
rrrrr001010RRRRR	

[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] ANDs the word data of general-purpose register reg2 with the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.5 ANDI

<Logical instruction>

AND immediate

ANDI

AND immediate

[Instruction format]	ANDI imm16, reg1, reg2
[Operation]	GR[reg2] ← GR[reg1] AND zero-extend (imm16)
[Format]	Format VI
[Opcode]	

15 0	31 16
`rrrrr110110RRRRR`	`iiiiiiiiiiiiiiii`

[Flags]

CY	—
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 immediate data, zero-extended to word length, and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.6 Bcond

<Branch instruction>

Bcond	Branch on condition code Conditional branch

[Instruction format] (1) Bcond disp9

(2) Bcond disp17

[Operation] (1) if conditions are satisfied
then PC ← PC + sign-extend (disp9)

(2) if conditions are satisfied
then PC ← PC + sign-extend (disp17)

[Format] (1) Format III

(2) Format VII

[Opcode]

(1)

```
15              0
ddddd1011dddcccc
```

`dddddddd` is the higher 8 bits of disp9.
`cccc` is the condition code of the condition indicated by cond (For details, see **Table 2.5 Bcond Instructions**).

(2)

```
15              0 31              16
00000111111Dcccc  ddddddddddddddd1
```

`Ddddddddddddddd` is the higher 16 bits of disp17.
`cccc` is the condition code of the condition indicated by cond (For details, see **Table 2.5 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.

[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.

Table 2.5 Bcond Instructions

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

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. Unlike the normal 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

<Data manipulation instruction>

BINS	Bitfield Insert Insert bit in register

[Instruction format] BINS reg1, pos, width, reg2

[Operation] GR[reg2] ← GR[reg2] (31:width+pos) || GR[reg1] (width-1:0) || GR[reg2] (pos-1:0)

[Format] Format IX

[Opcode]

15 0	31 16	
`rrrrr111111RRRRR`	`MMMMK0001001LLL0`	(msb ≥ 16, lsb ≥ 16)

15 0	31 16	
`rrrrr111111RRRRR`	`MMMMK0001011LLL0`	(msb ≥ 16, lsb < 16)

15 0	31 16	
`rrrrr111111RRRRR`	`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]

CY	—
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

<Data manipulation instruction>

BSH	Byte swap halfword Byte swap of halfword data

[Instruction format] BSH reg2, reg3

[Operation] GR[reg3] ← GR[reg2] (23:16) || GR[reg2] (31:24) || GR[reg2] (7:0) || GR[reg2] (15:8)

[Format] Format XII

[Opcode]

```
15                0 31              16
rrrrr11111100000  wwwww01101000010
```

[Flags] CY	"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] Executes endian swap.

2.2.3.9 BSW

<Data manipulation instruction>

Byte swap word

BSW

Byte swap of word data

[Instruction format]	BSW reg2, reg3
[Operation]	GR[reg3] ← GR[reg2] (7:0) || GR[reg2] (15:8) || GR[reg2] (23:16) || GR[reg2] (31:24)
[Format]	Format XII

[Opcode]

15 0	31 16
`rrrrr11111100000`	`wwwww01101000000`

[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] Executes endian swap.

2.2.3.10 CALLT

<Special instruction>

Call with table look up

CALLT

Subroutine call with table look up

[Instruction format]	CALLT imm6

[Operation]	adr ← CTBP + zero-extend (imm6 logically shift left by 1)$^{\text{Note 1}}$
	CheckException(MDP)
	CTPC ← PC + 2 (return PC)
	CTPSW(4:0) ← PSW(4:0)
	PC ← CTBP + zero-extend (Load-memory (adr, Halfword))
	Note 1. An MDP exception might occur depending on the result of address calculation.

[Format]	Format II
[Opcode]	15 0 `0000001000iiiiii`
[Flags]	CY — OV — S — Z — SAT —

[Description]	The following steps are taken.

<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 zero-extend 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]Note 1
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
```

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

[Format] Format XI

[Opcode]

15　　0	31　　16
`rrrrr111111RRRRR`	`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 general-purpose register reg2, and the result is indicated by flags in the PSW. Comparison is performed by subtracting the read word data from the word data in general-purpose register reg2. If the comparison result is “0”, word data in general-purpose register reg3 is stored in the generated address, otherwise the read word data is stored in the generated address. Afterward, the read word data is stored in general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.

CAUTIONS

1. 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] ← clip (GR[reg1], Sign, Byte)

[Format] Format IX

[Opcode]

15 0	31 16
rrrrr111111RRRRR	0000000000001000

[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 $FF80_H$ which is the negative minimum value, the instruction stores FFFF $FF80_H$ 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

When clearing the SAT flag to 0, load data in the PSW with the LDSR instruction.

2.2.3.13 CLIP.BU

<Data manipulation instruction>

CLIP.BU	Unsigned data conversion from word to byte with saturation (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
`rrrrr111111RRRRR`	`0000000000001010`

[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 byte data. If the value of the original data exceeds 0000 $00FF_H$ which is the maximum value of the byte data, the instruction stores 0000 $00FF_H$ 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

When clearing the SAT flag to 0, load data in the PSW with the LDSR instruction.

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
`rrrrr111111RRRRR`	`0000000000001100`

[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 $7FFF_H$ which is the positive maximum value of halfword data, the instruction stores 0000 $7FFF_H$ 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

When clearing the SAT flag to 0, load data in the PSW with the LDSR instruction.

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`	`0000000000001110`

[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 FFFF$_H$ which is the maximum value of the halfword data, the instruction stores 0000 FFFF$_H$ 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

When clearing the SAT flag to 0, load data in the PSW with the LDSR instruction.

2.2.3.16 CLL

<Special instruction>

CLL	Clear Load Link Clear atomic manipulation link

[Instruction format] CLL

[Operation] LLbit ← 0

[Format] Format X

[Opcode]

15 0	31 16
1111111111111111	1111000101100000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[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

<Bit manipulation instruction>

CLR1

Clear bit

Bit clear

[Instruction format]

(1) CLR1 bit#3, disp16[reg1]

(2) CLR1 reg2, [reg1]

[Operation]

(1)
```
adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, bit#3))
token ← clear-bit (token, bit#3)
Store-memory (adr, token, Byte)
```

(2)
```
adr ← GR[reg1]Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← clear-bit (token, reg2)
Store-memory (adr, token, Byte)
```

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

[Format]

(1) Format VIII

(2) Format IX

[Opcode]

(1)

15 0	31 16
10bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100100

[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]	The Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is executed, and does not indicate the content of the specified bit after this instruction is executed.

CAUTION

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

<Data manipulation instruction>

CMOV	Conditional move Conditional move

[Instruction format] (1) CMOV cccc, reg1, reg2, reg3

(2) CMOV cccc, imm5, reg2, reg3

[Operation] (1) if conditions are satisfied
then GR[reg3] ← GR[reg1]
else GR[reg3] ← GR[reg2]

(2) if conditions are satisfied
then GR[reg3] ← sign-extended (imm5)
else GR[reg3] ← GR[reg2]

[Format] (1) Format XI

(2) Format XII

[Opcode]

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww011001cccc0

(2)

15 0	31 16
rrrrr111111iiiii	wwwww011000cccc0

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

Condition 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	T	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	H	(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	T	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	H	(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>

CMP	Compare register/immediate (5-bit) Compare

[Instruction format]	(1)	CMP reg1, reg2
	(2)	CMP imm5, reg2
[Operation]	(1)	result ← GR[reg2] − GR[reg1]
	(2)	result ← GR[reg2] − sign-extend (imm5)
[Format]	(1)	Format I
	(2)	Format II

[Opcode]

(1)
```
15              0
rrrrr001111RRRRR
```

(2)
```
15              0
rrrrr010011iiiii
```

[Flags]	CY	"1" if a borrow occurs from MSB; otherwise, "0".
	OV	"1" if overflow occurs; otherwise, "0".
	S	"1" if the operation result is negative; otherwise, "0".
	Z	"1" if the operation result is "0"; otherwise, "0".
	SAT	—

[Description]

(1) Compares the word data of general-purpose register reg2 with the word data of general- purpose register reg1 and outputs the result through the PSW flags. Comparison is performed by subtracting the reg1 contents from the reg2 word data. General-purpose registers reg1 and reg2 are not affected.

(2) Compares the word data of general-purpose register reg2 with the 5-bit immediate data, sign-extended to word length, and outputs the result through the PSW flags. Comparison is performed by subtracting the sign-extended immediate data from the reg2 word data. General-purpose register reg2 is not affected.

2.2.3.20 CTRET

<Special instruction>

CTRET	Return from CALLT Return from subroutine call

[Instruction format] CTRET

[Operation] PC ← CTPC

PSW (4:0) ← CTPSW (4:0)

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000101000100

[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] Loads the return PC and PSW (the lower 5 bits) from the appropriate system register and returns from a routine under CALLT instruction. The following steps are taken:

<1> The return PC and the return PSW (the lower 5 bits) are loaded from the CTPC and CTPSW (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 ← 1 (Disables EI level maskable exception)

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000101100000

[Flags]

CY	—
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>

DISPOSE	Function dispose Restore registers from stack

[Instruction format]

(1) DISPOSE imm5, list12

(2) DISPOSE imm5, list12, [reg1]

[Operation]

(1)
```
tmp ← sp + zero-extend (imm5 logically shift by 2)
foreach (all regs in list12) {
        adr ← tmp[Note 1, Note 2]
        CheckException(MDP)
        GR[reg in list12] ← Load-memory (adr, Word)
        tmp ← tmp + 4
}
sp ← tmp
```

(2)
```
tmp ← sp + zero-extend (imm5 logically shift by 2)
foreach (all regs in list12) {
        adr ← tmp[Note 1, Note 2]
        CheckException(MDP)
        GR[reg in list12] ← Load-memory (adr, Word)
        tmp ← tmp + 4
}
PC ← GR[reg1]
sp ← tmp
```

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

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

[Format] Format XIII

[Opcode]

```
     15             0 31             16
(1) |0000011001iiiiiL|LLLLLLLLLLL00000|

     15             0 31             16
(2) |0000011001iiiiiL|LLLLLLLLLLLRRRRR|
```

RRRRR ≠ 00000 (Do not specify r0 for reg1.)
The values of `LLLLLLLLLLLL` 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_H
- When all of the register’s non-corresponding bits are “1”: 081F $FFFF_H$

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

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

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

CAUTIONS

1. 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] ← GR[reg2] ÷ GR[reg1]
	GR[reg3] ← GR[reg2] % GR[reg1]

[Format]	Format XI

[Opcode]

15 0	31 16
rrrrr111111RRRRR	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 **Section 2.2.2, Special Operations**.

[Supplement]	Overflow occurs when the maximum negative value (8000 0000_H) is divided by –1 with the quotient = 8000 0000_H and when the data is divided by 0. If reg2 and reg3 are the same register, the remainder is stored in that register. When an exception occurs during the DIV instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

CAUTION

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>

DIVH	Divide halfword / Division of (signed) halfword data

[Instruction format]

(1) DIVH reg1, reg2

(2) DIVH reg1, reg2, reg3

[Operation]

(1) GR[reg2] ← GR[reg2] ÷ sign-extend (GR[reg1] (15:0))

(2) GR[reg2] ← GR[reg2] ÷ sign-extend (GR[reg1] (15:0))
GR[reg3] ← GR[reg2] % sign-extend (GR[reg1] (15:0))

[Format]

(1) Format I

(2) Format XI

[Opcode]

(1)

15 0
`rrrrr000010RRRRR`

RRRRR ≠ 00000 (Do not specify r0 for reg1.)
`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01010000000`

[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 **Section 2.2.2, Special Operations**.

(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 **Section 2.2.2, Special Operations**.

[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>

Divide halfword unsigned

DIVHU

Division of (unsigned) halfword data

[Instruction format] DIVHU reg1, reg2, reg3

[Operation] GR[reg2] ← GR[reg2] ÷ zero-extend (GR[reg1] (15:0))

GR[reg3] ← GR[reg2] % zero-extend (GR[reg1] (15:0))

[Format] Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01010000010`

[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] 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 **Section 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 remainder is stored in that register. When an exception occurs during the DIVHU instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

CAUTION

If general-purpose registers reg2 and reg3 are specified as being the same register, the operation result quotient is not stored in reg2, so the flag is undefined.

2.2.3.26 DIVQ

<High-speed divide instructions>

Divide word quickly

DIVQ

Division of (signed) word data (variable steps)

[Instruction format]	DIVQ reg1, reg2, reg3

[Operation]	GR[reg2] ← GR[reg2] ÷ GR[reg1]
	GR[reg3] ← GR[reg2] % GR[reg1]

[Format]	Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01011111100`

[Flags]

CY	—
OV	"1" when overflow occurs; otherwise, "0".
S	"1" when operation result quotient is a negative value; otherwise, "0".
Z	"1" when operation result quotient is a "0"; otherwise, "0".
SAT	—

[Description]

Divides the word data in general-purpose register reg2 by the word data in general-purpose register reg1, stores the quotient in reg2, and stores the remainder in general-purpose register reg3. General-purpose register reg1 is not affected.

The minimum number of steps required for division is determined from the values in reg1 and reg2, then this operation is executed.

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_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 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.

2.2.3.27 DIVQU

<High-speed divide instructions>

Divide word unsigned quickly

DIVQU

Division of (unsigned) word data (variable steps)

[Instruction format]	DIVQU reg1, reg2, reg3

[Operation]	GR[reg2] ← GR[reg2] ÷ GR[reg1]
	GR[reg3] ← GR[reg2] % GR[reg1]

[Format]	Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01011111110`

[Flags]

CY	—
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 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.

2.2.3.28 DIVU

<Divide instruction>

DIVU	Divide word unsigned Division of (unsigned) word data

[Instruction format] DIVU reg1, reg2, reg3

[Operation] GR[reg2] ← GR[reg2] ÷ GR[reg1]

GR[reg3] ← GR[reg2] % GR[reg1]

[Format] Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`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 **Section 2.2.2, 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

<Special instruction>

EI	Enable interrupt Enable EI level maskable exception

[Instruction format] EI

[Operation] PSW.ID ← 0 (enables EI level maskable exception)

[Format] Format X

[Opcode]

15 … 0	31 … 16
1000011111100000	0000000101100000

[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

<Special instruction>

EIRET

Return from EI level trap or interrupt

Return from EI level exception

[Instruction format]	EIRET

[Operation]	PC ← EIPC
	PSW ← EIPSW

[Format]	Format X

[Opcode]

15 0	31 16
`0000011111100000`	`0000000101001000`

[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*n*) 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 interrupt Return from FE level exception

[Instruction format]	FERET

[Operation]	PC ← FEPC
	PSW ← FEPSW

[Format]	Format X

[Opcode]

15 0	31 16
0000011111100000	0000000101001010

[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 from the FEPC and FEPSW registers and set in the PC and PSW, and control is passed.

[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

<Special instruction>

FETRAP	FE-level Trap FE level software exception

[Instruction format] FETRAP vector4

[Operation]

FEPC ← PC + 2 (return PC)

FEPSW ← PSW

FEIC ← exception cause code[Note 1]

PSW.UM ← 0

PSW.NP ← 1

PSW.EP ← 1

PSW.ID ← 1

PC ← exception handler address[Note 1]

Note 1. 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 ≠ 0000).

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Saves the contents of the return PC (address of the instruction next to the FETRAP instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores the exception cause code in the FEIC register, and updates the PSW according to the exception causes listed in 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.

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

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

2.2.3.33 HALT

<Special instruction>

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
0000011111100000	0000000100100000

[Flags]

CY	—
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[Note 1].
- Terminating exceptions are masked by a mask setting specified by using the floating-point operation exception enable bit.
- Terminating exceptions are masked by a mask setting defined by a hardware function other than the above.

Note 1. 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_H (RH850G4MH2)).

[Supplement]

This instruction is a supervisor-privileged instruction.

2.2.3.34 HSH

<Data manipulation instructions>

Halfword swap halfword

HSH

Halfword swap of halfword data

[Instruction format]	HSH reg2, reg3
[Operation]	GR[reg3] ← GR[reg2]
[Format]	Format XII
[Opcode]	

15 0	31 16
`rrrrr11111100000`	`wwwww01101000110`

[Flags]	CY	"1" if the lower halfword of the operation result is "0"; otherwise, "0".
	OV	0
	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>

Halfword swap word

HSW

Halfword swap of word data

[Instruction format]	HSW reg2, reg3
[Operation]	GR[reg3] ← GR[reg2] (15:0) || GR[reg2] (31:16)
[Format]	Format XII

[Opcode]

15 0	31 16
`rrrrr11111100000`	`wwwww01101000100`

[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] Executes endian swap.

2.2.3.36 JARL

<Branch instruction>

JARL	Jump and register link Branch and register link

[Instruction format] (1) JARL disp22, reg2

(2) JARL disp32, reg1

(3) JARL [reg1], reg3

[Operation] (1) GR [reg2] ← PC + 4
PC ← PC + sign-extend (disp22)

(2) GR [reg1] ← PC + 6
PC ← PC + disp32

(3) GR[reg3] ← PC + 4
PC ← GR[reg1]

[Format] (1) Format V

(2) Format VI

(3) Format XI

[Opcode]

(1)

15 0	31 16
`rrrrr11110dddddd`	`ddddddddddddddd0`

`ddddddddddddddddddddd` is the higher 21 bits of disp22.
`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0	31 16	47 32
`00000010111RRRRR`	`ddddddddddddddd0`	`DDDDDDDDDDDDDDDD`

`DDDDDDDDDDDDDDDDddddddddddddddd` is the higher 31 bits of disp32.
`RRRRR` ≠ 00000 (Do not specify r0 for reg1.)

(3)

15 0	31 16
`11000111111RRRRR`	`wwwww00101100000`

`wwwww` ≠ 00000 (Do not specify r0 for reg3.)

[Flags]		
	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description]

(1) Saves the current PC value + 4 in general-purpose register reg2, adds the 22-bit displacement data, sign-extended to word length, to PC; stores the value in and transfers the control to PC. Bit 0 of the 22-bit displacement is masked to “0”.

(2) Saves the current PC value + 6 in general-purpose register reg1, adds the 32-bit displacement data to PC and stores the value in and transfers the control to PC. Bit 0 of the 32-bit displacement is masked to “0”.

(3) Stores the current PC value + 4 in reg3, specifies the contents of reg1 for the PC value, and then transfers the control. If reg1 and reg3 are the same, before store the value of current PC +4, specifies the contents of reg1 for the PC.

[Supplement]

The current PC value used for calculation is the address of the first byte of this instruction itself. The jump destination is this instruction with the displacement value = 0.

JARL instruction corresponds to the call function of the subroutine control instruction, and saves the return PC address in the general register which is specified by JARL instruction. JMP instruction corresponds to the return function of the subroutine control instruction, and can be used to specify general-purpose register containing the return address as reg1 to the return PC.

CAUTION

Do not specify r0 for the general-purpose register reg2 in the instruction format (1) JARL disp22, reg2.
Do not specify r0 for the general-purpose register reg1 in the instruction format (2) JARL disp32, reg1.
Do not specify r0 for the general-purpose register reg3 in the instruction format (3) JARL [reg1], reg3.

2.2.3.37 JMP

<Branch instruction>

JMP	Jump register Unconditional branch (register relative)

[Instruction format] (1) JMP [reg1]

(2) JMP disp32[reg1]

[Operation] (1) PC ← GR[reg1]

(2) PC ← GR[reg1] + disp32

[Format] (1) Format I

(2) Format VI

[Opcode]

(1)

15 0
`0000000011RRRRR`

(2)

15 0	31 16	47 32
`00000110111RRRRR`	`ddddddddddddddd0`	`DDDDDDDDDDDDDDDD`

`DDDDDDDDDDDDDDDDddddddddddddddd` is the higher 31 bits of disp32.

[Flags]

CY	—
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.

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 ← PC + sign-extend (disp22)

(2) PC ← PC + disp32

[Format] (1) Format V

(2) Format VI

[Opcode]

(1)

15 0	31 16
`0000011110dddddd`	`ddddddddddddddd0`

`ddddddddddddddddddddd` is the higher 21 bits of disp22.

(2)

15 0	31 16	47 32
`0000001011100000`	`ddddddddddddddd0`	`DDDDDDDDDDDDDDDD`

`DDDDDDDDDDDDDDDDddddddddddddddd` is the higher 31 bits of disp32.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

(1) Adds the 22-bit displacement data, sign-extended to word length, to the current PC and stores the value in and transfers the control to PC. Bit 0 of the 22-bit displacement is masked to “0”.

(2) Adds the 32-bit displacement data to the current PC and stores the value in PC and transfers the control to PC. Bit 0 of the 32-bit displacement is masked to “0”.

[Supplement]

The current PC value used for calculation is the address of the first byte of this instruction itself. The displacement value being “0” signifies that the branch destination is the instruction itself.

2.2.3.39 LD.B

<Load instruction>

LD.B	Load byte Load of (signed) byte data

[Instruction format]	(1)	LD.B disp16[reg1], reg2
	(2)	LD.B disp23[reg1], reg3
	(3)	LD.B [reg1]+, reg3
	(4)	LD.B [reg1]−, reg3

[Operation]

(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException (MDP)
GR[reg2] ← sign-extend (Load-memory (adr, Byte))

(2) adr ← GR[reg1] + sign-extend (disp23)Note 1
CheckException (MDP)
GR[reg3] ← sign-extend (Load-memory (adr, Byte))

(3) adr ← GR [reg1]Note 1
CheckException (MDP)
GR [reg3] ← sign-extend (Load-memory (adr, Byte))
GR [reg1] ← GR [reg1] + 1

(4) adr ← GR [reg1]Note 1
CheckException (MDP)
GR [reg3] ← sign-extend (Load-memory (adr, Byte))
GR [reg1] ← GR [reg1] – 1

Note 1. 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]

(1)

15 0	31 16
rrrrr111000RRRRR	dddddddddddddddd

(2)

15 0	31 16	47 32
00000111100RRRRR	wwwwwddddddd0101	DDDDDDDDDDDDDDDD

Where RRRRR = reg1, wwwww = reg3.
ddddddd is the lower 7 bits of disp23.
DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.

(3)

15 0	31 16
00010111111RRRRR	wwwww01101110000

(4)

15 0	31 16
00100111111RRRRR	wwwww01101110000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

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

(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException(MDP)
GR[reg2] ← zero-extend (Load-memory (adr, Byte))

(2) adr ← GR[reg1] + sign-extend (disp23)Note 1
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Byte))

(3) adr ← GR[reg1]Note 1
CheckException(MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Byte))
GR[reg1] ← GR[reg1] + 1

(4) adr ← GR[reg1]Note 1
CheckException(MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Byte))
GR[reg1] ← GR[reg1] − 1

Note 1. 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]

(1)

15 0	31 16
`rrrrr11110bRRRRR`	`ddddddddddddddd1`

`ddddddddddddddd` is the higher 15 bits of disp16, and b is bit 0 of disp16.
`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

	15 0	31 16	47 32
(2)	`00000111101RRRRR`	`wwwwwddddddd0101`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.
`ddddddd` is the lower 7 bits of disp23.
`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

	15 0	31 16
(3)	`00011111111RRRRR`	`wwwww01101110000`

	15 0	31 16
(4)	`00101111111RRRRR`	`wwwww01101110000`

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

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

(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 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, zero-extends it to word length, and stores the result in the general-purpose 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>

LD.DW

Load Double-word

Load of double-word data

[Instruction format] LD.DW disp23[reg1], reg3

[Operation]

```
adr ← GR[reg1] + sign-extend (disp23)Note 1
CheckException (MAE)
CheckException (MDP)
data ← Load-memory (adr, Double-word)
GR[reg3 + 1] || GR[reg3] ← data
```

Note 1. 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	wwwwwdddddd01001	DDDDDDDDDDDDDDDD

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]

CY	—
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]−, reg3

[Operation]

(1) adr ← GR[reg1] + sign-extend (disp16)[Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg2] ← sign-extend (Load-memory (adr, Halfword))

(2) adr ← GR[reg1] + sign-extend (disp23)[Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← sign-extend (Load-memory (adr, Halfword))

(3) adr ← GR [reg1][Note 1]
CheckException (MAE)
CheckException (MDP)
GR [reg3] ← sign-extend (Load-memory (adr, Halfword))
GR [reg1] ← GR [reg1] + 2

(4) adr ← GR [reg1][Note 1]
CheckException (MAE)
CheckException (MDP)
GR [reg3] ← sign-extend (Load-memory (adr, Halfword))
GR [reg1] ← GR [reg1] – 2

Note 1. 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]

(1)

15 0	31 16
`rrrrr111001RRRRR`	`ddddddddddddddd0`

Where `ddddddddddddddd` is the higher 15 bits of disp16.

(2)

15 0	31 16	47 32
`00000111100RRRRR`	`wwwwwdddddd00111`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.
`dddddd` is the lower side bits 6 to 1 of disp23.
`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

(3)

15 0	31 16
`00010111111RRRRR`	`wwwww01101110100`

(4)

15 0	31 16
`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, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address, sign-extended to word length, and stored in general-purpose register reg2.

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

(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 general-purpose 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]

(1) adr ← GR[reg1] + sign-extend (disp16)[Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg2] ← zero-extend (Load-memory (adr, Halfword))

(2) adr ← GR[reg1] + sign-extend (disp23)[Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Halfword))

(3) adr ← GR[reg1][Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Halfword))
GR[reg1] ← GR[reg1] + 2

(4) adr ← GR[reg1][Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Halfword))
GR[reg1] ← GR[reg1] − 2

Note 1. 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]

(1)

15 0	31 16
rrrrr111111RRRRR	ddddddddddddddd1

Where ddddddddddddddd is the higher 15 bits of disp16.
rrrrr ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0	31 16	47 32
00000111101RRRRR	wwwwwdddddd00111	DDDDDDDDDDDDDDDD

Where RRRRR = reg1, wwwww = reg3.
dddddd is the lower side bits 6 to 1 of disp23.
DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.

(3)

15 0	31 16
00011111111RRRRR	wwwww01101110100

(4)

15 0	31 16
00101111111RRRRR	wwwww01101110100

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

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

(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 general-purpose 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>

LD.W

Load word

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]

```
(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
    CheckException (MAE)
    CheckException (MDP)
    GR[reg2] ← Load-memory (adr, Word)
(2) adr ← GR[reg1] + sign-extend (disp23)Note 1
    CheckException (MAE)
    CheckException (MDP)
    GR[reg3] ← Load-memory (adr, Word)
(3) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    GR[reg3] ← Load-memory (adr, Word)
    GR[reg1] ← GR [reg1] + 4
(4) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    GR[reg3] ← Load-memory (adr, Word)
    GR[reg1] ← GR[reg1] − 4
```

Note 1. 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`	`ddddddddddddddd1`

Where `ddddddddddddddd` is the higher 15 bits of disp16.

	15 0	31 16	47 32
(2)	`00000111100RRRRR`	`wwwwwdddddd01001`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.
`dddddd` is the lower side bits 6 to 1 of disp23.
`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

	15 0	31 16
(3)	`00010111111RRRRR`	`wwwww01101111000`

	15 0	31 16
(4)	`00100111111RRRRR`	`wwwww01101111000`

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

(1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. 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

<Special instruction>

LDL.BU	Load Linked byte unsigned Load to start atomic byte data manipulation

[Instruction format] LDL.BU [reg1], reg3

[Operation]

adr ← GR[reg1]$^{\text{Note 1}}$
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Byte))
LLbit ← 1$^{\text{Note 2}}$

Note 1. An MDP exception may occur depending on the results of the address calculation.

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00001111111RRRRR	wwwww01101110000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Reads byte data from memory for an atomic read-modify-write, zero-extends it to word length, and stores the results in the general-purpose register reg3. Then, generates a link 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

<Special instruction>

Load Linked halfword unsigned

LDL.HU

Load to start atomic halfword data manipulation

[Instruction format] LDL.HU [reg1], reg3

[Operation]

```
adr ← GR[reg1]Note 1
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← zero-extend (Load-memory (adr, Halfword))
LLbit ← 1Note 2
```

Note 1. An MAE or MDP exception may occur depending on the results of the address calculation.

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00001111111RRRRR	wwwww01101110100

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	Reads halfword data from memory for an atomic read-modify-write, zero-extends it to word length, and stores the results in the general-purpose register reg3. Then, generates a 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

<Special instruction>

Load Linked word

LDL.W

Load to start atomic word data manipulation

[Instruction format] LDL.W [reg1], reg3

[Operation]

```
adr ← GR[reg1][Note 1]
CheckException (MAE)
CheckException (MDP)
GR[reg3] ← Load-memory (adr, Word)
LLbit ← 1[Note 2]
```

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

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00000111111RRRRR	wwwww01101111000

[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_H (RH850G4MH2))**

Load MPU entries

[Instruction format] LDM.MP [reg1], eh-et,

[Operation]

```
if (PSW.UM==0)
then
    if ( eh ≤ et )
    then
    cur ← eh
    end ← et
    tmp ← reg1
    while (cur ≤ end) {
        adr ← tmp[Note 1, Note 2]
        CheckException(MDP)
        MPLA[cur] ← Load-memory (adr, Word)
        tmp ← tmp + 4
        adr ← tmp[Note 1, Note 2]
        CheckException(MDP)
        MPUA[cur] ← Load-memory (adr, Word)
        tmp ← tmp + 4
        adr ← tmp[Note 1, Note 2]
        CheckException(MDP)
        MPAT[cur] ← Load-memory (adr, Word)
        tmp ← tmp + 4
        cur ← 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
rrrrr111111RRRRR	wwwww00101100110

rrrrr indicates eh.
wwwww indicates et.
RRRRR indicates reg1.

[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 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>

Load to system register

LDSR

Load to system register

[Instruction format] LDSR reg2, regID, selID
LDSR reg2, regID

[Operation] SR[regID, selID] ← GR[reg2][Note 1]

Note 1. 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
`rrrrr111111RRRRR`	`sssss00000100000`

`rrrrr`: regID, `sssss`: selID, RRRRR: reg2

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	Loads the word data of general-purpose register reg2 to the system register specified by the system register number and 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	Loop

[Instruction format] LOOP reg1, disp16

[Operation]

```
GR[reg1] ← GR[reg1] + (−1)^Note 1
if (GR[reg1] ! = 0)
then
    PC ← PC − zero-extend (disp16)
```

Note 1. −1 (FFFF FFFF$_H$) is added. The carry flag is updated in the same way as when the ADD instruction is executed.

[Format] Format VII

[Opcode]

15 ... 0	31 ... 16
`00000110111RRRRR`	`ddddddddddddddd1`

Where `ddddddddddddddd` is the higher 15 bits of disp16.

[Flags]

CY	"1" if a carry occurs from MSB in the reg1 operation; otherwise, "0".
OV	"1" if an overflow occurs in the reg1 operation; otherwise, "0".
S	"1" if reg1 is negative; otherwise, "0".
Z	"1" if reg1 is 0; otherwise, "0".
SAT	—

[Description]	Updates the general-purpose register reg1 by adding –1 from its contents. If the contents after this update are not 0, the following processing is performed. If the contents are 0, the system continues to the next instruction. • The result of logically shifting the 15-bit immediate data 1 bit to the left and zero-extending it to word length is subtracted from the current PC value, and then the control is transferred. • –1 (FFFF FFFF$_H$) 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] || GR[reg4] ← GR[reg2] × GR[reg1] + GR[reg3+1] || GR[reg3]

[Format] Format XI

[Opcode]

15 … 0	31 … 16
`rrrrr111111RRRRR`	`wwww0011110mmmm0`

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then adds the result (64-bit data) to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose register reg3+1 (for example, this would be “r7” if the reg3 value is r6 and “1” is added) as the higher 32 bits. Of the result (64-bit data), the higher 32 bits are stored in general-purpose register reg4+1 and the lower 32 bits are stored in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are handled as 32-bit signed integers. This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.

CAUTION

General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered 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] || GR [reg4] ← GR[reg2] × GR[reg1] + GR[reg3+1] || GR[reg3]

[Format] Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`wwww0011111mmmm0`

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then adds the result (64-bit data) to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose register reg3+1 (for example, this would be “r7” if the reg3 value is r6 and “1” is added) as the higher 32 bits. Of the result (64-bit data), the higher 32 bits are stored in general-purpose register reg4+1 and the lower 32 bits are stored in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are handled as 32-bit 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 (32-bit)

MOV

Data transfer

[Instruction format]	(1)	MOV reg1, reg2
	(2)	MOV imm5, reg2
	(3)	MOV imm32, reg1
[Operation]	(1)	GR[reg2] ← GR[reg1]
	(2)	GR[reg2] ← sign-extend (imm5)
	(3)	GR[reg1] ← imm32
[Format]	(1)	Format I
	(2)	Format II
	(3)	Format VI

[Opcode]

(1)

15 0
`rrrrr000000RRRRR`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0
`rrrrr010000iiiii`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(3)

15 0	31 16	47 32
`00000110001RRRRR`	`iiiiiiiiiiiiiiii`	`IIIIIIIIIIIIIIII`

`i` (bits 31 to 16) refers to the lower 16 bits of 32-bit immediate data.
`I` (bits 47 to 32) refers to the higher 16 bits of 32-bit immediate data.

[Flags]	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description]

(1) Copies and transfers the word data of general-purpose register reg1 to general-purpose register reg2. General-purpose register reg1 is not affected.

(2) Copies and transfers the 5-bit immediate data, sign-extended to word length, to general- purpose register reg2.

(3) Copies and transfers the 32-bit immediate data to general-purpose register reg1.

CAUTION

Do not specify r0 as reg2 in MOV reg1, reg2 for instruction format (1) or in MOV imm5, reg2 for instruction format (2).

2.2.3.54 MOVEA

<Arithmetic instruction>

MOVEA	Move effective address Effective address transfer

[Instruction format] MOVEA imm16, reg1, reg2

[Operation] GR[reg2] ← GR[reg1] + sign-extend (imm16)

[Format] Format VI

[Opcode]

15 0	31 16
`rrrrr110001RRRRR`	`iiiiiiiiiiiiiiii`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the 16-bit immediate data, sign-extended to word length, to the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. Neither general-purpose register reg1 nor the flags is affected.

[Supplement] This instruction is to execute a 32-bit address calculation with the PSW flag value unchanged.

CAUTION

Do not specify r0 for reg2.

2.2.3.55 MOVHI

<Arithmetic instruction>

Move high halfword

MOVHI

Higher halfword transfer

[Instruction format]　MOVHI imm16, reg1, reg2

[Operation]　GR[reg2] ← GR[reg1] + (imm16 || 0^{16})

[Format]　Format VI

[Opcode]

15　　　　　　　　0	31　　　　　　16
`rrrrr110010RRRRR`	`iiiiiiiiiiiiiiii`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]　Adds the word data with its higher 16 bits specified as the 16-bit immediate data and the lower 16 bits being “0” to the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. Neither general-purpose register reg1 nor the flags is affected.

[Supplement]　This instruction is to generate the higher 16 bits of a 32-bit address.

CAUTION

Do not specify r0 for reg2.

2.2.3.56 MUL

<Multiply instruction>

MUL	Multiply word by register/immediate (9-bit)
	Multiplication of (signed) word data

[Instruction format] (1) MUL reg1, reg2, reg3

(2) MUL imm9, reg2, reg3

[Operation] (1) GR[reg3] || GR[reg2] ← GR[reg2] × GR[reg1]

(2) GR[reg3] || GR[reg2] ← GR[reg2] × sign-extend (imm9)

[Format] (1) Format XI

(2) Format XII

[Opcode]

(1)

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01000100000`

(2)

15 0	31 16
`rrrrr111111iiiii`	`wwwww01001IIII00`

`iiiii` are the lower 5 bits of 9-bit immediate data.
`IIII` are the higher 4 bits of 9-bit immediate data.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	(1) Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then stores the higher 32 bits of the result (64-bit data) in general- purpose register reg3 and the lower 32 bits in general-purpose register reg2. The contents of general-purpose registers reg1 and reg2 are handled as 32-bit signed integers. General-purpose register reg1 is not affected.
	(2) Multiplies the word data in general-purpose register reg2 by 9-bit immediate data, extended to word length, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2.
[Supplement]	When general-purpose register reg2 and general-purpose register reg3 are the same register, only the higher 32 bits of the multiplication result are stored in the register.

2.2.3.57 MULH

<Multiply instruction>

MULH	Multiply halfword by register/immediate (5-bit)
	Multiplication of (signed) halfword data

[Instruction format]	(1)	MULH reg1, reg2
	(2)	MULH imm5, reg2
[Operation]	(1)	GR[reg2] ← GR[reg2] (15:0) × GR[reg1] (15:0)
	(2)	GR[reg2] ← GR[reg2] × sign-extend (imm5)
[Format]	(1)	Format I
	(2)	Format II

[Opcode]

(1)
```
15             0
rrrrr000111RRRRR
```

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)
```
15             0
rrrrr010111iiiii
```

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[Flags]		
	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description]	(1) Multiplies the lower halfword data of general-purpose register reg2 by the halfword data of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected. (2) Multiplies the lower halfword data of general-purpose register reg2 by the 5-bit immediate data, sign-extended to halfword length, and stores the result in general-purpose register reg2.
[Supplement]	In the case of a multiplier or a multiplicand, the higher 16 bits of general-purpose registers reg1 and reg2 are ignored.

CAUTION

Do not specify r0 for reg2.

2.2.3.58 MULHI

<Multiply instruction>

Multiply halfword by immediate (16-bit)

MULHI

Multiplication of (signed) halfword immediate data

[Instruction format]	MULHI imm16, reg1, reg2
[Operation]	GR[reg2] ← GR[reg1] (15:0) × imm16
[Format]	Format VI

[Opcode]

15 0	31 16
`rrrrr110111RRRRR`	`iiiiiiiiiiiiiiii`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[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 reg2.

2.2.3.59 MULU

<Multiply instruction>

Multiply word unsigned by register/immediate (9-bit)

MULU

Multiplication of (unsigned) word data

[Instruction format] (1) MULU reg1, reg2, reg3

(2) MULU imm9, reg2, reg3

[Operation] (1) GR[reg3] || GR[reg2] ← GR[reg2] × GR [reg1]

(2) GR[reg3] | GR[reg2] ← GR[reg2] × zero-extend (imm9)

[Format] (1) Format XI

(2) Format XII

[Opcode]

(1)

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01000100010`

(2)

15 0	31 16
`rrrrr111111iiiii`	`wwwww01001IIII10`

`iiiii` are the lower 5 bits of 9-bit immediate data.
`IIII` are the higher 4 bits of 9-bit immediate data.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	(1) Multiplies the word data in general-purpose register reg2 by the word data in general-purpose register reg1, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2. General-purpose register reg1 is not affected. (2) Multiplies the word data in general-purpose register reg2 by 9-bit immediate data, zero-extended to word length, then stores the higher 32 bits of the result (64-bit data) in general-purpose register reg3 and the lower 32 bits in general-purpose register reg2.
[Supplement]	When general-purpose register reg2 and general-purpose register reg3 are the same register, only the higher 32 bits of the multiplication result are stored in the register.

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
0000000000000000
```

[Flags]	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description]	Performs no processing and executes the next instruction.
[Supplement]	The opcode is the same as that of MOV r0, r0.

2.2.3.61 NOT

<Logical instruction>

NOT

NOT

Logical negation (1's complement)

[Instruction format]	NOT reg1, reg2
[Operation]	GR[reg2] ← NOT (GR[reg1])
[Format]	Format I
[Opcode]	

```
15              0
rrrrr000001RRRRR
```

[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] Logically negates the word data of general-purpose register reg1 using 1's complement and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.62 NOT1

<Bit manipulation instruction>

NOT1

NOT bit

NOT bit

[Instruction format]	(1) NOT1 bit#3, disp16[reg1]
	(2) NOT1 reg2, [reg1]

[Operation]

(1)
```
adr ← GR [reg1] + sign-extend (disp16)^Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, bit#3))
token ← not-bit (token, bit#3)
Store-memory (adr, token, Byte)
```

(2)
```
adr ← GR[reg1]^Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← not-bit (token, reg2)
Store-memory (adr, token, Byte)
```

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

[Format]	(1) Format VIII
	(2) Format IX

[Opcode]

(1)

15 0	31 16
`01bbb111110RRRRR`	`dddddddddddddddd`

(2)

15 0	31 16
`rrrrr111111RRRRR`	`0000000011100010`

[Flags]

CY	—
OV	—
S	—
Z	“1” if bit specified by operand = “0”, “0” if bit specified by operand = “1”.
SAT	—

[Description]	(1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, then the bits indicated by the 3-bit bit number are inverted (0 → 1, 1 → 0) and the data is written back to the original address. If the specified bit of the read byte data is “0”, the Z flag is set to “1”, and if the specified bit is “1”, the Z flag is cleared to “0”.
	(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, then the bits specified by lower 3 bits of general- purpose register reg2 are inverted (0 → 1, 1 → 0) and the data is written back to the original address. If the specified bit of the read byte data is “0”, the Z flag is set to “1”, and if the specified bit is “1”, the Z flag is cleared to “0”.
[Supplement]	The Z flag of PSW indicates the status of the specified bit (0 or 1) before this instruction is executed and does not indicate the content of the specified bit resulting from the instruction execution.

CAUTION

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.

2.2.3.63 OR

<Logical instruction>

OR	OR OR

[Instruction format]	OR reg1, reg2
[Operation]	GR[reg2] ← GR[reg2] OR GR[reg1]
[Format]	Format I

[Opcode]

```
15            0
rrrrr001000RRRRR
```

[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 reg2 with the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.64 ORI

<Logical instruction>

ORI

OR immediate (16-bit)

OR immediate

[Instruction format]	ORI imm16, reg1, reg2
[Operation]	GR[reg2] ← GR[reg1] OR zero-extend (imm16)
[Format]	Format VI

[Opcode]

15 0	31 16
`rrrrr110100RRRRR`	`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 immediate data, zero-extended to word length, and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.65 POPSP

<Special instruction>

POPSP	Pop registers from stack Pop registers from stack

[Instruction format] POPSP rh-rt

[Operation]

```
if rh ≤ rt
then cur ← rt
     end ← rh
     tmp ← sp
     while (cur ≥ end) {
          adr ← tmp[Note 1, Note 2]
          CheckException(MDP)
          GR[cur] ← Load-memory (adr, Word)
          cur ← cur − 1
          tmp ← tmp + 4
     }
     sp ← tmp
```

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

Note 2. The lower 2 bits of adr are masked to 0.

[Format] Format XI

[Opcode]

15 0	31 16
01100111111RRRRR	wwwww00101100000

RRRRR indicates rh.

wwwww indicates rt.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Loads general-purpose register rt to rh from the stack in descending order (rt, rt – 1, rt – 2, …, rh). After all the registers down to the specified register have been loaded, sp is updated (incremented).

[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>

PREPARE

Save registers to stack

Create stack frame

[Instruction format]

(1) PREPARE list12, imm5

(2) PREPARE list12, imm5, sp/imm[Note 1]

Note 1. The sp/imm values are specified by bits 19 and 20 of the sub-opcode.

[Operation]

```
(1) tmp ← sp
    foreach (all regs in list12) {
        tmp ← tmp − 4
        adr ← tmp[Note 1, Note 2]
        CheckException (MDP)
        Store-memory (adr, GR[reg in list12], Word)
    }
    sp ← tmp − zero-extend (imm5 logically shift left by 2)

(2) tmp ← sp
    foreach (all regs in list12) {
        tmp ← tmp − 4
        adr ← tmp[Note 1, Note 2]
        CheckException (MDP)
        Store-memory (adr, GR[reg in list12], Word)
    }
    sp ← tmp − zero-extend (imm5 logically shift left by 2)

    case
      ff = 00: ep ← sp
      ff = 01: ep ← sign-extend (imm16)
      ff = 10: ep ← imm16 logically shift left by 16
      ff = 11: ep ← imm32
```

Note 1. An MDP exception might occur depending on the result of address calculation.
Note 2. The lower 2 bits of adr are masked to 0.

[Format] Format XIII

[Opcode]

(1)

15 0	31 16
`0000011110iiiiiL`	`LLLLLLLLLLL00001`

(2)

15 0	31 16	Option (47-32 or 63-32)
`0000011110iiiiiL`	`LLLLLLLLLLLff011`	`imm16/imm32`

In the case of 32-bit immediate data (imm32), bits 47 to 32 are the lower 16 bits of imm32 and bits 63 to 48 are the higher 16 bits of imm32.

`ff` = 00: sp is loaded to ep
`ff` = 01: Sign-extended 16-bit immediate data (bits 47 to 32) is loaded to ep
`ff` = 10: 16-bit logical left-shifted 16-bit immediate data (bits 47 to 32) is loaded to ep
`ff` = 11: 32-bit immediate data (bits 63 to 32) is loaded to ep

The values of `LLLLLLLLLLLL` 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_H
- When all of the register’s non-corresponding bits are “1”: 081F $FFFF_H$

[Flags]

CY	—
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]

list12 general-purpose registers are saved in ascending order (r20, r21, ..., r31).

imm5 is used to create a stack frame that is used for auto variables and temporary data.

The lower two bits of the address specified by sp are masked to 0 and aligned to the word boundary.

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 registers to stack Push registers to stack

[Instruction format] PUSHSP rh-rt

[Operation]

```
if rh ≤ rt
then cur ← rh
     end ← rt
     tmp ← sp
     while (cur ≤ end) {
          tmp ← tmp − 4
          adr ← tmp[Note 1, Note 2]
          CheckException (MDP)
          Store-memory (adr, GR[cur], Word)
          cur ← cur + 1
     }
     sp ← tmp
```

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

Note 2. The lower 2 bits of adr are masked to 0.

[Format] Format XI

[Opcode]

15 0	31 16
01000111111RRRRR	wwwww00101100000

RRRRR indicates rh.

wwwww indicates rt.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	Stores general-purpose register rh to rt in the stack in ascending order (rh, rh + 1, rh + 2, …, rt). After all the specified registers have been stored, sp is updated (decremented).
[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>

Restore contexts from register bank

RESBANK

Restore from register bank

[Instruction format] RESBANK

[Operation]

```
if (RBNR.BN > 0)
then
    if (RBCR0.MD == 0)
    then
            adr ← RBIP – RBNR.BN × 60h
            CheckException (MDP)
            GR[30] ← Load-memory (adr, Word)
            cur ← 19
            adr ← adr + 4
    else
            adr ← RBIP – RBNR.BN × 90h
            cur ← 31
            adr ← adr + 4
    while (cur > 0) {
            CheckException(MDP)
            GR[cur] ← Load-memory (adr, Word)
            cur    ← cur – 1
            adr    ← adr + 4
    }
    CheckException (MDP)
    FPSR ← Load-memory (adr, Word)
    adr ← adr + 4
    CheckException (MDP)
    EIIC ← Load-memory (adr, Word)
    adr ← adr + 4
    CheckException (MDP)
    EIPSW ← Load-memory (adr, Word)
    adr ← adr + 4
    CheckException (MDP)
    EIPC ← Load-memory (adr, Word)
    RBNR.BN ← RBNR.BN – 1
    PSW.ID ← 1
else
    FEPC ← PC (return PC)
    FEPSW ← PSW
    FEIC ← 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_H)

[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>

Reserved instruction exception

RIE

Reserved instruction exception

[Instruction format] (1) RIE

(2) RIE imm5, imm4

[Operation]

FEPC ← PC (return PC)
FEPSW ← PSW
FEIC ← exception cause code (0000 0060_H)
PSW.UM ← 0
PSW.NP ← 1
PSW.EP ← 1
PSW.ID ← 1
PC ← exception handler address (offset address 60_H)

[Format] (1) Format I

(2) Format X

[Opcode]

(1)

15 … 0
`0000000001000000`

(2)

15 … 0	31 … 16
`iiiii1111111IIII`	`0000000000000000`

Where `iiiii` = imm5, `IIII` = imm4.

[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_H. For details, see the hardware manual of the product used.

2.2.3.70 ROTL

<Data manipulation instruction>

ROTL

Rotate Left

Rotate

[Instruction format]

(1) ROTL imm5, reg2, reg3

(2) ROTL reg1, reg2, reg3

[Operation]

(1) GR[reg3] ← GR[reg2] rotate left by zero-extend (imm5)

(2) GR[reg3] ← GR[reg2] rotate left by GR[reg1]

[Format]

Format VII

[Opcode]

	15 0	31 16
(1)	`rrrrr111111iiiii`	`wwwww00011000100`
(2)	`rrrrr111111RRRRR`	`wwwww00011000110`

[Flags]

CY	“1” if operation result bit 0 is “1”; otherwise “0”, including if the rotate amount is “0”.
OV	0
S	“1” if the operation result is negative; otherwise, “0”.
Z	“1” if the operation result is “0”; otherwise, “0”.
SAT	—

[Description]

(1) Rotates the word data of general-purpose register reg2 to the left by the specified shift amount, which is indicated by a 5-bit immediate value zero-extended to word length. The result is written to general-purpose register reg3. General-purpose register reg2 is not affected.

(2) Rotates the word data of general-purpose register reg2 to the left by the specified shift amount indicated by the lower 5 bits of general-purpose register reg1. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.

2.2.3.71 SAR

<Data manipulation instruction>

SAR

Shift arithmetic right by register/immediate (5-bit)

Arithmetic right shift

[Instruction format]
(1) SAR reg1, reg2
(2) SAR imm5, reg2
(3) SAR reg1, reg2, reg3

[Operation]
(1) GR[reg2] ← GR[reg2] arithmetically shift right by GR[reg1]
(2) GR[reg2] ← GR[reg2] arithmetically shift right by zero-extend (imm5)
(3) GR[reg3] ← GR[reg2] arithmetically shift right by GR[reg1]

[Format]
(1) Format IX
(2) Format II
(3) Format XI

[Opcode]

```
     15              0 31             16
(1)  rrrrr111111RRRRR 0000000010100000

     15              0
(2)  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>

Shift and set flag condition

SASF

Shift and flag condition setting

[Instruction format] SASF cccc, reg2

[Operation] if conditions are satisfied

```
    then GR[reg2] ← (GR[reg2] Logically shift left by 1) OR 0000 0001_H
    else GR[reg2] ← (GR[reg2] Logically shift left by 1) OR 0000 0000_H
```

[Format] Format IX

[Opcode]

15 0	31 16
rrrrr1111110cccc	0000001000000000

[Flags]

CY	—
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 Code	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	H	(CY or Z) = 0

Condition Code	Name	Condition Formula
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	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

[Supplement]

See the SETF instruction.

2.2.3.73 SATADD

<Saturated operation instructions>

SATADD	Saturated add register/immediate (5-bit) Saturated addition

[Instruction format] (1) SATADD reg1, reg2
(2) SATADD imm5, reg2
(3) SATADD reg1, reg2, reg3

[Operation] (1) GR[reg2] ← saturated (GR[reg2] + GR[reg1])
(2) GR[reg2] ← saturated (GR[reg2] + sign-extend (imm5))
(3) GR[reg3] ← saturated (GR[reg2] + GR[reg1])

[Format] (1) Format I
(2) Format II
(3) Format XI

[Opcode]

(1)

15 0
`rrrrr000110RRRRR`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0
`rrrrr010001iiiii`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(3)

15 0	31 16
`rrrrr111111RRRRR`	`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”.
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 $FFFF_H$, 7FFF $FFFF_H$ is stored in reg2, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg2; then the SAT flag is set (1). General-purpose register reg1 is not affected.

(2) Adds the 5-bit immediate data, sign-extended to the word length, to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, when the result exceeds the maximum positive value 7FFF $FFFF_H$, 7FFF $FFFF_H$ is stored in reg2, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg2; then the SAT flag is set (1).

(3) Adds the word data of general-purpose register reg1 to the word data of general-purpose register reg2, and stores the result in general-purpose register reg3. However, when the result exceeds the maximum positive value 7FFF $FFFF_H$, 7FFF $FFFF_H$ is stored in reg3, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.

[Supplement]

The SAT flag is a cumulative flag. The saturate result sets the flag to “1” and will not be cleared to “0” even if the result of the subsequent operation is not saturated.

The saturated operation instruction is executed normally, even with the SAT flag set to “1”.

CAUTIONS

1. Use LDSR instruction and load data to the PSW to clear the SAT flag to “0”.
2. Do not specify r0 as reg2 in instruction format (1) SATADD reg1, reg2 and in instruction format (2) SATADD imm5, reg2.

2.2.3.74 SATSUB

<Saturated operation instructions>

Saturated subtract

SATSUB

Saturated subtraction

[Instruction format]	(1)	SATSUB reg1, reg2
	(2)	SATSUB reg1, reg2, reg3
[Operation]	(1)	GR[reg2] ← saturated (GR[reg2] − GR[reg1])
	(2)	GR[reg3] ← saturated (GR[reg2] − GR[reg1])
[Format]	(1)	Format I
	(2)	Format XI

[Opcode]

(1)

15 0
`rrrrr000101RRRRR`

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

(2)

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01110011010`

[Flags]	CY	“1” if a borrow occurs from MSB; otherwise, “0”.
	OV	“1” if overflow occurs; otherwise, “0”.
	S	“1” if saturated operation result is negative; otherwise, “0”.
	Z	“1” if saturated operation result is “0”; otherwise, “0”.
	SAT	“1” if OV = 1; otherwise, does not change.

[Description]

(1) Subtracts the word data of general-purpose register reg1 from the word data of general- purpose register reg2 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF $FFFF_H$, 7FFF $FFFF_H$ is stored in reg2; if the result exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg2. The SAT flag is set to “1”. General-purpose register reg1 is not affected.

(2) Subtracts the word data of general-purpose register reg1 from the word data of general- purpose register reg2, and stores the result in general-purpose register reg3. However, when the result exceeds the maximum positive value 7FFF $FFFF_H$, 7FFF $FFFF_H$ is stored in reg3, and when it exceeds the maximum negative value 8000 0000_H, 8000 0000_H is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.

[Supplement]

The SAT flag is a cumulative flag. The saturate result sets the flag to “1” and will not be cleared to “0” even if the result of the subsequent operation is not saturated.

The saturated operation instruction is executed normally, even with the SAT flag set to “1”.

CAUTIONS

1. Use LDSR instruction and load data to the PSW to clear the SAT flag to “0”.
2. Do not specify r0 as reg2 in instruction format (1) SATSUB reg1, reg2.

2.2.3.75 SATSUBI

<Saturated operation instructions>

Saturated subtract immediate

SATSUBI

Saturated subtraction

[Instruction format]	SATSUBI imm16, reg1, reg2
[Operation]	GR[reg2] ← saturated (GR[reg1] − sign-extend (imm16))
[Format]	Format VI

[Opcode]

15 0	31 16
rrrrr110011RRRRR	iiiiiiiiiiiiiiii

rrrrr ≠ 00000 (Do not specify r0 for reg2.)

[Flags]

CY	“1” if a borrow occurs from MSB; otherwise, “0”.
OV	“1” if overflow occurs; otherwise, “0”.
S	“1” if saturated operation result is negative; otherwise, “0”.
Z	“1” if saturated operation result is “0”; otherwise, “0”.
SAT	“1” if OV = 1; otherwise, does not change.

[Description] Subtracts the 16-bit immediate data, sign-extended to word length, from the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF FFFF$_H$, 7FFF FFFF$_H$ is stored in reg2; if the result exceeds the maximum negative value 8000 0000$_H$, 8000 0000$_H$ is stored in reg2. The SAT flag is set to “1”. General-purpose register reg1 is not affected.

[Supplement] 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>

SATSUBR	Saturated subtract reverse Saturated reverse subtraction

[Instruction format]	SATSUBR reg1, reg2
[Operation]	GR[reg2] ← saturated (GR[reg1] − GR[reg2])
[Format]	Format I

[Opcode]

```
15             0
rrrrr000100RRRRR
```

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[Flags]		
	CY	"1" if a borrow occurs from MSB; otherwise, "0".
	OV	"1" if overflow occurs; otherwise, "0".
	S	"1" if saturated operation result is negative; otherwise, "0".
	Z	"1" if saturated operation result is "0"; otherwise, "0".
	SAT	"1" if OV = 1; otherwise, does not change.

[Description] Subtracts the word data of general-purpose register reg2 from the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. If the result exceeds the maximum positive value 7FFF FFFF$_H$, 7FFF FFFF$_H$ is stored in reg2; if the result exceeds the maximum negative value 8000 0000$_H$, 8000 0000$_H$ is stored in reg2. The SAT flag is set to "1". General-purpose register reg1 is not affected.

[Supplement] The SAT flag is a cumulative flag. The saturation result sets the flag to "1" and will not be cleared to "0" even if the result of the subsequent operation is not saturated.

The saturated operation instruction is executed normally, even with the SAT flag set to "1".

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] ← GR[reg2] − GR[reg1] − 1
else GR[reg3] ← GR[reg2] − GR[reg1] − 0
```

[Format] Format XI

[Opcode]

15 0	31 16
`rrrrr111111RRRRR`	`wwwww011100cccc0`

[Flags]

CY	“1” if a borrow occurs from MSB; otherwise, “0”.
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
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	H	(CY or Z) = 0

Condition Code	Name	Condition Formula
0100	S/N	S = 1
1100	NS/P	S = 0
0101	T	Always (Unconditional)
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
(1101)	Setting prohibited	

2.2.3.78 SCH0L

<Bit search instructions>

Search zero from left

SCH0L

Bit (0) search from MSB side

[Instruction format] SCH0L reg2, reg3

[Operation] GR[reg3] ← search zero from left of GR[reg2]

[Format] Format IX

[Opcode]

15 0	31 16
`rrrrr11111100000`	`wwwww01101100100`

[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] Searches word data of general-purpose register reg2 from the left side (MSB side), and writes the number of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to general- purpose register reg3 (e.g., when bit 31 of reg2 is 0, 01_H is written to reg3).

When bit (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the bit (0) found is the LSB, the CY flag is set (1).

2.2.3.79 SCH0R

<Bit search instructions>

SCH0R	Search zero from right Bit (0) search from LSB side

[Instruction format] SCH0R reg2, reg3

[Operation] GR[reg3] ← search zero from right of GR[reg2]

[Format] Format IX

[Opcode]

15 0	31 16
rrrrr11111100000	wwwww01101100000

[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] Searches word data of general-purpose register reg2 from the right side (LSB side), and writes the number of 1s before the bit position (0 to 31) at which 0 is first found plus 1 to general- purpose register reg3 (e.g., when bit 0 of reg2 is 0, 01_H is written to reg3).

When bit (0) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the bit (0) found is the MSB, the CY flag is set (1).

2.2.3.80 SCH1L

<Bit search instructions>

SCH1L	Search one from left Bit (1) search from MSB side

[Instruction format] SCH1L reg2, reg3

[Operation] GR[reg3] ← search one from left of GR[reg2]

[Format] Format IX

[Opcode]

15 0	31 16
`rrrrr11111100000`	`wwwww01101100110`

[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] Searches word data of general-purpose register reg2 from the left side (MSB side), and writes the number of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to general- purpose register reg3 (e.g., when bit 31 of reg2 is 1, 01_H is written to reg3).

When bit (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the bit (1) found is the LSB, the CY flag is set (1).

2.2.3.81 SCH1R

<Bit search instructions>

SCH1R	Search one from right Bit (1) search from LSB side

[Instruction format] SCH1R reg2, reg3

[Operation] GR[reg3] ← search one from right of GR [reg2]

[Format] Format IX

[Opcode]

15 0	31 16
rrrrr11111100000	wwwww01101100010

[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] Searches word data of general-purpose register reg2 from the right side (LSB side), and writes the number of 0s before the bit position (0 to 31) at which 1 is first found plus 1 to general- purpose register reg3 (e.g., when bit 0 of reg2 is 1, 01_H is written to reg3).

When bit (1) is not found, 0 is written to reg3, and the Z flag is simultaneously set (1). If the bit (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]

```
(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
    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)

(2) adr ← GR[reg1]Note 1
    CheckException (MDP)
    token ← Load-memory (adr, Byte)
    Z flag ← Not (extract-bit (token, reg2))
    token ← set-bit (token, reg2)
    Store-memory (adr, token, Byte)
```

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

[Format]

(1) Format VIII

(2) Format IX

[Opcode]

(1)

15 0	31 16
00bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100000

[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, the bits indicated by the 3-bit bit number are set (1) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0". (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the lower 3 bits indicated of general-purpose register reg2 are set (1) and the data is written back to the original address. If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
[Supplement]	The Z flag of PSW indicates the initial status of the specified bit (0 or 1) and does not indicate the content of the specified bit resulting from the instruction execution.

CAUTION

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] ← 0000 0001_H

else GR[reg2] ← 0000 0000_H

[Format] Format IX

[Opcode]

15 0	31 16
rrrrr1111110cccc	0000000000000000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

When the condition specified by condition code “cccc” is met, stores “1” to general-purpose 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	T	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	H	(CY or Z) = 0	1111	GT	((S xor OV) or Z) = 0

[Supplement]

Examples of SETF instruction:

(1) Translation of multiple condition clauses
If A of statement *if (A)* in C language consists of two or greater condition clauses (a_1, a_2, a_3, and so on), it is usually translated to a sequence of *if (a1) then, if (a2) then*. The object code executes “conditional branch” by checking the result of evaluation equivalent to a_n. Because a pipeline operation requires more time to execute “condition judgment” + “branch” than to execute an ordinary operation, the result of evaluating each condition clause *if (an)* is stored in register Ra. By performing a logical operation to Ra_n after all the condition clauses have been evaluated, the pipeline delay can be prevented.

(2) Double-length operation
To execute a double-length operation, such as “Add with Carry”, the result of the CY flag can be stored in general-purpose register reg2. Therefore, a carry from the lower bits can be represented as a numeric value.

2.2.3.84 SHL

<Data manipulation instruction>

SHL	Shift logical left by register/immediate (5-bit) Logical left shift

[Instruction format]	(1)	SHL reg1, reg2
	(2)	SHL imm5, reg2
	(3)	SHL reg1, reg2, reg3
[Operation]	(1)	GR[reg2] ← GR[reg2] logically shift left by GR[reg1]
	(2)	GR[reg2] ← GR[reg2] logically shift left by zero-extend (imm5)
	(3)	GR[reg3] ← GR[reg2] logically shift left by GR[reg1]
[Format]	(1)	Format IX
	(2)	Format II
	(3)	Format XI

[Opcode]

(1)

15 0	31 16
rrrrr111111RRRRR	0000000011000000

(2)

15 0
rrrrr010110iiiii

(3)

15 0	31 16
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), the position specified by the lower 5 bits of general-purpose register reg1, by shifting “0” to LSB. The result is written to general-purpose register reg2. General-purpose register reg1 is not affected.

(2) Logically left-shifts the word data of general-purpose register reg2 by “n” (0 to +31), the 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>

Shift logical right by register/immediate (5-bit)

SHR

Logical right shift

[Instruction format]	(1)	SHR reg1, reg2
	(2)	SHR imm5, reg2
	(3)	SHR reg1, reg2, reg3
[Operation]	(1)	GR[reg2] ← GR[reg2] logically shift right by GR[reg1]
	(2)	GR[reg2] ← GR[reg2] logically shift right by zero-extend (imm5)
	(3)	GR[reg3] ← GR[reg2] logically shift right by GR[reg1]
[Format]	(1)	Format IX
	(2)	Format II
	(3)	Format XI

[Opcode]

(1)

15 … 0	31 … 16
`rrrrr111111RRRRR`	`0000000010000000`

(2)

15 … 0
`rrrrr010100iiiii`

(3)

15 … 0	31 … 16
`rrrrr111111RRRRR`	`wwwww00010000010`

[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>

Short format load byte

SLD.B

Load of (signed) byte data

[Instruction format]	SLD.B disp7[ep], reg2

[Operation]	adr ← ep + zero-extend (disp7)Note 1
	CheckException(MDP)
	GR[reg2] ← sign-extend (Load-memory (adr, Byte))
	Note 1. An MDP exception might occur depending on the result of address calculation.

[Format]	Format IV

[Opcode]

```
15              0
rrrrr0110ddddddd
```

[Flags]	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description] Adds the 7-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and stored in reg2.

2.2.3.87 SLD.BU

<Load instruction>

SLD.BU	Short format load byte unsigned Load of (unsigned) byte data

[Instruction format]	SLD.BU disp4[ep], reg2

[Operation]	adr ← ep + zero-extend (disp4)Note 1
	CheckException (MDP)
	GR[reg2] ← zero-extend (Load-memory (adr, Byte))

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

[Format] Format IV

[Opcode]

```
15              0
rrrrr0000110dddd
```

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the 4-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored in reg2.

CAUTION

Do not specify r0 for reg2.

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 ← ep + zero-extend (disp8)[Note 1]

CheckException (MAE)

CheckException (MDP)

GR[reg2] ← sign-extend (Load-memory (adr, Halfword))

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

[Format] Format IV

[Opcode]

```
15             0
rrrrr1000ddddddd
```

`ddddddd` is the higher 7 bits of disp8.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address. 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>

SLD.HU	Short format load halfword unsigned Load of (unsigned) halfword data

[Instruction format]	SLD.HU disp5[ep], reg2

[Operation]

adr ← ep + zero-extend (disp5)Note 1

CheckException (MAE)

CheckException (MDP)

GR [reg2] ← zero-extend (Load-memory (adr, Halfword))

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

[Format] Format IV

[Opcode]

```
15                 0
rrrrr0000111dddd
```

`rrrrr` ≠ 00000 (Do not specify r0 for reg2.)
`dddd` is the higher 4 bits of disp5.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the element pointer to the 5-bit displacement 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 Load of word data

[Instruction format] SLD.W disp8 [ep] , reg2

[Operation]

```
adr ← ep + zero-extend (disp8)Note 1
CheckException (MAE)
CheckException (MDP)
GR[reg2] ← Load-memory (adr, Word)
```

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

[Format] Format IV

[Opcode]

```
15              0
rrrrr1010dddddd0
```

`dddddd` is the higher 6 bits of disp8.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address. Word data is read from this 32-bit address, and stored in general-purpose register reg2.

CAUTION

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

2.2.3.91 SNOOZE

<Special instruction>

Snooze

SNOOZE

Snooze

[Instruction format]	Snooze
[Operation]	Snooze while hardware-defined period
[Format]	Format X

[Opcode]

15 … 0	31 … 16
0000111111100000	0000000100100000

[Flags]		
	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

[Description]

Temporarily 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[Note 1].
- Terminating exceptions are masked by a mask setting specified by using the floating-point operation exception enable bit.
- Terminating exceptions are masked by a mask setting defined by a hardware function other than the above.

Note 1. 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 ← ep + zero-extend (disp7)[Note 1] CheckException (MDP) Store-memory (adr, GR[reg2], Byte)

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

[Format]	Format IV

[Opcode]

```
15               0
rrrrr0111ddddddd
```

[Flags]

CY	—
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>

SST.H	Short format store halfword Storage of halfword data

[Instruction format]	SST.H reg2, disp8[ep]
[Operation]	adr ← ep + zero-extend (disp8)[Note 1] CheckException (MAE) CheckException (MDP) Store-memory (adr, GR[reg2], Halfword)

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

[Format] Format IV

[Opcode]

```
15                0
rrrrr1001ddddddd
```

`ddddddd` is the higher 7 bits of disp8.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address, and stores the 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>

SST.W	Short format store word Storage of word data

[Instruction format] SST.W reg2, disp8[ep]

[Operation] adr ← ep + zero-extend (disp8)[Note 1]

CheckException (MAE)

CheckException (MDP)

Store-memory (adr, GR[reg2], Word)

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

[Format] Format IV

[Opcode]

```
15               0
rrrrr1010dddddd1
```

`dddddd` is the higher 6 bits of disp8.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Adds the element pointer to the 8-bit displacement data, zero-extended to word length, to generate a 32-bit address and stores the word data of reg2 to the generated 32-bit address.

CAUTION

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

2.2.3.95 ST.B

<Store instruction>

ST.B

Store byte

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]

(1) adr ← GR [reg1] + sign-extend (disp16)^Note 1^
CheckException (MDP)
Store-memory (adr, GR[reg2], Byte)

(2) adr ← GR[reg1] + sign-extend (disp23)^Note 1^
CheckException (MDP)
Store-memory (adr, GR[reg3], Byte)

(3) adr ← GR[reg1]^Note 1^
CheckException (MDP)
Store-memory (adr, GR[reg3], Byte)
GR[reg1] ← GR[reg1] + 1

(4) adr ← GR[reg1]^Note 1^
CheckException (MDP)
Store-memory (adr, GR[reg3], Byte)
GR[reg1] ← GR[reg1] − 1

Note 1. 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]

(1)

15 0	31 16
rrrrr111010RRRRR	dddddddddddddddd

(2)

15 0	31 16	47 32
00000111100RRRRR	wwwwwddddddd1101	DDDDDDDDDDDDDDDD

Where RRRRR = reg1, wwwww = reg3.
ddddddd is the lower 7 bits of disp23.
DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.

(3)

15 0	31 16
00010111111RRRRR	wwwww01101110010

(4)

15 0	31 16
00100111111RRRRR	wwwww01101110010

[Flags]

CY	—
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 lowest byte data of general-purpose register reg2 to the generated address.

(2) Adds the data of general-purpose register reg1 to the 23-bit displacement data, sign-extended to word length, to generate a 32-bit address and stores the lowest byte data of general-purpose register reg3 to the generated address.

(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 Storage of double-word data

[Instruction format] ST.DW reg3, disp23[reg1]

[Operation]

adr ← GR[reg1] + sign-extend (disp23)Note 1

CheckException (MAE)

CheckException (MDP)

data ← GR[reg3+1] || GR[reg3]

Store-memory (adr, data, Double-word)

Note 1. 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	wwwwwdddddd01111	DDDDDDDDDDDDDDDD

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]

CY	—
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>

ST.H	Store halfword 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]

```
(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg2], Halfword)
(2) adr ← GR[reg1] + sign-extend (disp23)Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Halfword)
(3) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Halfword)
    GR[reg1] ← GR[reg1] + 2
(4) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Halfword)
    GR[reg1] ← GR[reg1] − 2
```

Note 1. 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]

(1)

	15 0	31 16
(1)	`rrrrr111011RRRRR`	`ddddddddddddddd0`

Where `ddddddddddddddd` is the higher 15 bits of disp16.

	15 0	31 16	47 32
(2)	`00000111101RRRRR`	`wwwwwdddddd01101`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.
`dddddd` is the lower side bits 6 to 1 of disp23.
`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

	15 0	31 16
(3)	`00010111111RRRRR`	`wwwww01101110110`

	15 0	31 16
(4)	`00100111111RRRRR`	`wwwww01101110110`

[Flags]

CY	—
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.

(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. Adds 2 to the contents of the general-purpose register reg1 and stores the result in the general-purpose register reg1.

(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.

CAUTION

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

2.2.3.98 ST.W

<Store instruction>

ST.W	Store word 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]

```
(1) adr ← GR[reg1] + sign-extend (disp16)Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg2], Word)
(2) adr ← GR[reg1] + sign-extend (disp23)Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Word)
(3) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Word)
    GR[reg1] ← GR[reg1] + 4
(4) adr ← GR[reg1]Note 1
    CheckException (MAE)
    CheckException (MDP)
    Store-memory (adr, GR[reg3], Word)
    GR[reg1] ← GR[reg1] − 4
```

Note 1. 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]

(1)

15 0	31 16
`rrrrr111011RRRRR`	`ddddddddddddddd1`

Where `ddddddddddddddd` is the higher 15 bits of disp16.

(2)

15 0	31 16	47 32
`00000111100RRRRR`	`wwwwwdddddd01111`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.
`dddddd` is the lower side bits 6 to 1 of disp23.
`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

(3)

15 0	31 16
`00010111111RRRRR`	`wwwww01101111010`

(4)

15 0	31 16
`00100111111RRRRR`	`wwwww01101111010`

[Flags]

CY	—
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 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

<Special instruction>

Store conditional byte

STC.B

Conditional storage when atomic byte data manipulation is complete

[Instruction format] STC.B reg3, [reg1]

[Operation]

```
adr ← GR[reg1]Note 1
CheckException (MDP)
data ← GR[reg3]
token ← LLbitNote 2

if (token == 1)
then Store-memory (adr, data, Byte)
       GR[reg3] ← 1
else GR[reg3] ← 0

LLbit ← 0Note 2
```

Note 1. An MDP exception may occur depending on the results of the address calculation.

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00000111111RRRRR	wwwww01101110010

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] This instruction can only be executed successfully if a link exists that corresponds to the specified address. If a corresponding link exists, the byte data of general-purpose register reg3 is stored in the memory and an atomic read-modify-write is executed.
If the corresponding link has been lost, the data is not stored in the memory and execution of this instruction fails.

Whether execution of the STC.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

<Special instruction>

Store conditional halfword

STC.H

Conditional storage when atomic halfword data manipulation is complete

[Instruction format] STC.H reg3, [reg1]

[Operation]

```
adr ← GR[reg1]Note 1
CheckException (MAE)
CheckException (MDP)
data ← GR[reg3]
token ← LLbitNote 2

if (token == 1)
then Store-memory (adr, data, Halfword)
     GR[reg3] ← 1
else GR[reg3] ← 0

LLbit ← 0Note 2
```

Note 1. An MAE or MDP exception may occur depending on the results of the address calculation.

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00000111111RRRRR	wwwww01101110110

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

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

Whether execution of the STC.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 ← GR[reg1]Note 1
CheckException (MAE)
CheckException (MDP)
data ← GR[reg3]
token ← LLbitNote 2

if (token == 1)
then Store-memory (adr, data, Word)
        GR[reg3] ← 1
else GR[reg3] ← 0

LLbit ← 0Note 2
```

Note 1. An MAE or MDP exception may occur depending on the results of the address calculation.

Note 2. For the link operation, see the hardware manual of the product used.

[Format] Format VII

[Opcode]

15 0	31 16
00000111111RRRRR	wwwww01101111010

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

This instruction can only be executed successfully if a link exists that corresponds to the specified address. If a corresponding link exists, the word data of general-purpose register reg3 is stored in the memory and an atomic read-modify-write is executed.
If the corresponding link has been lost, the data is not stored in the memory and execution of this instruction fails.

Whether execution of the STC.W instruction has succeeded or not can be ascertained by checking the contents of general-purpose register reg3 after the instruction has been executed. If execution of the STC.W instruction was successful, general-purpose register reg3 will be set (1). If execution failed, reg3 will be cleared (0).

This instruction can be used together with the LDL.W instruction to ensure accurate updating of the memory in a multi-core system. 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.MP **(Supported only when Architecture Identifier bit PID[31:24] = 07_H (RH850G4MH2))**

Store MPU entries

[Instruction format] STM.MP eh-et, [reg1]

[Operation]

```
if (PSW.UM==0)
then
    if ( eh ≤ et )
    then
        cur ← eh
        end ← et
        tmp ← reg1
        while (cur ≤ end) {
            adr ← tmp[Note 1, Note 2]
            CheckException(MDP)
            Store-memory (adr, MPLA[cur], Word)
            tmp ← tmp + 4
            adr ← tmp[Note 1, Note 2]
            CheckException(MDP)
            Store-memory (adr, MPUA[cur], Word)
            tmp ← tmp + 4
            adr ← tmp[Note 1, Note 2]
            CheckException(MDP)
            Store-memory (adr, MPAT[cur], Word)
            tmp ← tmp + 4
            cur ← 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
`rrrrr111111RRRRR`	`wwwww00101100100`

`rrrrr` indicates eh.
`wwwww` indicates et.
`RRRRR` indicates reg1.

[Flags]

CY	—
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>

STSR	Store contents of system register Storage of contents of system register

[Instruction format] STSR regID, reg2, selID
STSR regID, reg2

[Operation] GR[reg2] ← SR[regID, selID][Note 1]

Note 1. 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
`rrrrr111111RRRRR`	`sssss00001000000`

`rrrrr`: reg2, `sssss`: selID, RRRRR: regID

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Stores the system register contents specified by the system register number and 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 reg1, reg2
[Operation]	GR[reg2] ← GR[reg2] − GR[reg1]
[Format]	Format I

[Opcode]

```
15             0
rrrrr001101RRRRR
```

[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 reg1 from the word data of general-purpose register reg2 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.105 SUBR

<Arithmetic instruction>

SUBR	Subtract reverse Reverse subtraction

[Instruction format]	SUBR reg1, reg2
[Operation]	GR[reg2] ← GR[reg1] − GR[reg2]
[Format]	Format I

[Opcode]

```
15              0
rrrrr001100RRRRR
```

[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 of general-purpose register reg1 and stores the result in general-purpose register reg2. General-purpose register reg1 is not affected.

2.2.3.106 SWITCH

<Special instruction>

SWITCH	Jump with table look up Jump with table look up

[Instruction format] SWITCH reg1

[Operation] adr ← (PC + 2) + (GR[reg1] logically shift left by 1)[Note 1]

CheckException (MDP)

PC ← (PC + 2) + (sign-extend (Load-memory (adr, Halfword))) logically shift left by 1

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

[Format] Format I

[Opcode]

```
15              0
00000000010RRRRR
```

RRRRR ≠ 00000 (Do not specify r0 for reg1.)

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]	The following steps are taken.

<1> Adds the start address (the one subsequent to the SWITCH instruction) to general- purpose register reg1, logically left-shifted by 1, to generate a 32-bit table entry address.

<2> Loads the halfword entry data indicated by the address generated in step <1>.

<3> Adds the table start address after sign-extending the loaded halfword data and logically left-shifting it by 1 (the one subsequent to the SWITCH instruction) to generate a 32-bit target address.

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

CAUTIONS

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] ← sign-extend (GR[reg1] (7:0))
[Format]	Format I

[Opcode]

15	0
00000000101RRRRR	

[Flags]	CY	—
	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] ← sign-extend (GR[reg1] (15:0))

[Format] Format I

[Opcode]

```
15              0
00000000111RRRRR
```

[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 exceptions Exception synchronization instruction

[Instruction format]	SYNCE
[Operation]	No operation is performed.
[Format]	Format I
[Opcode]	

15 0
0000000000011101

[Flags]	CY	—
	OV	—
	S	—
	Z	—
	SAT	—

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

2.2.3.110 SYNCI

<Special instruction>

SYNCI	Synchronize instruction fetch Instruction fetch synchronization instruction

[Instruction format] SYNCI

[Operation] Performs instruction fetch synchronization processing.

[Format] Format I

[Opcode]

```
15              0
0000000000011100
```

[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

<Special instruction>

SYNCM	Synchronize memory Memory synchronize instruction

[Instruction format] SYNCM

[Operation] Performs memory access synchronization processing.

[Format] Format I

[Opcode]

```
15              0
0000000000011110
```

[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

<Special instruction>

SYNCP	Synchronize pipeline Pipeline synchronize instruction

[Instruction format]	SYNCP
[Operation]	Performs pipeline synchronization processing.
[Format]	Format I

[Opcode]

15 0
0000000000011111

[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. 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 execution.

[Supplement] For details on synchronization processing, see the hardware manual of the product used.

2.2.3.113 SYSCALL

<Special instruction>

SYSCALL	System call System call exception

[Instruction format] SYSCALL vector8

[Operation]

```
tmp ← PSW
PSW.UM ← 0
PSW.EP ← 1
PSW.ID ← 1
if (vector8 <= SCCFG.SIZE) is satisfied
    then adr ← SCBP + zero-extend (vector8 logically shift left by 2)Note 3
    else adr ← SCBPNote 3
CheckException (MDP)Note 2
EIPC ← PC + 4 (return PC)
EIPSW ← tmp
EIIC ←exception cause codeNote 1
PC ← SCBP + Load-memory (adr, Word)
```

Note 1. See the hardware manual of the product used.
Note 2. When an exception occurs, the PSW before execution stored in tmp is saved.
Note 3. An MDP exception might occur depending on the result of address calculation.

[Format] Format X

[Opcode]

15 0	31 16
11010111111vvvvv	00VVV00101100000

Where VVV is the higher 3 bits of vector8 and vvvvv is the lower 5 bits of vector8.

[Flags]

CY	—
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>.
<3>	Saves the contents of the return PC (address of the instruction next to the 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_H.
<5>	Updates the PSW according to the exception causes listed in the hardware manual of the product used.
<6>	Loads the word of the address generated in <1>.
<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

<Special instruction>

TRAP	Trap Software exception

[Instruction format] TRAP vector5

[Operation]

EIPC ← PC + 4 (return PC)

EIPSW ← PSW

EIIC ← exception cause code[Note 1]

PSW.UM ← 0

PSW.EP ← 1

PSW.ID ← 1

PC ← exception handler address[Note 1]

Note 1. See the hardware manual of the product used.

[Format] Format X

[Opcode]

15 … 0	31 … 16
`00000111111vvvvv`	`0000000100000000`

vvvvv = vector5

[Flags]

CY	—
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_H	$0000\ 0040_H$	40_H
01_H	$0000\ 0041_H$	
…		
$0F_H$	$0000\ 004F_H$	
10_H	$0000\ 0050_H$	50_H
11_H	$0000\ 0051_H$	
…		
$1F_H$	$0000\ 005F_H$	

2.2.3.115 TST

<Logical instruction>

TST

Test

Test

[Instruction format]	TST reg1, reg2
[Operation]	result ← GR[reg2] AND GR[reg1]
[Format]	Format I

[Opcode]

```
15                0
rrrrr001011RRRRR
```

[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] ANDs the word data of general-purpose register reg2 with the word data of general-purpose register reg1. The result is not stored with only the flags being changed. General-purpose registers reg1 and reg2 are not affected.

2.2.3.116 TST1

<Bit manipulation instruction>

TST1

Test bit

Bit test

[Instruction format]

(1) TST1 bit#3, disp16[reg1]

(2) TST1 reg2, [reg1]

[Operation]

(1)
```
adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, bit#3))
```

(2)
```
adr ← GR[reg1]Note 1
CheckException (MDP)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
```

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

[Format]

(1) Format VIII

(2) Format IX

[Opcode]

(1)

15 0	31 16
`11bbb111110RRRRR`	`dddddddddddddddd`

(2)

15 0	31 16
`rrrrr111111RRRRR`	`0000000011100110`

[Flags]

CY	—
OV	—
S	—
Z	“1” if bit specified by operand = “0”, “0” if bit specified by operand = “1”.
SAT	—

[Description]

(1) Adds the word data of general-purpose register reg1 to the16-bit displacement data, sign- extended to word length, to generate a 32-bit address; checks the bit specified by the 3-bit number at the byte data location referenced by the generated address. If the specified bit is “0”, “1” is set to the Z flag of PSW and if the bit is “1”, the Z flag is cleared to “0”. The byte data, including the specified bit, is not affected.

(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address; checks the bit specified by the lower 3 bits of reg2 at the byte data location referenced by the generated address. If the specified bit is “0”, “1” is set to the Z flag of PSW and if the bit is “1”, the Z flag is cleared to “0”. The byte data, including the specified bit, is not affected.

2.2.3.117 XOR

<Logical instruction>

XOR	Exclusive OR Exclusive OR

[Instruction format]	XOR reg1, reg2
[Operation]	GR[reg2] ← GR[reg2] XOR GR[reg1]
[Format]	Format I

[Opcode]

```
15                 0
rrrrr001001RRRRR
```

[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] Exclusively ORs the word data of general-purpose register reg2 with the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. General- 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 imm16, reg1, reg2

[Operation] GR[reg2] ← GR[reg1] XOR zero-extend (imm16)

[Format] Format VI

[Opcode]

15 0	31 16
`rrrrr110101RRRRR`	`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] Exclusively ORs the word data of general-purpose register reg1 with the 16-bit immediate data, zero-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] ← zero-extend (GR[reg1] (7:0))
[Format]	Format I

[Opcode]

```
15                 0
00000000100RRRRR
```

[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] ← zero-extend (GR[reg1] (15:0))
[Format]	Format I

[Opcode]

```
15             0
00000000110RRRRR
```

[Flags]	CY	—
	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`

`ppPPPPP` indicates cacheop.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Sets the word data of general-purpose register reg1 as a 32-bit address or the cache index and manipulates the cache specified by cacheop. For details about the cache index specification method, see 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.

Table 2.7 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
0000000	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

[Instruction format] PREF prefop, [reg1]

[Operation] Executes the prefetch operation specified by prefop.

[Format] Format X

[Opcode]

15 … 0	31 … 16
`11011111111RRRRR`	`PPPPP00101100000`

PPPPP indicates prefop.

[Flags]

CY	—
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. **Table 2.8** 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

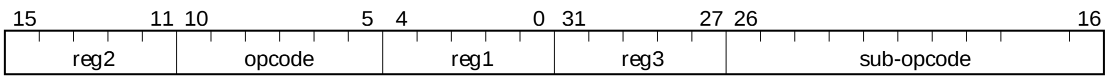
2.4.1 Instruction Formats

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) → Lower address side
- Higher part of instruction format (including bit 15 or bit 31) → 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.

Table 2.9 List of Conditions for Comparison Instructions

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 2.10 Definitions of Condition Code Bits and Their Logical Inversions

Mnemonic (True)	Condition Code fcond		Bit Definition of Condition Code fcond(3:0)				Inverted Logic (False)
			Less than	Equal to	Unordered	Invalid operation exception occurs when unordered	
	Decimal	Binary	fcond(2)	fcond(1)	fcond(0)	fcond(3)	
F	0	0b0000	F	F	F	No	(T)
UN	1	0b0001	F	F	T	No	(OR)
EQ	2	0b0010	F	T	F	No	(NEQ)
UEQ	3	0b0011	F	T	T	No	(OLG)
OLT	4	0b0100	T	F	F	No	(UGE)
ULT	5	0b0101	T	F	T	No	(OGE)
OLE	6	0b0110	T	T	F	No	(UGT)
ULE	7	0b0111	T	T	T	No	(OGT)
SF	8	0b1000	F	F	F	Yes	(ST)
NGLE	9	0b1001	F	F	T	Yes	(GLE)
SEQ	10	0b1010	F	T	F	Yes	(SNE)
NGL	11	0b1011	F	T	T	Yes	(GL)
LT	12	0b1100	T	F	F	Yes	(NLT)
NGE	13	0b1101	T	F	T	Yes	(GE)
LE	14	0b1110	T	T	F	Yes	(NLE)
NGT	15	0b1111	T	T	T	Yes	(GT)

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 [*a*]	Value stored in general-purpose register *a*
SR [*a*, *b*]	Value stored in system register (RegID = *a*, SelID = *b*)
result	Result is reflected in flag
==	Comparison (true upon a match)
+	Add
−	Subtract
||	Bit concatenation
×	Multiply
÷	Divide
abs	Absolute value
ceil	Rounding in +∞ direction
compare	Comparison
cvt	Converts type according to rounding mode
floor	Rounding in –∞ direction
max	Maximum value
min	Minimum value
neg	Sign inversion
round	Rounding to closest value
sqrt	Square root
trunc	Rounding in zero direction
fma (*a*, *b*, *c*)	Result of multiplying *a* and *b* and then adding *c*
fms (*a*, *b*, *c*)	Result of multiplying *a* and *b* and then subtracting *c*

Table 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
I	Single bit data of immediate data (indicates higher bit of immediate data)
i	Single bit data of immediate data
fff	3-bit data that specifies the bit number (fcbit) of the condition bit that stores the result of a floating-point comparison instruction
FFFF	4-bit data corresponding to the mnemonic or condition code (fcond) of the comparison condition of a comparison instruction

2.4.4.1 ABSF.D

<Floating-point instruction>

Floating-point Absolute Value (Double)

ABSF.D

Floating-point absolute value (double precision)

[Instruction format] ABSF.D reg2, reg3

[Operation] reg3 ← 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 1 0 0	0
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

<Floating-point instruction>

Floating-point Absolute Value (Single)

ABSF.S

Floating-point absolute value (single precision)

[Instruction format] ABSF.S reg2, reg3

[Operation] reg3 ← 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

<Floating-point instruction>

Floating-point Add (Double)

ADDF.D

Floating-point add (double precision)

[Instruction format] ADDF.D reg1, reg2, reg3

[Operation] reg3 ← 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 0 0 0	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction adds the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 with 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.4.4.4 ADDF.S

<Floating-point instruction>

Floating-point Add (Single)

ADDF.S

Floating-point add (single precision)

[Instruction format] ADDF.S reg1, reg2, reg3

[Operation] reg3 ← 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]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.4.4.5 CEILF.DL

<Floating-point instruction>

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 ← ceil reg2 (double → 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 0	1 1 1 1 1 1	0 0 0 1 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} – 1$ 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.6 CEILF.DUL

<Floating-point instruction>

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

CEILF.DUL

Conversion to unsigned fixed-point format (double precision)

[Instruction format] CEILF.DUL reg2, reg3

[Operation] reg3 ← ceil reg2 (double → 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 0	1 1 1 1 1 1	1 0 0 1 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} – 1$ 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

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

[Operation result]

reg2 (A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.7 CEILF.DUW

<Floating-point instruction>

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

CEILF.DUW

Conversion to unsigned fixed-point format (double precision)

[Instruction format] CEILF.DUW reg2, reg3

[Operation] reg3 ← ceil reg2 (double → 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 0	1 1 1 1 1 1	1 0 0 1 0	w w w w w	1	0 0 0	1 0	1 0 0 0	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by 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 +∞ 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

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

[Operation result]

reg2 (A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.8 CEILF.DW

<Floating-point instruction>

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

CEILF.DW

Conversion to fixed-point format (double precision)

[Instruction format] CEILF.DW reg2, reg3

[Operation] reg3 ← ceil reg2 (double → 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 0	1 1 1 1 1 1	0 0 0 1 0	w w w w w	1	0 0 0	1 0	1 0 0 0	0
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 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{31} – 1$ 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or -∞: $–2^{31}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.9 CEILF.SL

<Floating-point instruction>

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

CEILF.SL

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
r r r r r	1 1 1 1 1 1	0 0 0 1 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, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} – 1$ 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.10 CEILF.SUL

<Floating-point instruction>

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

CEILF.SUL

Conversion to unsigned fixed-point format (single precision)

[Instruction format] CEILF.SUL reg2, reg3

[Operation] reg3 ← ceil reg2 (single → 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 0 0 1 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 specified by general-purpose register reg2 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} – 1$ 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.11 CEILF.SUW

<Floating-point instruction>

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

CEILF.SUW

Conversion to unsigned fixed-point format (single precision)

[Instruction format] CEILF.SUW reg2, reg3

[Operation] reg3 ← ceil reg2 (single → 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 1 0	w w w w w	1	0 0 0	1 0	0 0 0 0	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the single-precision floating-point format contents specified by 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 +∞ 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.12 CEILF.SW

<Floating-point instruction>

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

CEILF.SW

Conversion to fixed-point format (single precision)

[Instruction format]	CEILF.SW reg2, reg3
[Operation]	reg3 ← ceil reg2 (single → 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 0	w w w w w	1	0 0 0	1 0	0 0 0 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 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{31} – 1$ 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.13 CMOVF.D

<Floating-point instruction>

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 ← reg1
else
    reg3 ← 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 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 instruction>

Floating-point Conditional Move (Single)

CMOVF.S

Conditional move (single precision)

[Instruction format] CMOVF.S fcbit, reg1, reg2, reg3

[Operation]

```
if (FPSR.CCn == 1) then
    reg3 ← reg1
else
    reg3 ← 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 R R R	w w w w w	1	0 0 0	0 0	0 f f f	0
reg2		reg1	reg3*1		category	type	sub-op	

Remark: fcbit: `fff`

Note 1. reg3: `wwwww != 0`

`wwwww` ≠ `00000` (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 reg1 is stored in reg3. When these bits are false (0), the reg2 data is stored in 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.

2.4.4.15 CMPF.D

<Floating-point instruction>

Compare floating-point values(Double)

CMPF.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 ← reg2 < reg1
    result.equal ← reg2 == reg1
    result.unordered ← 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	

Remark: fcond: FFFF
fcbit: fff

[Description] This instruction compares the double-precision floating-point format contents of the register pair specified by general-purpose register reg2 with the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, based on the condition "fcond", and sets the result (1 if true, 0 if false) to the condition bits (the CC(7:0) bits: bits 31 to 24) in the FPSR register specified by fcbit in the opcode. If fcbit is omitted, the result is set to the CC0 bit (bit 24).

For description of the comparison condition "fcond" code, see **Table 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.

Table 2.14 Comparison Conditions

Comparison Conditions				Detection of Invalid Operation Exception by Unordered
	fcond	Definition	Description	
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

Note: ?: Unordered (invalid comparison)

```
# When explicitly testing Q-NaN
    CMPF.D    OLT,r12,r14,0    # Check if r12 < r14
    CMPF.D    UN,r12,r14,1     # Check if Unordered
    TRFSR     0
    BT        L2               # If true, go to L2
    TRFSR     1
    BT        ERROR            # If true, go to error processing
# Enter code for processing when neither Unordered nor r12 < r14
L2:
# Enter code for processing when r12 < r14
      :
# When using a comparison to detect Q-NaN
    CMPF.D    LT,r12,r14,0     # Check if r12 < r14
    TRFSR     0
    BT        L2               # If true, go to L2
# Enter code for processing when not r12 < r14
L2:
# Enter code for processing when r12 < r14
      :
```

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

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

[Condition code (fcond) = 8 to 15]

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

Note: [ ] indicates an exception that must occur.

2.4.4.16 CMPF.S

<Floating-point instruction>

Compare floating-point values (Single)

CMPF.S

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 ← reg2 < reg1
    result.equal ← reg2 == reg1
    result.unordered ← 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	

Remark: fcond: `FFFF`
fcbit: `fff`

[Description] This instruction compares the single-precision floating-point format contents of general-purpose register reg2 with the single-precision floating-point format contents of general-purpose 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.

Table 2.15 Comparison Conditions

Comparison Conditions				Detection of Invalid Operation Exception by Unordered
	fcond	Definition	Description	
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

Note: ?: Unordered (invalid comparison)

```
# When explicitly testing Q-NaN
      CMPF.S      OLT,r12,r14,0     # Check if r12 < r14
      CMPF.S      UN,r12,r14,1      # Check if Unordered
      TRFSR       0
      BT          L2                # If true, go to L2
      TRFSR       1
      BT          ERROR             # If true, go to error processing
# Enter code for processing when neither Unordered nor r12 < r14
L2:
# Enter code for processing when r12 < r14
        :
# When using a comparison to detect Q-NaN
      CMPF.S      LT,r12,r14,0      # Check if r12 < r14
      TRFSR       0
      BT          L2                # If true, go to L2
# Enter code for processing when not r12 < r14
L2:
# Enter code for processing when r12 < r14
        :
```

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

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

[Condition code (fcond) = 8 to 15]

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

Note: [ ] indicates an exception that must occur.

2.4.4.17 CVTF.DL

<Floating-point instruction>

CVTF.DL

Floating-point Convert Double to Long (Double)

Conversion to fixed-point format (double precision)

[Instruction format] CVTF.DL reg2, reg3

[Operation] reg3 ← cvt reg2 (double → 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 0	1 1 1 1 1 1	0 0 1 0 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or -∞: $–2^{63}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.18 CVTF.DS

<Floating-point instruction>

Floating-point Convert Double to Single(Double)

CVTF.DS

Conversion to floating-point format (double precision)

[Instruction format] CVTF.DS reg2, reg3

[Operation] reg3 ← cvt reg2 (double → 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 1 1	w w w w w	1	0 0 0	1 0	1 0 0 1	0
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 exceptions]

Unimplemented operation exception (E)

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 (Single)		+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.

2.4.4.19 CVTF.DUL

<Floating-point instruction>

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

CVTF.DUL

Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.DUL reg2, reg3

[Operation] reg3 ← cvt reg2 (double → 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 0	1 1 1 1 1 1	1 0 1 0 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 unsigned 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, 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 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 (Integer)		Max U-Int[V]	0[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.

2.4.4.20 CVTF.DUW

<Floating-point instruction>

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

CVTF.DUW

Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.DUW reg2, reg3

[Operation] reg3 ← cvt reg2(double → 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 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
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 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

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

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.21 CVTF.DW

<Floating-point instruction>

Floating-point Convert Double to Word (Double)

CVTF.DW

Conversion to fixed-point format (double precision)

[Instruction format]	CVTF.DW reg2, reg3
[Operation]	reg3 ← cvt reg2 (double → 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 0	1 1 1 1 1 1	0 0 1 0 0	w w w w w	1	0 0 0	1 0	1 0 0 0	0
reg2			reg3		category	type	sub-op	

[Description]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by 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.

[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 (Integer)		+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>

Floating-point Convert Half to Single (Single)

CVTF.HS

Conversion to floating-point format (single precision)

[Instruction format] CVTF.HS reg2, reg3

[Operation] reg3 ← cvt reg2 (half → 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 1	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 (Single)		+0	−0	+∞	−∞	Q-NaN	Q-NaN[V]

Note: [ ] indicates an exception that must occur.

2.4.4.23 CVTF.LD

<Floating-point instruction>

Floating-point Convert Long to Double (Double)

CVTF.LD

Conversion to floating-point format (double precision)

[Instruction format] CVTF.LD reg2, reg3

[Operation] reg3 ← cvt reg2 (long-word → 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 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 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	0 (Integer)
Operation result [exception]	A (Normal)		+0

2.4.4.24 CVTF.LS

<Floating-point instruction>

Floating-point Convert Long to Single (Single)

CVTF.LS

Conversion to floating-point format (single precision)

[Instruction format] CVTF.LS reg2, reg3

[Operation] reg3 ← cvt reg2 (long-word → 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 w	1	0 0 0	1 0	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 (Normal)		+0

2.4.4.25 CVTF.SD

<Floating-point instruction>

Floating-point Convert Single to Double (Double)

CVTF.SD

Conversion to floating-point format (double precision)

[Instruction format] CVTF.SD reg2, reg3

[Operation] reg3 ← cvt reg2 (single → 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 r	1 1 1 1 1 1	0 0 0 1 0	w w w w 0	1	0 0 0	1 0	1 0 0 1	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 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] 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 (Double)		+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.

2.4.4.26 CVTF.SL

<Floating-point instruction>

Floating-point Convert Single to Long (Single)

CVTF.SL

Conversion to fixed-point format (single precision)

[Instruction format] CVTF.SL reg2, reg3

[Operation] reg3 ← cvt reg2 (single → 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	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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.27 CVTF.SH

<Floating-point instruction>

Floating-point Convert Single to Half (Single)

CVTF.SH

Conversion to half-precision floating-point format (single precision)

[Instruction format] CVTF.SH reg2, reg3

[Operation] reg3 ← zero-extend (cvt reg2 (single → half))

[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 1	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 result 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 exceptions]

Unimplemented operation exception (E)

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 (Half)		+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.

2.4.4.28 CVTF.SUL

<Floating-point instruction>

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

CVTF.SUL

Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.SUL reg2, reg3

[Operation] reg3 ← cvt reg2 (single → 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 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 unsigned 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, 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 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.29 CVTF.SUW

<Floating-point instruction>

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

CVTF.SUW

Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.SUW reg2, reg3

[Operation] reg3 ← cvt reg2 (single → 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 1 0 0	w w w w w	1	0 0 0	1 0	0 0 0 0	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to unsigned 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.30 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 ← cvt reg2 (single → 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 1 0 0	w w w w w	1	0 0 0	1 0	0 0 0 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 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.31 CVTF.ULD

<Floating-point instruction>

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

CVTF.ULD

Conversion to floating-point format (double precision)

[Instruction format] CVTF.ULD reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned long-word → 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	1 0 0 0 1	w w w w 0	1	0 0 0	1 0	1 0 0 1	0
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	0 (Integer)
Operation result [exception]	A (Normal)		+0

2.4.4.32 CVTF.ULS

<Floating-point instruction>

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

CVTF.ULS

Conversion to floating-point format (single precision)

[Instruction format] CVTF.ULS reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned long-word → 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	1 0 0 0 1	w w w w w	1	0 0 0	1 0	0 0 0 1	0
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 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 (Normal)		+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 ← cvt reg2 (unsigned word → 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 r	1 1 1 1 1 1	1 0 0 0 0	w w w w 0	1	0 0 0	1 0	1 0 0 1	0
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

[Operation result]

reg2(A)	+Integer	−Integer	0 (Integer)
Operation result [exception]	A (Normal)		+0

2.4.4.34 CVTF.UWS

<Floating-point instruction>

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

CVTF.UWS

Conversion to floating-point format (single precision)

[Instruction format] CVTF.UWS reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned word → 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	1 0 0 0 0	w w w w w	1	0 0 0	1 0	0 0 0 1	0
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 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 (Normal)		+0

2.4.4.35 CVTF.WD

<Floating-point instruction>

Floating-point Convert Word to Double (Double)

CVTF.WD

Conversion to floating-point format (double precision)

[Instruction format] CVTF.WD reg2, reg3

[Operation] reg3 ← cvt reg2 (word → 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 r	1 1 1 1 1 1	0 0 0 0 0	w w w w 0	1	0 0 0	1 0	1 0 0 1	0
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

[Operation result]

reg2(A)	+Integer	−Integer	0 (Integer)
Operation result [exception]	A (Normal)		+0

2.4.4.36 CVTF.WS

<Floating-point instruction>

Floating-point Convert Word to Single (single)

CVTF.WS

Conversion to floating-point format (single precision)

[Instruction format] CVTF.WS reg2, reg3

[Operation] reg3 ← cvt reg2 (word → 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 0 0	w w w w w	1	0 0 0	1 0	0 0 0 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the 32-bit fixed-point format contents of general-purpose register reg2 to single-precision floating-point format, and stores the result in general-purpose register reg3. The result is rounded in accordance with the current rounding mode.

[Floating-point operation exceptions] Inexact exception (I)

[Operation result]

reg2(A)	+Integer	−Integer	0 (Integer)
Operation result [exception]	A (Normal)		+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 ← 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 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 exceptions]

Unimplemented operation exception (E)

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	B ÷ A				+∞	−∞		
−Normal					−∞	+∞		
+0	±∞[Z]		Q-NaN[V]		+∞	−∞		
−0					−∞	+∞		
+∞	+0	−0	+0	−0	Q-NaN[V]			
−∞	−0	+0	−0	+0				
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 FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

2.4.4.38 DIVF.S

<Floating-point instruction>

Floating-point Divide (Single)

DIVF.S

Floating-point division (single precision)

[Instruction format] DIVF.S reg1, reg2, reg3

[Operation] reg3 ← 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 1 1 1	0
reg2		reg1	reg3		category	type	sub-op	

[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)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B÷A				+∞	−∞		
−Normal					−∞	+∞		
+0	±∞[Z]		Q-NaN[V]		+∞	−∞		
−0					−∞	+∞		
+∞	+0	−0	+0	−0	Q-NaN[V]			
−∞	−0	+0	−0	+0				
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 FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in the hardware manual of the product used.

2.4.4.39 FLOORF.DL

<Floating-point instruction>

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

FLOORF.DL

Conversion to fixed-point format (double precision)

[Instruction format] FLOORF.DL reg2, reg3

[Operation] reg3 ← floor reg2 (double → 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 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
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, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the –∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} – 1$ 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or -∞: $–2^{63}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.40 FLOORF.DUL

<Floating-point instruction>

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

FLOORF.DUL

Conversion to unsigned fixed-point format (double precision)

[Instruction format] FLOORF.DUL reg2, reg3

[Operation] reg3 ← floor reg2 (double → 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 0	1 1 1 1 1 1	1 0 0 1 1	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the –∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} – 1$ 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.41 FLOORF.DUW

<Floating-point instruction>

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

FLOORF.DUW

Conversion to unsigned fixed-point format (double precision)

[Instruction format] FLOORF.DUW reg2, reg3

[Operation] reg3 ← floor reg2 (double → 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 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			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 unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg3.

The result is rounded in the –∞ direction, regardless of the current rounding mode.

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

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

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.42 FLOORF.DW

<Floating-point instruction>

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

FLOORF.DW

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]

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 1 1	w w w w w	1	0 0 0	1 0	1 0 0 0	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by 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.

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 (Integer)		0 (Integer)		+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.

2.4.4.43 FLOORF.SL

<Floating-point instruction>

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

FLOORF.SL

Conversion to fixed-point format (single precision)

[Instruction format] FLOORF.SL reg2, reg3

[Operation] reg3 ← floor reg2 (single → 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	0 0 0 1 1	w w w w 0	1	0 0 0	1 0	0 0 1 0	0
reg2			reg3		category	type	sub-op	

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

The result is rounded in the –∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} – 1$ 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		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.

2.4.4.44 FLOORF.SUL

<Floating-point instruction>

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

FLOORF.SUL

Conversion to unsigned fixed-point format (single precision)

[Instruction format] FLOORF.SUL reg2, reg3

[Operation] reg3 ← floor reg2 (single → 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 0 0 1 1	w w w w 0	1	0 0 0	1 0	0 0 1 0	0
reg2			reg3		category	type	sub-op	

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

The result is rounded in the –∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} – 1$ 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.45 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 ← floor reg2 (single → 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 1 1	w w w w w	1	0 0 0	1 0	0 0 0 0	0
reg2			reg3		category	type	sub-op	

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

The result is rounded in the –∞ direction, regardless of the current rounding mode.

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

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.46 FLOORF.SW

<Floating-point instruction>

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

FLOORF.SW

Conversion to fixed-point format (single precision)

[Instruction format] FLOORF.SW reg2, reg3

[Operation] reg3 ← floor reg2 (single → 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
reg2			reg3		category	type	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 –∞ 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		+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.

2.4.4.47 FMAF.S

<Floating-point instruction>

FMAF.S

Floating-point Fused-Multiply-add (Single)

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

[Instruction format] FMAF.S reg1, reg2, reg3

[Operation] reg3 ← 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 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 exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg3(C)	reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
±0	+Normal	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
+∞	+Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
−∞	+Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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 and MULF instructions.

2.4.4.48 FMSF.S

<Floating-point instruction>

Floating-point Fused-Multiply-subtract (Single)

FMSF.S

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

[Instruction format] FMSF.S reg1, reg2, reg3

[Operation] reg3 ← 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 0 1	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, subtracts 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 subtract operation 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]

reg3(C)	reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
±0	+Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
+∞	+Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	+Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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>

Floating-point Fused-Negate-Multiply-add (Single)

FNMAF.S

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

[Instruction format] FNMAF.S reg1, reg2, reg3

[Operation] reg3 ← neg (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 w	1	0 0 1	1 1	0 0 1 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, 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 add operation is rounded, in accordance with the current rounding mode. The signs are reversed after rounding.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg3(C)	reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
±0	+Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
−∞	+Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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>

Floating-point Fused-Negate-Multiply-subtract (Single)

FNMSF.S

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

[Instruction format] FNMSF.S reg1, reg2, reg3

[Operation] reg3 ← 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
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, 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 operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg3(C)	reg2(B) reg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
±0	+Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	+Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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>

Floating-point Maximum (Double)

MAXF.D

Floating-point maximum value (double precision)

[Instruction format]	MAXF.D reg1, reg2, reg3
[Operation]	reg3 ← max (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 0	0
reg2		reg1	reg3		category	type	sub-op	

[Description]

This instruction extracts the maximum 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, it is undefined whether +0 or –0 is stored in reg3.

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

Note: [ ] indicates an exception that must occur.

2.4.4.52 MAXF.S

<Floating-point instruction>

Floating-point Maximum (Single)

MAXF.S

Floating-point maximum value (single precision)

[Instruction format] MAXF.S reg1, reg2, reg3

[Operation] reg3 ← max (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 1 0 0	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction extracts the maximum value from the single-precision floating-point format data in general-purpose registers reg1 and reg2, and stores it in 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, it is undefined whether +0 or –0 is stored in reg3.

A subnormal input will not be flushed even if the FS bit of the FPSR register is 1.

[Operation result]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
−Normal	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
+0	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
−0	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
+∞	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
−∞	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	MAX (B, A)	reg1(A)	Q-NaN[V]
Q-NaN	reg2(A)	reg2(A)	reg2(A)	reg2(A)	reg2(A)	reg2(A)	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]

Note: [ ] indicates an exception that must occur.

2.4.4.53 MINF.D

<Floating-point instruction>

MINF.D

Floating-point Minimum (Double)

Floating-point minimum value (double precision)

[Instruction format] MINF.D reg1, reg2, reg3

[Operation] reg3 ← 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	MIN (B, A)						reg1(A)	
−Normal								
+0								
−0								
+∞								
−∞								
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-point minimum value (single precision)

[Instruction format] MINF.S reg1, reg2, reg3

[Operation] reg3 ← 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 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 0 1	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction extracts the minimum value from the single-precision floating-point format data in general-purpose registers reg1 and reg2, and stores it in 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]

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

Note: [ ] indicates an exception that must occur.

2.4.4.55 MULF.D

<Floating-point instruction>

Floating-point Multiply (Double)

MULF.D

Floating-point multiplication (double precision)

[Instruction format] MULF.D reg1, reg2, reg3

[Operation] reg3 ← 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 0 1 0	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction multiplies the double-precision floating-point format contents in the register pair specified by general-purpose register reg2 by the double-precision floating-point format contents in the register pair specified by general-purpose register reg1 and stores the results in the register pair specified by general-purpose register reg3.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B × A	B × A	B × A	B × A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B × A	B × A	B × A	B × A	−∞	+∞	Q-NaN	Q-NaN[V]
+0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.4.4.56 MULF.S

<Floating-point instruction>

Floating-point Multiply (Single)

MULF.S

Floating-point multiplication (single precision)

[Instruction format] MULF.S reg1, reg2, reg3

[Operation] reg3 ← 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 1 0	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction multiplies 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.

[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	B × A	B × A	B × A	B × A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B × A	B × A	B × A	B × A	−∞	+∞	Q-NaN	Q-NaN[V]
+0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

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 ← neg reg2

[Format] Format F: I

[Opcode]

15 11	10 5	4 0	31 27	26	25 23	22 21	20 17	16
r r r r 0	1 1 1 1 1 1	0 0 0 0 1	w w w w 0	1	0 0 0	1 0	1 1 0 0	0
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 ← 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

<Floating-point instruction>

RECIPF.D

Reciprocal of a Floating-point Value (Double)

Reciprocal (double precision)

[Instruction format] RECIPF.D reg2, reg3

[Operation] reg3 ← 1 ÷ 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 1 1 1	0
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 exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Division by zero exception (Z)

Underflow exception (U)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	1/A[I]		+∞[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

2.4.4.60 RECIPF.S

<Floating-point instruction>

Reciprocal of a Floating-point Value (Single)

RECIPF.S

Reciprocal (single precision)

[Instruction format] RECIPF.S reg2, reg3

[Operation] reg3 ← 1 ÷ 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 1 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction approximates the reciprocal of the single-precision floating-point format contents of general-purpose register reg2, and stores the result in general-purpose register reg3. The result differs from the result obtained by using the DIVF instruction.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Division by zero exception (Z)

Underflow exception (U)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	1/A[I]		+∞[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

2.4.4.61 ROUNDF.DL

<Floating-point instruction>

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

ROUNDF.DL

Convert to integer format (double-precision)

[Instruction format] ROUNDF.DL reg2, reg3

[Operation] reg3 ← round reg2 (double → 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 0	1 1 1 1 1 1	0 0 0 0 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 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 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		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.

2.4.4.62 ROUNDF.DUL

<Floating-point instruction>

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

ROUNDF.DUL

Convert to unsigned integer format (double-precision)

[Instruction format] ROUNDF.DUL reg2, reg3

[Operation] reg3 ← round reg2 (double → 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 0	1 1 1 1 1 1	1 0 0 0 0	w w w w 0	1	0 0 0	1 0	1 0 1 0	0
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 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.63 ROUNDF.DUW

<Floating-point instruction>

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

ROUNDF.DUW

Convert to unsigned integer format (double-precision)

[Instruction format] ROUNDF.DUW reg2, reg3

[Operation] reg3 ← round reg2 (double → 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 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
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 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.64 ROUNDF.DW

<Floating-point instruction>

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

ROUNDF.DW

Convert to integer format (double-precision)

[Instruction format] ROUNDF.DW reg2, reg3

[Operation] reg3 ← round reg2 (double → 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 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			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 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		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.

2.4.4.65 ROUNDF.SL

<Floating-point instruction>

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

ROUNDF.SL

Convert to integer format (single precision)

[Instruction format] ROUNDF.SL reg2, reg3

[Operation] reg3 ← round reg2 (single → 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	0 0 0 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 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		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.

2.4.4.66 ROUNDF.SUL

<Floating-point instruction>

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

ROUNDF.SUL

Convert to unsigned integer format (single precision)

[Instruction format] ROUNDF.SUL reg2, reg3

[Operation] reg3 ← round reg2 (single → 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 0 0 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 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

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

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.67 ROUNDF.SUW

<Floating-point instruction>

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

ROUNDF.SUW

Convert to unsigned integer format (single precision)

[Instruction format] ROUNDF.SUW reg2, reg3

[Operation] reg3 ← round reg2 (single → 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 0	w w w w w	1	0 0 0	1 0	0 0 0 0	0
reg2			reg3		category	type	sub-op	

[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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.68 ROUNDF.SW

<Floating-point instruction>

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

ROUNDF.SW

Convert to integer format (single precision)

[Instruction format] ROUNDF.SW reg2, reg3

[Operation] reg3 ← round reg2 (single → 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			reg3		category	type	sub-op	

[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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		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.

2.4.4.69 RSQRTF.D

<Floating-point instruction>

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

RSQRTF.D

Reciprocal of square root (double precision)

[Instruction format] RSQRTF.D reg2, reg3

[Operation] reg3 ← 1 ÷ (sqrt 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 1 0	w w w w 0	1	0 0 0	1 0	1 1 1 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction obtains the arithmetic positive square root of the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, then approximates the reciprocal of this result and stores the result in the register pair specified by general-purpose register reg3.

The result differs from the result obtained when using a combination of the SQRTF and DIVF instructions.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Division by zero exception (Z)

[Operation result]

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

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

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 2ULP error range against the results of computing 1/√x.
ULP: Unit in the Least-significant Place

2.4.4.70 RSQRTF.S

<Floating-point instruction>

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

RSQRTF.S

Reciprocal of square root (single precision)

[Instruction format] RSQRTF.S reg2, reg3

[Operation] reg3 ← 1 ÷ (sqrt 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 1 0	w w w w w	1	0 0 0	1 0	0 1 1 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction obtains the arithmetic positive square root of the single-precision floating-point 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 exceptions]
Unimplemented operation exception (E)
Invalid operation exception (V)
Inexact exception (I)
Division by zero exception (Z)

[Operation result]

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

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

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 2ULP error range against the results of computing 1/√x.
ULP: Unit in the Least-significant Place

2.4.4.71 SQRTF.D

<Floating-point instruction>

Floating-point Square Root (Double)

SQRTF.D

Square root (double precision)

[Instruction format] SQRTF.D 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	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 1 1 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction obtains the arithmetic positive square root of the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in the register pair specified by general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode. When the source operand value is –0, the result becomes –0.

[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	Q-NaN[V]	+0	−0	+∞	Q-NaN[V]	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.

2.4.4.72 SQRTF.S

<Floating-point instruction>

Floating-point Square Root (Single)

SQRTF.S

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	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 1 1	0
reg2			reg3		category	type	sub-op	

[Description] This instruction obtains the arithmetic positive square root of the single-precision floating-point format contents of general-purpose register reg2, and stores it 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. When the source operand value is –0, the result becomes –0.

[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	Q-NaN[V]	+0	−0	+∞	Q-NaN[V]	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.

2.4.4.73 SUBF.D

<Floating-point instruction>

Floating-point Subtract (Double)

SUBF.D

Floating-point subtraction (double precision)

[Instruction format] SUBF.D reg1, reg2, reg3

[Operation] reg3 ← 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 0 0 1	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction subtracts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 from 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
−∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.4.4.74 SUBF.S

<Floating-point instruction>

Floating-point Subtract (Single)

SUBF.S

Floating-point subtraction (single precision)

[Instruction format] SUBF.S reg1, reg2, reg3

[Operation] reg3 ← 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 1	0
reg2		reg1	reg3		category	type	sub-op	

[Description] This instruction subtracts the single-precision floating-point format contents of general-purpose register reg1 from 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]

reg1(A) \ reg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B – A	B – A	B – A	B – A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
−∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.4.4.75 TRFSR

<Floating-point instruction>

Transfers specified CC bit to Zero flag in PSW (Single)

TRFSR

Flag transfer

[Instruction format] TRFSR fcbit

TRFSR

[Operation] PSW.Z ← fcbit

[Format] Format F: I

[Opcode]

15 11	10 5	4 0	31 27	26	25 23	22 21	20 17	16
0 0 0 0 0	1 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	1	0 0 0	0 0	0 f f f	0
					category	type	sub-op	

Remark: fcbit: `fff`

[Description] This instruction transfers the condition bits (the CC(7:0) bits: bits 31 to 24) in the FPSR register specified by fcbit to the Z flag in the PSW. If fcbit is omitted, this instruction transfers the CC0 bit (bit 24).

[Floating-point operation exceptions] None

2.4.4.76 TRNCF.DL

<Floating-point instruction>

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

TRNCF.DL

Conversion to fixed-point format (double precision)

[Instruction format] TRNCF.DL reg2, reg3

[Operation] reg3 ← trunc reg2 (double → 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 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
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, 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		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.

2.4.4.77 TRNCF.DUL

<Floating-point instruction>

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

TRNCF.DUL

Conversion to unsigned fixed-point format (double precision)

[Instruction format] TRNCF.DUL reg2, reg3

[Operation] reg3 ← trunc reg2 (double → 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 0	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			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 unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg3.

The result is rounded in the 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 $2^{64} – 1$ 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 +∞: $2^{64} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.78 TRNCF.DUW

<Floating-point instruction>

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

TRNCF.DUW

Conversion to unsigned fixed-point format (double precision)

[Instruction format] TRNCF.DUW reg2, reg3

[Operation] reg3 ← trunc reg2 (double → 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 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			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 unsigned 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, 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.79 TRNCF.DW

<Floating-point instruction>

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

TRNCF.DW

Conversion to fixed-point format (double precision)

[Instruction format] TRNCF.DW reg2, reg3

[Operation] reg3 ← trunc reg2 (double → 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 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
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 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		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.

2.4.4.80 TRNCF.SL

<Floating-point instruction>

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

TRNCF.SL

Conversion to fixed-point format (single precision)

[Instruction format] TRNCF.SL reg2, reg3

[Operation] reg3 ← trunc reg2 (single → 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	0 0 0 0 1	w w w w 0	1	0 0 0	1 0	0 0 1 0	0
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg2 to 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 +∞: $2^{63} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{63}$ 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 (Integer)		0 (Integer)		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.

2.4.4.81 TRNCF.SUL

<Floating-point instruction>

TRNCF.SUL	Floating-point Convert Single to Unsigned-Long, round toward zero (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] TRNCF.SUL reg2, reg3

[Operation] reg3 ← trunc reg2 (single → 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 0 0 0 1	w w w w 0	1	0 0 0	1 0	0 0 1 0	0
reg2			reg3		category	type	sub-op	

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

The result is rounded in the zero direction, regardless of the current rounding mode.

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

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.82 TRNCF.SUW

<Floating-point instruction>

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

TRNCF.SUW

Conversion to unsigned fixed-point format (single precision)

[Instruction format] TRNCF.SUW reg2, reg3

[Operation] reg3 ← trunc reg2 (single → 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
reg2			reg3		category	type	sub-op	

[Description] This instruction arithmetically converts the single-precision floating-point number 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 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

[Operation result]

reg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Integer)	0[V]	0 (Integer)		Max U-Int[V]	0[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.

2.4.4.83 TRNCF.SW

<Floating-point instruction>

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

TRNCF.SW

Conversion to fixed-point format (single precision)

[Instruction format] TRNCF.SW reg2, reg3

[Operation] reg3 ← trunc reg2 (single → 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 1	w w w w w	1	0 0 0	1 0	0 0 0 0	0
reg2			reg3		category	type	sub-op	

[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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ 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 (Integer)		0 (Integer)		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.

2.5 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.

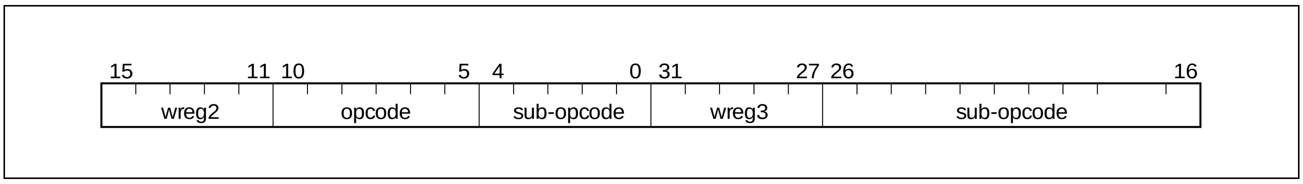
2.5.1 Instruction Format

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) → 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

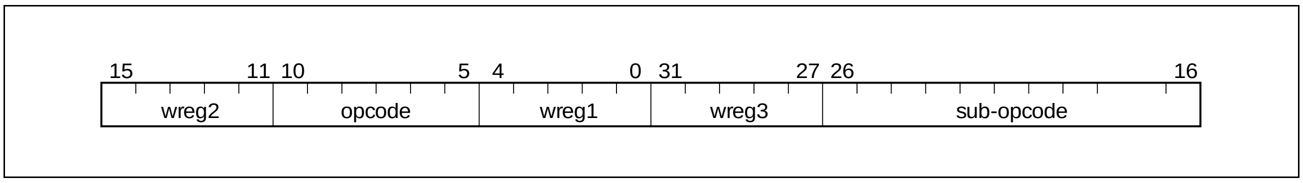
(1) Format M: 2OP

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.



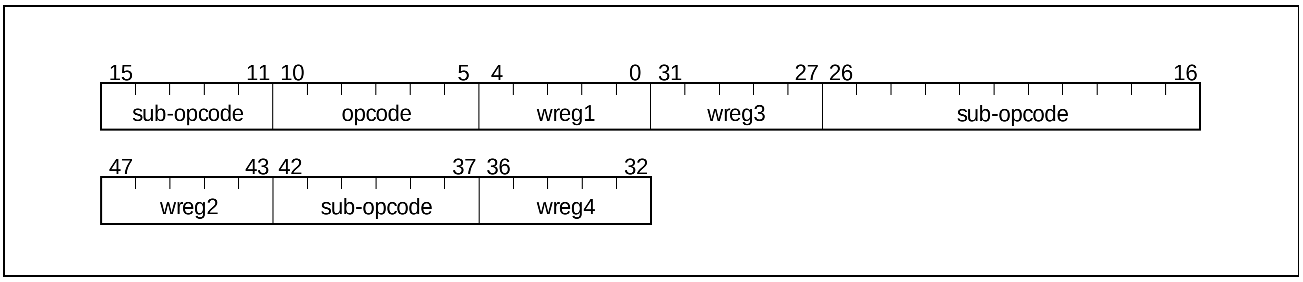
(2) Format M: 3OP

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

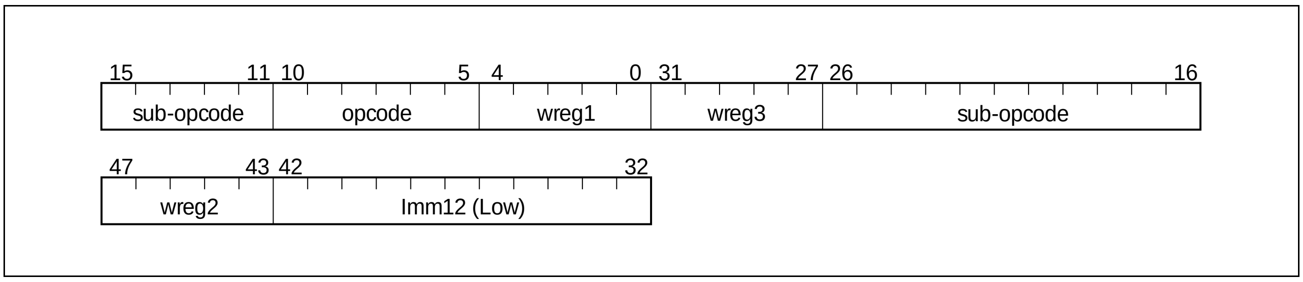
A 48-bit instruction format that has a 6-bit opcode field and 4 vector register specification fields and that combines the other bits in the sub-opcode field.



(4) Format M: imm12

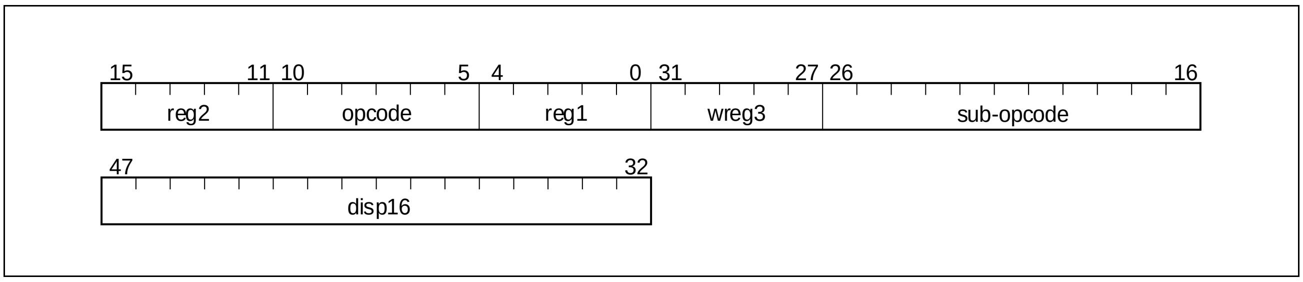
A 48-bit instruction format that has a 6-bit opcode field, 3 vector register specification fields, and a 12-bit immediate field and that combines the other bits in the sub-opcode field.

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.

Table 2.16 Instruction Format Legends

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 2.17 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 *a* and, if one is detected, suspends the instruction execution and performs exception processing.
result	Reflect result in flags
zero-extend (*n*)	Zero-extends *n* to word
sign-extend (*n*)	Sign-extends *n* to word
load-memory (*a, b*)	Reads data of size *b* from address *a*
store-memory (*a, b, c*)	Writes data *b* of size *c* to address *a*
abs (*n*)	Absolute value of *n*
ceil (*n*)	Rounds *n* toward +∞
cvt (*n*)	Converts type of *n* according to rounding mode
floor (*n*)	Rounds *n* toward −∞
max (*a, b*)	Maximum value of *a* and *b*
min (*a, b*)	Minimum value of *a* and *b*
neg (*n*)	Sign inversion of *n*
round (*n*)	Rounds *n* to closest value
sqrt (*n*)	Square root of *n*
trunc (*n*)	Rounds *n* to zero
fma (*a, b, c*)	Result of multiplying *a* and *b* and then adding *c*
fms (*a, b, c*)	Result of multiplying *a* and *b* and then subtracting *c*
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)
(*n*:*m*) or (*n*)	Bit selection
(h*x*)	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*x*)	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*x*)	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
I	1-bit data of immediate (indicates higher bits of immediate)
i	1-bit data of immediate

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

<Extended Floating-point Instructions>

Floating-point Absolute Value (Single)

ABSF.S4

Floating-point absolute value (single)

[Instruction format] ABSF.S4 wreg2, wreg3

[Operation]
WR[wreg3](w3) ← abs(WR[wreg2](w3))
WR[wreg3](w2) ← abs(WR[wreg2](w2))
WR[wreg3](w1) ← abs(WR[wreg2](w1))
WR[wreg3](w0) ← abs(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 0 0 0 0	w w w w w	1	0 1 1	0 1 0 0 0 0	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

<Extended Floating-point Instructions>

Floating-point SIMD Add (single)

ADDF.S4

Extended floating-point add (single precision)

[Instruction format] ADDF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) + WR[wreg1](w3)

WR[wreg3](w2) ← WR[wreg2](w2) + WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) + WR[wreg1](w1)

WR[wreg3](w0) ← 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 exceptions]

Unimplemented operation exception (E)

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	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

Floating-point SIMD Add Reduction (single)

ADDRF.S4

Extended floating-point add reduction (single precision)

[Instruction format] ADDRF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) + WR[wreg2](w2)

WR[wreg3](w2) ← WR[wreg1](w3) + WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) + WR[wreg2](w0)

WR[wreg3](w0) ← 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 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 exceptions]

Unimplemented operation exception (E)

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]

A*4 \ B*4	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	A + B	A + B	A + B	A + B	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	A + B	A + B	A + B	A + B	+∞	−∞	Q-NaN	Q-NaN[V]
+0	A + B	A + B	A + B	A + B	+∞	−∞	Q-NaN	Q-NaN[V]
−0	A + B	A + B	A + B	A + B	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

Floating-point SIMD Add/Subtract (single)

ADDSUBF.S4

Extended floating-point add/subtract (single precision)

[Instruction format] ADDSUBF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) + WR[wreg1](w3)

WR[wreg3](w2) ← WR[wreg2](w2) − WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) + WR[wreg1](w1)

WR[wreg3](w0) ← 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	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 exceptions]

Unimplemented operation exception (E)

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	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B + A	B + A	B + A	B + A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]

<Even-number element>

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

2.5.4.5 ADDSUBNF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Add/Subtract Negative (single)

ADDSUBNF.S4

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

[Instruction format] ADDSUBNF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(WR[wreg2](w3) + WR[wreg1](w3))

WR[wreg3](w2) ← neg(WR[wreg2](w2) − WR[wreg1](w2))

WR[wreg3](w1) ← neg(WR[wreg2](w1) + WR[wreg1](w1))

WR[wreg3](w0) ← neg(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	1 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-number element of the vector register wreg2 and the contents of single-precision floating-point data in the odd-number element of the vector register wreg1 and stores the results in the odd-number element of the vector register wreg3 with their sign inverted.

Subtracts the contents of single-precision floating-point data in the even-number element of the vector register wreg1 from the contents of 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 with their sign inverted.

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 exceptions]

Unimplemented operation exception (E)

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]

<Odd-number element>

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

<Even-number element>

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

<Extended Floating-point Instructions>

Floating-point SIMD Add/Subtract Negative Exchange (single)

ADDSUBNXF.S4

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

[Instruction format] ADDSUBNXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(WR[wreg2](w3) + WR[wreg1](w2))

WR[wreg3](w2) ← neg(WR[wreg2](w2) − WR[wreg1](w3))

WR[wreg3](w1) ← neg(WR[wreg2](w1) + WR[wreg1](w0))

WR[wreg3](w0) ← neg(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 w	1	0 1 1	1 0 1 1 1 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 single-precision floating-point data in the even-number element of the vector register wreg1 and stores the results in the odd-number element of the vector register wreg3 with their sign inverted.

Subtracts the contents of single-precision floating-point data in the odd-number element of the vector register wreg1 from the contents of 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 with their sign inverted.

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 exceptions]

Unimplemented operation exception (E)

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]

<Odd-number element>

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

<Even-number element>

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

Floating-point SIMD Add/Subtract Exchange (single)

ADDSUBXF.S4

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

[Instruction format] ADDSUBXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) + WR[wreg1](w2)

WR[wreg3](w2) ← WR[wreg2](w2) − WR[wreg1](w3)

WR[wreg3](w1) ← WR[wreg2](w1) + WR[wreg1](w0)

WR[wreg3](w0) ← 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 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)

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]

<Odd-number element>

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

<Even-number element>

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

Floating-point SIMD Add Exchange (single)

ADDXF.S4

Extended floating-point add exchange (single precision)

[Instruction format] ADDXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) + WR[wreg1](w2)

WR[wreg3](w2) ← WR[wreg2](w2) + WR[wreg1](w3)

WR[wreg3](w1) ← WR[wreg2](w1) + WR[wreg1](w0)

WR[wreg3](w0) ← 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 w	1	0 1 1	1 0 0 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.

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 exceptions]

Unimplemented operation exception (E)

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]

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

2.5.4.9 CEILF.SUW4

<Extended Floating-point Instructions>

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

CEILF.SUW4

Extended floating-point type conversion (Single precision → unsigned integer)

[Instruction format] CEILF.SUW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← ceil(WR[wreg2](w3) (single → unsigned word))

WR[wreg3](w2) ← ceil(WR[wreg2](w2) (single → unsigned word))

WR[wreg3](w1) ← ceil(WR[wreg2](w1) (single → unsigned word))

WR[wreg3](w0) ← ceil(WR[wreg2](w0) (single → unsigned word))

[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	0 0 1 0 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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 +∞ 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 +∞: $2^{32} - 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

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 (Integer)		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.

2.5.4.10 CEILF.SW4

<Extended Floating-point Instructions>

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

CEILF.SW4

Extended floating-point type conversion (Single precision → Integer)

[Instruction format] CEILF.SW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← ceil(WR[wreg2](w3) (single → word))

WR[wreg3](w2) ← ceil(WR[wreg2](w2) (single → word))

WR[wreg3](w1) ← ceil(WR[wreg2](w1) (single → word))

WR[wreg3](w0) ← ceil(WR[wreg2](w0) (single → word))

[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	0 0 1 0 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-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 rounded toward +∞ 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ is returned.

[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 (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.

2.5.4.11 CMOVF.W4

<Extended Floating-point Instructions>

Floating-point SIMD Conditional Move

CMOVF.W4

Conditional move of vector register (Single precision)

[Instruction format] CMOVF.W4 wreg4, wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← (WR[wreg4](w3) != 0) ? WR[wreg1](w3): WR[wreg2](w3)

WR[wreg3](w2) ← (WR[wreg4](w2) != 0) ? WR[wreg1](w2): WR[wreg2](w2)

WR[wreg3](w1) ← (WR[wreg4](w1) != 0) ? WR[wreg1](w1): WR[wreg2](w1)

WR[wreg3](w0) ← (WR[wreg4](w0) != 0) ? WR[wreg1](w0): WR[wreg2](w0)

[Format] Format M: 4OP

[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	1 1 0 0 0 0 1 1 1 0	1
sub-op		wreg1	wreg3	sub-op	

47 43	42 37	36 32
r r r r r	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>

Floating-point SIMD Comparison (single)

CMPF.S4

Extended floating-point comparison (Single precision)

[Instruction format] CMPF.S4 fcond, wreg1, wreg2, wreg3

[Operation]

```
WR[wreg3](w3) ← cmpf (fcond, WR[wreg2](w3), WR[wreg1](w3))
WR[wreg3](w2) ← cmpf (fcond, WR[wreg2](w2), WR[wreg1](w2))
WR[wreg3](w1) ← cmpf (fcond, WR[wreg2](w1), WR[wreg1](w1))
WR[wreg3](w0) ← cmpf (fcond, WR[wreg2](w0), WR[wreg1](w0))
    cmpf (fcond, A, B) =
      if isNaN(A) or isNaN(B) then
            result.less ← 0
            result.equal ← 0
            result.unordered ← 1
            if fcond[3] == 1 then
               Invalid operation exception is detected.
            endif
      else
            result.less    ←A < B
            result.equal ←A == B
            result.unordered ← 0
      endif
      result ← (fcond[2] & result.less) | (fcond[1] & result.equal) |
                  (fcond[0] & result.unordered)
      return (result == true) ? FFFF FFFF_H: 0000 0000_H
```

[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 0 F F F F	0
wreg2		wreg1	wreg3		category	sub-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.

Table 2.19 Comparison Conditions

Comparison Conditions				Detection of Invalid operation Exception by Unordered
	fcond	Definition	Description	
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

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
    BT          L2                  # Branch to L2 if true.
    TRFSRV.W4   5,wr16
    BT          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.
L2:
# Put the code to be executed when wr14 < wr12 is established for all elements.


# Example of using the comparison for notifying Q-NaN
    CMPF.S4     LT,wr12,wr14,wr15   # Check wr14 < wr12 for each
                                      element.
    TRFSRV.W4   4,wr15
    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 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.

[Operation result]

[Condition code (fcond) = 0 to 7]

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

[Condition code (fcond) = 8 to 15]

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

Note: [ ] indicates an exception that must occur.

2.5.4.13 CVTF.HS4

<Extended Floating-point Instructions>

Floating-point SIMD Convert Half to Single (single)

CVTF.HS4

Extended floating-point type conversion (Half-precision → Single precision)

[Instruction format] CVTF.HS4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← cvt(WR[wreg2](h6) (half → single))

WR[wreg3](w2) ← cvt(WR[wreg2](h4) (half → single))

WR[wreg3](w1) ← cvt(WR[wreg2](h2) (half → single))

WR[wreg3](w0) ← cvt(WR[wreg2](h0) (half → single))

[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 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[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.

[Operation result]

wreg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	A (Single)		+0	−0	+∞	−∞	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

<Extended Floating-point Instructions>

Floating-point SIMD Convert Single to Half (single)

CVTF.SH4

Extended floating-point type conversion (Single precision → Half-precision)

[Instruction format] CVTF.SH4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← zero-extend(cvt(WR[wreg2](w3) (single → half)))

WR[wreg3](w2) ← zero-extend(cvt(WR[wreg2](w2) (single → half)))

WR[wreg3](w1) ← zero-extend(cvt(WR[wreg2](w1) (single → half)))

WR[wreg3](w0) ← zero-extend(cvt(WR[wreg2](w0) (single → 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 0 0 0	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 exceptions]

Unimplemented operation exception (E)

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 (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.

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

2.5.4.15 CVTF.SUW4

<Extended Floating-point Instructions>

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

CVTF.SUW4

Extended floating-point type conversion (Single precision → Unsigned integer)

[Instruction format] CVTF.SUW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← cvt(WR[wreg2](w3) (single → unsigned word))
WR[wreg3](w2) ← cvt(WR[wreg2](w2) (single → unsigned word))
WR[wreg3](w1) ← cvt(WR[wreg2](w1) (single → unsigned word))
WR[wreg3](w0) ← cvt(WR[wreg2](w0) (single → unsigned 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 1 0 0 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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.

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 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 (Integer)	0[V]	0 (Integer)		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.

2.5.4.16 CVTF.SW4

<Extended Floating-point Instructions>

Floating-point SIMD Convert Single to Word (single)

CVTF.SW4

Extended floating-point type conversion (Single precision → Integer)

[Instruction format] CVTF.SW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← cvt(WR[wreg2](w3) (single → word))

WR[wreg3](w2) ← cvt(WR[wreg2](w2) (single → word))

WR[wreg3](w1) ← cvt(WR[wreg2](w1) (single → word))

WR[wreg3](w0) ← cvt(WR[wreg2](w0) (single → 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 1 0 0 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[Descriptions]

This instruction arithmetically converts the single-precision floating-point format contents in vector register wreg2 to 32-bit integer format and stores the results in vector register wreg3.

When the source operand is infinite, 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ is returned.

[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 (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.

2.5.4.17 CVTF.UWS4

<Extended Floating-point Instructions>

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

CVTF.UWS4

Extended floating-point type conversion (Unsigned integer → Single precision)

[Instruction format] CVTF.UWS4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← cvt(WR[wreg2](w3) (unsigned word → single))
WR[wreg3](w2) ← cvt(WR[wreg2](w2) (unsigned word → single))
WR[wreg3](w1) ← cvt(WR[wreg2](w1) (unsigned word → single))
WR[wreg3](w0) ← cvt(WR[wreg2](w0) (unsigned word → single))

[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 0 1 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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 (Normal)		+0

2.5.4.18 CVTF.WS4

<Extended Floating-point Instructions>

Floating-point SIMD Convert Word to Single (single)

CVTF.WS4

Extended floating-point type conversion (Integer → Single precision)

[Instruction format] CVTF.WS4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← cvt(WR[wreg2](w3) (word → single))
WR[wreg3](w2) ← cvt(WR[wreg2](w2) (word → single))
WR[wreg3](w1) ← cvt(WR[wreg2](w1) (word → single))
WR[wreg3](w0) ← cvt(WR[wreg2](w0) (word → single))

[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 0 1 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[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 (Normal)		+0

2.5.4.19 DIVF.S4

<Extended Floating-point Instructions>

Floating-point Divide (Single)

DIVF.S4

Extended floating-point divide (Single precision)

[Instruction format] DIVF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) ÷ WR[wreg1](w3)

WR[wreg3](w2) ← WR[wreg2](w2) ÷ WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) ÷ WR[wreg1](w1)

WR[wreg3](w0) ← WR[wreg2](w0) ÷ 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 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

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]

wreg1(A) \ wreg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B ÷ A	B ÷ A	B ÷ A	B ÷ A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B ÷ A	B ÷ A	B ÷ A	B ÷ A	−∞	+∞	Q-NaN	Q-NaN[V]
+0	±∞[Z]	±∞[Z]	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−0	±∞[Z]	±∞[Z]	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
+∞	+0	−0	+0	−0	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	−0	+0	−0	+0	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

2.5.4.20 FLOORF.SUW4

<Extended Floating-point Instructions>

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

FLOORF.SUW4

Extended floating-point type conversion (Single precision → Unsigned integer)

[Instruction format] FLOORF.SUW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← floor WR[wreg2](w3) (single → unsigned word)

WR[wreg3](w2) ← floor WR[wreg2](w2) (single → unsigned word)

WR[wreg3](w1) ← floor WR[wreg2](w1) (single → unsigned word)

WR[wreg3](w0) ← floor WR[wreg2](w0) (single → unsigned 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 1 1 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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 –∞ 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 +∞: 2^{32} – 1 is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

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 (Integer)		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.

2.5.4.21 FLOORF.SW4

<Extended Floating-point Instructions>

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

FLOORF.SW4

Extended floating-point type conversion (Single precision → Integer)

[Instruction format]	FLOORF.SW4 wreg2, wreg3
[Operation]	WR[wreg3](w3) ← floor WR[wreg2](w3) (single → word) WR[wreg3](w2) ← floor WR[wreg2](w2) (single → word) WR[wreg3](w1) ← floor WR[wreg2](w1) (single → word) WR[wreg3](w0) ← floor WR[wreg2](w0) (single → 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 1 1 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[Descriptions]

This instruction arithmetically converts the single-precision floating-point format contents in vector register wreg2 to 32-bit integer format and stores the results in vector register wreg3.

The results are rounded toward -∞ 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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $– 2^{31}$ is returned.

[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 (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.

2.5.4.22 FLPV.S4

<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) ← WR[wreg2](w3)
    WR[wreg3](w2) ← neg(WR[wreg2](w2))
    WR[wreg3](w1) ← WR[wreg2](w1)
    WR[wreg3](w0) ← 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) ← neg(WR[wreg2](w2))
    WR[wreg3](w2) ← WR[wreg2](w3)
    WR[wreg3](w1) ← neg(WR[wreg2](w0))
    WR[wreg3](w0) ← WR[wreg2](w1)
```

[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 i i	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-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 exceptions]	None

2.5.4.23 FMAF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Fused-Multiply-Add (Single)

FMAF.S4

Extended floating-point fused-multiply-add operation (Single precision)

[Instruction format] FMAF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← fma(WR[wreg2](w3), WR[wreg1](w3), WR[wreg3](w3))

WR[wreg3](w2) ← fma(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2))

WR[wreg3](w1) ← fma(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1))

WR[wreg3](w0) ← 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 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

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]

wreg3(C) \ wreg2(B)	wreg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	fma (B, A, C)				+∞	−∞		
	−Normal					−∞	+∞		
	±0					Q-NaN[V]			
	+∞	+∞	−∞	Q-NaN[V]		+∞	−∞		
	−∞	−∞	+∞			−∞	+∞		
±0	+Normal	fma (B, A, C)				+∞	−∞		
	−Normal					−∞	+∞		
	±0					Q-NaN[V]			
	+∞	+∞	−∞	Q-NaN[V]		+∞	−∞		
	−∞	−∞	+∞			−∞	+∞		
+∞	+Normal	+∞				+∞	Q-NaN[V]		
	−Normal					Q-NaN[V]	+∞		
	±0					Q-NaN[V]			
	+∞	+∞	Q-NaN[V]	Q-NaN[V]		+∞	Q-NaN[V]		
	−∞	Q-NaN[V]	+∞			Q-NaN[V]	+∞		
−∞	+Normal	−∞				Q-NaN[V]	−∞		
	−Normal					−∞	Q-NaN[V]		
	±0					Q-NaN[V]			
	+∞	Q-NaN[V]	−∞	Q-NaN[V]		Q-NaN[V]	−∞		
	−∞	−∞	Q-NaN[V]			−∞	Q-NaN[V]		
Q-NaN	±Normal	Q-NaN							
	±0								
	±∞								
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

<Extended Floating-point Instructions>

Floating-point SIMD Fused-Multiply-Subtract (Single)

FMSF.S4

Extended floating-point fused-multiply-subtract operation (Single precision)

[Instruction format] FMSF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← fms(WR[wreg2](w3), WR[wreg1](w3), WR[wreg3](w3))

WR[wreg3](w2) ← fms(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2))

WR[wreg3](w1) ← fms(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1))

WR[wreg3](w0) ← fms(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 1	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, subtracts the single-precision floating-point format contents in vector register wreg3, and stores the results in the corresponding elements of vector register wreg3. The results of the multiply operation is not rounded, but the results of the subtract operation 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)

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]

wreg3(C)	wreg2(B) / wreg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
±0	+Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	−Normal	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	±0	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
+∞	+Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	+Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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 Instructions>

Floating-point SIMD Fused-Negative-Multiply-Add (Single)

FNMAF.S4

Extended floating-point fused-multiply-add operation (Single precision)

[Instruction format] FNMAF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(fma(WR[wreg2](w3), WR[wreg1](w3), WR[wreg3](w3)))
WR[wreg3](w2) ← neg(fma(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2)))
WR[wreg3](w1) ← neg(fma(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1)))
WR[wreg3](w0) ← neg(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 1 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 vector register wreg3, and stores the results in the corresponding element of vector register wreg3 with its sign inverted. 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)
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]

wreg3(C)	wreg2(B) / wreg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
±0	+Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	–fma (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
−∞	+Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don't care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don't care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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 Instructions>

Floating-point SIMD Fused-Negative-Multiply-Subtract (Single)

FNMSF.S4

Extended floating-point fused-multiply-subtract operation (Single precision)

[Instruction format] FNMSF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(fms(WR[wreg2](w3), WR[wreg1](w3), WR[wreg3](w3)))

WR[wreg3](w2) ← neg(fms(WR[wreg2](w2), WR[wreg1](w2), WR[wreg3](w2)))

WR[wreg3](w1) ← neg(fms(WR[wreg2](w1), WR[wreg1](w1), WR[wreg3](w1)))

WR[wreg3](w0) ← neg(fms(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 1 1	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, subtracts the single-precision floating-point format contents in vector register wreg3, and stores the results in the corresponding elements of vector register wreg3 with its sign inverted. The results of the multiply operation is not rounded, but the results of the subtract operation 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)

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]

wreg3(C)	wreg2(B) wreg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
±0	+Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	−∞	+∞	Q-NaN	Q-NaN[V]
	−Normal	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	+∞	−∞	Q-NaN	Q-NaN[V]
	±0	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	–fms (B, A, C)	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	+Normal	+∞	+∞	+∞	+∞	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−Normal	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	±0	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN	Q-NaN[V]
	−∞	+∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
−∞	+Normal	−∞	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−Normal	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
	±0	−∞	−∞	−∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
	+∞	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
	−∞	Q-NaN[V]	−∞	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
Q-NaN	±Normal	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±0	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
	±∞	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Not S-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
Don’t care	S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]
S-NaN	Don’t care	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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.

[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 ← GR[reg1] + sign-extend (disp16)Note 1
CheckException(MAE)
CheckException(MDP)
val ← Load-memory(adr, Double-word)
WR[wreg3](dw1) ← (imm2[1] == 1) ? val: WR[wreg3](dw1)
WR[wreg3](dw0) ← (imm2[0] == 1) ? val: WR[wreg3](dw0)
```

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 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
d d d d d d d d d d d d d	0	0	0
disp16			

Where `ii` = 2-bit immediate data
`ddddddddddddd` = 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 double-word 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>

Load Vector (Quad-Word)

LDV.QW

Load quad-word data to vector register

[Instruction format] LDV.QW disp16[reg1], wreg3

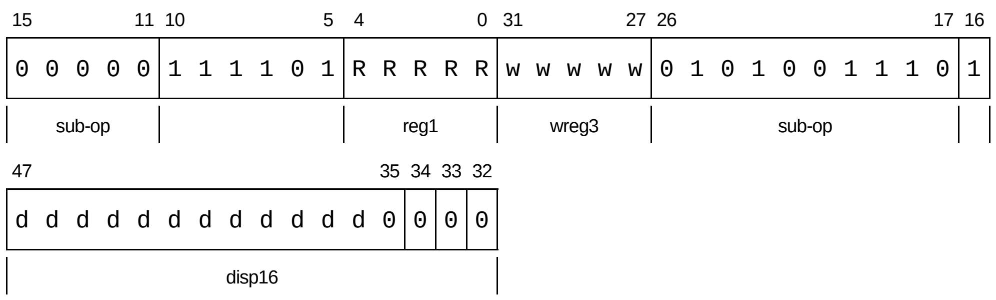
[Operation]

```
adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException(MAE)
CheckException(MDP)
WR[wreg3] ← Load-memory(adr, Quad-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 0 1 1 1 1 0 1 R R R R R w w w w w 0 1 0 1 0 0 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 d 0 0 0 0
disp16

dddddddddddd = 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)Note 1
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 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
sub-op		reg1	wreg3	sub-op	

47 35	34	33	32
d d d d d d d d d d d d d	d	0	0
disp16			

Where `iiii` = 4-bit immediate data

`dddddddddddddd` = 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 4-bit 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>

LDVZ.H4	Load Vector at Even Halfword field Interleaved load of halfword to vector register

[Instruction format] LDVZ.H4 disp16[reg1], wreg3

[Operation]

```
adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException(MAE)
CheckException(MDP)
val[63:0] ← Load-memory(adr, Double-word)
WR[wreg3](h7) ← 0
WR[wreg3](h6) ← val[63:48]
WR[wreg3](h5) ← 0
WR[wreg3](h4) ← val[47:32]
WR[wreg3](h3) ← 0
WR[wreg3](h2) ← val[31:16]
WR[wreg3](h1) ← 0
WR[wreg3](h0) ← val[15:0]
```

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 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
sub-op		reg1	wreg3	sub-op	

47 35	34	33	32
d d d d d d d d d d d d d	0	0	0
disp16			

Where ddddddddddddd = Higher-order 13 bits of disp16

[Descriptions]	This instruction reads double-word data from the 32-bit address which is generated by adding together the data in general-purpose register reg1 and the 16-bit displacement data that is sign-extended to word length. The read double-word data is zero-extended in 16-bit 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>

Floating-point SIMD Maximum (single)

MAXF.S4

Extended floating-point maximum value (Single precision)

[Instruction format] MAXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← max (WR[wreg2](w3), WR[wreg1](w3))

WR[wreg3](w2) ← max (WR[wreg2](w2), WR[wreg1](w2))

WR[wreg3](w1) ← max (WR[wreg2](w1), WR[wreg1](w1))

WR[wreg3](w0) ← max (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 1 0 1	0
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 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.

[Operation result]

wreg2(B) / wreg1(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
−Normal	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
+0	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
−0	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
+∞	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
−∞	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	max (B, A)	wreg1(A)	Q-NaN[V]
Q-NaN	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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

<Extended Floating-point Instructions>

Floating-point SIMD Maximum Reduction (single)

MAXRF.S4

Extended floating-point maximum value reduction (Single precision)

[Instruction format] MAXRF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← max (WR[wreg2](w3), WR[wreg2](w2))

WR[wreg3](w2) ← max (WR[wreg1](w3), WR[wreg1](w2))

WR[wreg3](w1) ← max (WR[wreg2](w1), WR[wreg2](w0))

WR[wreg3](w0) ← max (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 w	1	0 1 1	0 1 1 1 0 1	0
wreg2		wreg1	wreg3		category	sub-op	

[Descriptions] This instruction stores the maximum values 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 maximum values 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.

[Operation result]

A^{*2} \ B^{*2}	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
−Normal	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
+0	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
−0	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
+∞	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
−∞	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	max (A, B)	A	Q-NaN[V]
Q-NaN	B	B	B	B	B	B	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

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) ← min (WR[wreg2](w3), WR[wreg1](w3)) WR[wreg3](w2) ← min (WR[wreg2](w2), WR[wreg1](w2)) WR[wreg3](w1) ← min (WR[wreg2](w1), WR[wreg1](w1)) WR[wreg3](w0) ← 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
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 1 0	0
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.

[Operation result]

wreg1(A) \ wreg2(B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
−Normal	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
+0	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
−0	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
+∞	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
−∞	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	min (B, A)	wreg1(A)	Q-NaN[V]
Q-NaN	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	wreg2(B)	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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 Instructions>

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) ← min (WR[wreg2](w3), WR[wreg2](w2))

WR[wreg3](w2) ← min (WR[wreg1](w3), WR[wreg1](w2))

WR[wreg3](w1) ← min (WR[wreg2](w1), WR[wreg2](w0))

WR[wreg3](w0) ← 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 w	1	0 1 1	0 1 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.

[Operation result]

A*2 \ B*2	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
−Normal	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
+0	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
−0	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
+∞	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
−∞	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	min (A, B)	A	Q-NaN[V]
Q-NaN	B	B	B	B	B	B	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

MOVV.W4

Move vector register to vector register

Move vector register

[Instruction format] MOVV.W4 wreg2, wreg3

[Operation] WR[wreg3] ← 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
wreg2		sub-op	wreg3		category	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) ← WR[wreg2](w3) × WR[wreg1](w3) WR[wreg3](w2) ← WR[wreg2](w2) × WR[wreg1](w2) WR[wreg3](w1) ← WR[wreg2](w1) × WR[wreg1](w1) WR[wreg3](w0) ← 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 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 exceptions]

Unimplemented operation exception (E)

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	B × A	B × A	B × A	B × A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B × A	B × A	B × A	B × A	−∞	+∞	Q-NaN	Q-NaN[V]
+0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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) ← WR[wreg2](w3) × WR[wreg2](w2)

WR[wreg3](w2) ← WR[wreg1](w3) × WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) × WR[wreg2](w0)

WR[wreg3](w0) ← 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 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 exceptions]

Unimplemented operation exception (E)

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]

A*4 \ B*4	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	A × B	A × B	A × B	A × B	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	A × B	A × B	A × B	A × B	−∞	+∞	Q-NaN	Q-NaN[V]
+0	A × B	A × B	A × B	A × B	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−0	A × B	A × B	A × B	A × B	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

Floating-point SIMD Multiply Exchange (single)

MULXF.S4

Extended floating-point multiply exchange (Single precision)

[Instruction format] MULXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) × WR[wreg1](w2)

WR[wreg3](w2) ← WR[wreg2](w2) × WR[wreg1](w3)

WR[wreg3](w1) ← WR[wreg2](w1) × WR[wreg1](w0)

WR[wreg3](w0) ← 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 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 exceptions]

Unimplemented operation exception (E)

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	B × A	B × A	B × A	B × A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B × A	B × A	B × A	B × A	−∞	+∞	Q-NaN	Q-NaN[V]
+0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
−0	B × A	B × A	B × A	B × A	Q-NaN[V]	Q-NaN[V]	Q-NaN	Q-NaN[V]
+∞	+∞	−∞	Q-NaN[V]	Q-NaN[V]	+∞	−∞	Q-NaN	Q-NaN[V]
−∞	−∞	+∞	Q-NaN[V]	Q-NaN[V]	−∞	+∞	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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 Instructions>

Floating-point SIMD Negative (single)

NEGF.S4

Extended floating-point sign inversion (Single precision)

[Instruction format] NEGF.S4 wreg2, wreg3

[Operation]
WR[wreg3](w3) ← neg(WR[wreg2](w3))
WR[wreg3](w2) ← neg(WR[wreg2](w2))
WR[wreg3](w1) ← neg(WR[wreg2](w1))
WR[wreg3](w0) ← 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 0 0 0 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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) ← 1 ÷ WR[wreg2](w3)

WR[wreg3](w2) ← 1 ÷ WR[wreg2](w2)

WR[wreg3](w1) ← 1 ÷ WR[wreg2](w1)

WR[wreg3](w0) ← 1 ÷ 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 0 0 1 1	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[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.

[Operation result]

wreg2(A)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	1/A[I]		+∞[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 Instructions>

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

ROUNDF.SUW4

Extended floating-point type conversion (Single precision → Unsigned integer)

[Instruction format] ROUNDF.SUW4 wreg2, wreg3

[Operation]
WR[wreg3](w3) ← round WR[wreg2](w3) (single → unsigned word)
WR[wreg3](w2) ← round WR[wreg2](w2) (single → unsigned word)
WR[wreg3](w1) ← round WR[wreg2](w1) (single → unsigned word)
WR[wreg3](w0) ← round WR[wreg2](w0) (single → unsigned 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 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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 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 +∞: $2^{32} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: 0 is returned.

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

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 (Integer)		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.SW4

Extended floating-point type conversion (Single precision → Integer)

[Instruction format] ROUNDF.SW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← round WR[wreg2](w3) (single → word)

WR[wreg3](w2) ← round WR[wreg2](w2) (single → word)

WR[wreg3](w1) ← round WR[wreg2](w1) (single → word)

WR[wreg3](w0) ← round WR[wreg2](w0) (single → 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 0 0 0	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 +∞: $2^{31} – 1$ is returned.
- Source is a negative number, not-a-number, or –∞: $–2^{31}$ is returned.

[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 (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 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← 1 ÷ (sqrt WR[wreg2](w3))

WR[wreg3](w2) ← 1 ÷ (sqrt WR[wreg2](w2))

WR[wreg3](w1) ← 1 ÷ (sqrt WR[wreg2](w1))

WR[wreg3](w0) ← 1 ÷ (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 0 1 0 0	w w w w w	1	0 1 1	0 1 0 0 0 0	0
wreg2		sub-op	wreg3		category	sub-op	

[Descriptions] This instruction approximates the reciprocal of 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 results differs from the results obtained by executing a combination of SQRTF.S4 and DIVF.S4 instructions.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

Division by zero exception (Z)

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[I]	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/√x.
ULP: Unit in the Least-significant Place

2.5.4.44 SHFLV.W4

<Extended Floating-point Instructions>

Vector Shuffle

SHFLV.W4

Vector shuffle

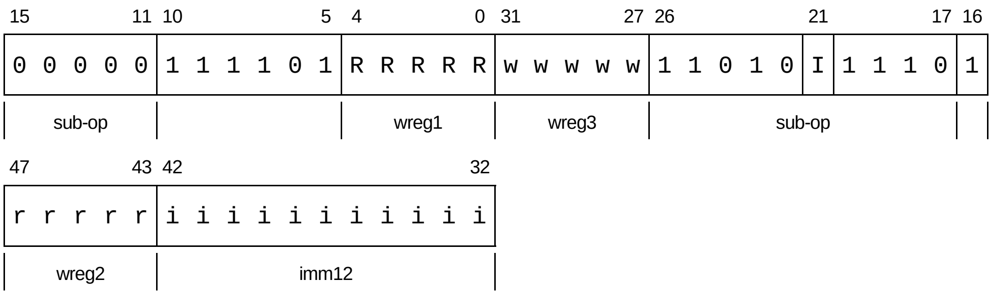
[Instruction format] SHFLV.W4 imm12, wreg1, wreg2, wreg3

[Operation]

```
Sel3[2:0] || Sel2[2:0] || Sel1[2:0] || Sel0[2:0] ← imm12
foreach N (0, 1, 2, 3)
    case
        SelN = 7: WR[wreg3](wN) ← WR[wreg2](w3)
        SelN = 6: WR[wreg3](wN) ← WR[wreg2](w2)
        SelN = 5: WR[wreg3](wN) ← WR[wreg2](w1)
        SelN = 4: WR[wreg3](wN) ← WR[wreg2](w0)
        SelN = 3: WR[wreg3](wN) ← WR[wreg1](w3)
        SelN = 2: WR[wreg3](wN) ← WR[wreg1](w2)
        SelN = 1: WR[wreg3](wN) ← WR[wreg1](w1)
        SelN = 0: WR[wreg3](wN) ← WR[wreg1](w0)
```

[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 w | 1 1 0 1 0 | I | 1 1 1 0 | 1
sub-op
wreg1
wreg3
sub-op
47 43 42 32
r r r r r | i i i i i i i i i i i
wreg2
imm12

Where `I` = the higher-order 1 bit of 12-bit immediate data

`iiiiiiiiiii` = The lower-order 11 bits of 12-bit immediate data

[Descriptions] This instruction selects an arbitrary element of vector register wreg2 or wreg1 according to the value of the 12-bit immediate data and updates each element of vector register wreg3 with the results. The 12-bit immediate data is aligned in 3 bit units, each of which specifies one element to be selected.

For example, the element of vector register wreg1 or wreg2 designated by bits 2-0 of the 12-bit immediate data is stored in the 0th element of vector register wreg3.

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]

$WR[wreg3](w3) \leftarrow \text{sqrt } WR[wreg2](w3)$

$WR[wreg3](w2) \leftarrow \text{sqrt } WR[wreg2](w2)$

$WR[wreg3](w1) \leftarrow \text{sqrt } WR[wreg2](w1)$

$WR[wreg3](w0) \leftarrow \text{sqrt } WR[wreg2](w0)$

[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 0 0 1 0	w w w w w	1	0 1 1	0 1 0 0 0 0	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>

Store Vector (Double-Word)

STV.DW

Store double-word data from vector register

[Instruction format] STV.DW imm1, wreg3, disp16[reg1]

[Operation]

adr ← GR[reg1] + sign-extend (disp16)[Note 1]

CheckException(MAE)

CheckException(MDP)

val ← (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 … 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 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 d d`	`0`	`0`	`0`
disp16			

Where `i` = 1-bit immediate data
`ddddddddddddd` = Higher-order 13 bit 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 double-word data from an arbitrary element of vector register wreg3 is stored in the address 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 ← GR[reg1] + sign-extend (disp16)Note 1

CheckException(MAE)

CheckException(MDP)

Store-memory (adr, WR[wreg3], Quad-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 0	1 1 1 1 0 1	R R R R R	w 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
d d d d d d d d d d d d	0	0	0	0
disp16				

Where dddddddddddd = 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>

Store Vector (Word)

STV.W

Store word data from vector register

[Instruction format] STV.W imm2, wreg3, disp16[reg1]

[Operation]

```
adr ← GR[reg1] + sign-extend (disp16)Note 1
CheckException(MAE)
CheckException(MDP)
val ← (imm2 == 0) ?   WR[wreg3](w0):
          (imm2 == 1) ?   WR[wreg3](w1):
          (imm2 == 2) ?   WR[wreg3](w2):
                          WR[wreg3](w3)
Store-memory (adr, val, 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 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
sub-op		reg1	wreg3	sub-op	

47 34	33	32
d d d d d d d d d d d d d d	0	0
disp16		

Where `ii` = 2-bit immediate data
`dddddddddddd` = Higher-order 14 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 word data from an arbitrary element of vector register wreg3 is stored in the address that is 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.
2. 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>

STVZ.H4

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)Note 1
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 0	1 1 1 1 0 1	R R R R R	w w w w w	0 1 1 1 1 1 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 d d	0	0	0
disp16			

Where ddddddddddddd = Higher-order 13 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 lower-order 16 bits that are taken out of each element of vector register wreg3 are 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 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>

Floating-point SIMD Subtract/Add (single)

SUBADDF.S4

Extended floating-point subtract/add (Single precision)

[Instruction format] SUBADDF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) − WR[wreg1](w3)

WR[wreg3](w2) ← WR[wreg2](w2) + WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) − WR[wreg1](w1)

WR[wreg3](w0) ← 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	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 exceptions]

Unimplemented operation exception (E)

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]

<Even-number element>

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

<Odd-number element>

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

SUBADDNF.S4

Extended floating-point subtract/add negative (Single precision)

[Instruction format] SUBADDNF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(WR[wreg2](w3) − WR[wreg1](w3))

WR[wreg3](w2) ← neg(WR[wreg2](w2) + WR[wreg1](w2))

WR[wreg3](w1) ← neg(WR[wreg2](w1) − WR[wreg1](w1))

WR[wreg3](w0) ← neg(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	1 0 1 1 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 with their sign inverted.

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 even-number elements of vector register wreg1 and stores the results in the even-number elements of vector register wreg3 with their sign inverted.

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)

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]

<Even-number element>

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

<Odd-number element>

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

2.5.4.52 SUBADDNXF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Subtract/Add Negative Exchange (single)

SUBADDNXF.S4

Extended floating-point subtract/add negative exchange (Single precision)

[Instruction format] SUBADDNXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← neg(WR[wreg2](w3) − WR[wreg1](w2))

WR[wreg3](w2) ← neg(WR[wreg2](w2) + WR[wreg1](w3))

WR[wreg3](w1) ← neg(WR[wreg2](w1) − WR[wreg1](w0))

WR[wreg3](w0) ← neg(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 w	1	0 1 1	1 0 1 1 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 with their sign inverted.

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 with their sign inverted.

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)

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]

<Even-number element>

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

<Odd-number element>

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

2.5.4.53 SUBADDXF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Subtract/Add Negative Exchange (single)

SUBADDXF.S4

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

[Instruction format] SUBADDXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) − WR[wreg1](w2)

WR[wreg3](w2) ← WR[wreg2](w2) + WR[wreg1](w3)

WR[wreg3](w1) ← WR[wreg2](w1) − WR[wreg1](w0)

WR[wreg3](w0) ← 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 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 exceptions]

Unimplemented operation exception (E)

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]

<Even-number element>

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

<Odd-number element>

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

2.5.4.54 SUBF.S4

<Extended Floating-point Instructions>

Floating-point SIMD Subtract (single)

SUBF.S4

Extended floating-point subtract (Single precision)

[Instruction format] SUBF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) − WR[wreg1](w3)

WR[wreg3](w2) ← WR[wreg2](w2) − WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) − WR[wreg1](w1)

WR[wreg3](w0) ← 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 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 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.

[Floating-point operation exceptions]

Unimplemented operation exception (E)

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]

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

Floating-point SIMD Subtract Reduction (single)

SUBRF.S4

Extended floating-point subtract reduction (Single precision)

[Instruction format] SUBRF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) − WR[wreg2](w2)

WR[wreg3](w2) ← WR[wreg1](w3) − WR[wreg1](w2)

WR[wreg3](w1) ← WR[wreg2](w1) − WR[wreg2](w0)

WR[wreg3](w0) ← 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 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 floating-point format contents in the even-number elements of vector register wreg2 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 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 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 exceptions]

Unimplemented operation exception (E)

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]

A^{*4} \ B^{*4}	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
+Normal	B − A	B − A	B − A	B − A	+∞	−∞	Q-NaN	Q-NaN[V]
−Normal	B − A	B − A	B − A	B − A	+∞	−∞	Q-NaN	Q-NaN[V]
+0	B − A	B − A	B − A	B − A	+∞	−∞	Q-NaN	Q-NaN[V]
−0	B − A	B − A	B − A	B − A	+∞	−∞	Q-NaN	Q-NaN[V]
+∞	−∞	−∞	−∞	−∞	Q-NaN[V]	−∞	Q-NaN	Q-NaN[V]
−∞	+∞	+∞	+∞	+∞	+∞	Q-NaN[V]	Q-NaN	Q-NaN[V]
Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN	Q-NaN[V]
S-NaN	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	Q-NaN[V]	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>

Floating-point SIMD Subtract Exchange (single)

SUBXF.S4

Extended floating-point subtract exchange (Single precision)

[Instruction format] SUBXF.S4 wreg1, wreg2, wreg3

[Operation]

WR[wreg3](w3) ← WR[wreg2](w3) − WR[wreg1](w2)

WR[wreg3](w2) ← WR[wreg2](w2) − WR[wreg1](w3)

WR[wreg3](w1) ← WR[wreg2](w1) − WR[wreg1](w0)

WR[wreg3](w0) ← 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 w	1	0 1 1	1 0 0 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 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 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)

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]

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

Transfers compare result to PSW

TRFSRV.W4

Vector register flag transfer

[Instruction format] TRFSRV.W4 imm3, wreg2

[Operation]

```
val0 ← WR[wreg2](w0) == 0 ? 0: 1
val1 ← WR[wreg2](w1) == 0 ? 0: 1
val2 ← WR[wreg2](w2) == 0 ? 0: 1
val3 ← WR[wreg2](w3) == 0 ? 0: 1
imm3 = 0: PSW.Z ← val0
imm3 = 1: PSW.Z ← val1
imm3 = 2: PSW.Z ← val2
imm3 = 3: PSW.Z ← val3
imm3 = 4: PSW.Z ← val3 & val2 & val1 & val0
imm3 = 5: PSW.Z ← 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
wreg2		sub-op			category	sub-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 exceptions] None

NOTE

The value of PSW.Z becomes 0 if 6 or 7 is specified in imm3.

2.5.4.58 TRNCF.SUW4

<Extended Floating-point Instructions>

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

TRNCF.SUW4

Extended floating-point type conversion (Single precision → Unsigned integer)

[Instruction format] TRNCF.SUW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← trunc WR[wreg2](w3) (single → unsigned word)

WR[wreg3](w2) ← trunc WR[wreg2](w2) (single → unsigned word)

WR[wreg3](w1) ← trunc WR[wreg2](w1) (single → unsigned word)

WR[wreg3](w0) ← trunc WR[wreg2](w0) (single → unsigned 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 1 1	w w w w w	1	0 1 1	0 1 0 0 0 0	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 32-bit unsigned integer format and stores the results in the corresponding element of vector register wreg3.

The results are 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 FXSR register is set as an invalid operation and no exception occurs. The return value differs according to value of the source as follows:

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

[Floating-point operation exceptions]

Unimplemented operation exception (E)

Invalid operation exception (V)

Inexact exception (I)

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 (Integer)		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>

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

TRNCF.SW4

Extended floating-point type conversion (Single precision → Integer)

[Instruction format] TRNCF.SW4 wreg2, wreg3

[Operation]

WR[wreg3](w3) ← trunc WR[wreg2](w3) (single → word)

WR[wreg3](w2) ← trunc WR[wreg2](w2) (single → word)

WR[wreg3](w1) ← trunc WR[wreg2](w1) (single → word)

WR[wreg3](w0) ← trunc WR[wreg2](w0) (single → 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 1 0	w w w w w	1	0 1 1	0 1 0 0 0 0	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 32-bit integer format and stores the results in the corresponding element of vector register wreg3.

The results are 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 FXSR register is set as an invalid operation and no exception occurs. The return value differs according to value of the source as follows:

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

[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 (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.

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

(1/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Load instruction	LD.B	disp16[reg1],reg2	4	1	1	3*1
	LD.B	disp23[reg1],reg3	6	1	1	3*1
	LD.B	[reg1]+,reg3	4	1	1	3*1, 1*19
	LD.B	[reg1]−,reg3	4	1	1	3*1, 1*19
	LD.BU	disp16[reg1],reg2	4	1	1	3*1
	LD.BU	disp23[reg1],reg3	6	1	1	3*1
	LD.BU	[reg1]+,reg3	4	1	1	3*1, 1*19
	LD.BU	[reg1]−,reg3	4	1	1	3*1, 1*19
	LD.DW	disp23[reg1],reg3	6	1	1	3*1
	LD.H	disp16[reg1],reg2	4	1	1	3*1
	LD.H	disp23[reg1],reg3	6	1	1	3*1
	LD.H	[reg1]+,reg3	4	1	1	3*1, 1*19
	LD.H	[reg1]−,reg3	4	1	1	3*1, 1*19
	LD.HU	disp16[reg1],reg2	4	1	1	3*1
	LD.HU	disp23[reg1],reg3	6	1	1	3*1
	LD.HU	[reg1]+,reg3	4	1	1	3*1, 1*19
	LD.HU	[reg1]−,reg3	4	1	1	3*1, 1*19
	LD.W	disp16[reg1],reg2	4	1	1	3*1
	LD.W	disp23[reg1],reg3	6	1	1	3*1
	LD.W	[reg1]+,reg3	4	1	1	3*1, 1*19
	LD.W	[reg1]−,reg3	4	1	1	3*1, 1*19
	SLD.B	disp7[ep],reg2	2	1	1	3*1
	SLD.BU	disp4[ep],reg2	2	1	1	3*1
	SLD.H	disp8[ep],reg2	2	1	1	3*1
	SLD.HU	disp5[ep],reg2	2	1	1	3*1
	SLD.W	disp8[ep],reg2	2	1	1	3*1

(2/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Store instruction	ST.B	reg2,disp16[reg1]	4	1	1	1
	ST.B	reg3,disp23[reg1]	6	1	1	1
	ST.B	reg3,[reg1]+	4	1	1	1, 1[*19]
	ST.B	reg3,[reg1]−	4	1	1	1, 1[*19]
	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[*19]
	ST.H	reg3,[reg1]−	4	1	1	1, 1[*19]
	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[*19]
	ST.W	reg3,[reg1]−	4	1	1	1, 1[*19]
	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 instruction	CLR1	bit#3,disp16[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	CLR1	reg2,[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	NOT1	bit#3,disp16[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	NOT1	reg2,[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	SET1	bit#3,disp16[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	SET1	reg2,[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	TST1	bit#3,disp16[reg1]	4	1[*22]	1[*22]	8[*2,*23]
	TST1	reg2,[reg1]	4	1[*22]	1[*22]	8[*2,*23]
Special instruction	CAXI	[reg1],reg2,reg3	4	10[*24]	10[*24]	8[*2,*17], 10[*2,*17]
	DISPOSE[*4]	imm5,list12	4	N+1[*3]	N+2[*3]	N+1[*3]
	LDL.BU	[reg1],reg3	4	12[*24]	12[*24]	9[*2,*17], 12[*2,*17]
	LDL.HU	[reg1],reg3	4	12[*24]	12[*24]	9[*2,*17], 12[*2,*17]
	LDL.W	[reg1],reg3	4	12[*24]	12[*24]	9[*2,*17], 12[*2,*17]
	POPSP	rh-rt	4	N+1[*5]	N+2[*5]	N+1[*5]
	LDM.MP	[reg1], eh-et	4	N+8[*14,*25]	N+8[*14,*25]	N+8[*14,*25]
	PREPARE	list12,imm5	4	N+1[*3]	N+2[*3]	N+1[*3]
	PREPARE	list12,imm5,sp	4	N+2[*3]	N+3[*3]	N+2[*3]
	PREPARE	list12,imm5,imm16	6	N+2[*3]	N+3[*3]	N+2[*3]
	PREPARE	list12,imm5,imm16<<16	6	N+2[*3]	N+3[*3]	N+2[*3]
	PREPARE	list12,imm5,imm32	8	N+2[*3]	N+3[*3]	N+2[*3]
	PUSHSP	rh-rt	4	N+1[*5]	N+2[*5]	N+1[*5]
	RESBANK		4	25, 30[*20]	25, 30[*20]	25, 30[*20]
	STC.B	reg3,[reg1]	4	8, 12[*18,*24]	8, 12[*18,*24]	6, 10[*18]
	STC.H	reg3,[reg1]	4	8, 12[*18,*24]	8, 12[*18,*24]	6, 10[*18]

(3/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Special instruction	STC.W	reg3,[reg1]	4	8, 12[*18,*24]	8, 12[*18,*24]	6, 10[*18]
	STM.MP	eh-et, [reg1]	4	N+2[*14,*25]	N+2[*14,*25]	N+2[*14,*25]
Multiplication instruction	MUL	reg1,reg2,reg3	4	1	1	3
	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-accumulate operation instruction	MAC	reg1,reg2,reg3,reg4	4	2	2	4
	MACU	reg1,reg2,reg3,reg4	4	2	2	4
Arithmetic instruction	ADD	reg1,reg2	2	1	1	1
	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 operation instruction	ADF	cccc,reg1,reg2,reg3	4	1	1	1
	SBF	cccc,reg1,reg2,reg3	4	1	1	1
Saturated operation instruction	SATADD	reg1,reg2	2	1	1	1
	SATADD	imm5,reg2	2	1	1	1
	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 instruction	AND	reg1,reg2	2	1	1	1
	ANDI	imm16,reg1,reg2	4	1	1	1
	NOT	reg1,reg2	2	1	1	1
	OR	reg1,reg2	2	1	1	1
	ORI	imm16,reg1,reg2	4	1	1	1
	TST	reg1,reg2	2	1	1	1
	XOR	reg1,reg2	2	1	1	1
	XORI	imm16,reg1,reg2	4	1	1	1

(4/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Data operation instruction	BINS	reg1,pos,width,reg2	4	1	1	1
	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 instruction	SCH0L	reg2,reg3	4	1	1	1
	SCH0R	reg2,reg3	4	1	1	1
	SCH1L	reg2,reg3	4	1	1	1
	SCH1R	reg2,reg3	4	1	1	1
Division instruction	DIV	reg1,reg2,reg3	4	1	19	19
	DIVH	reg1,reg2	2	1	19	19
	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 instruction	DIVQ	reg1,reg2,reg3	4	1	N+3*6	N+3*6
	DIVQU	reg1,reg2,reg3	4	1	N+3*6	N+3*6

(5/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Branch instruction*7	Bcond	disp9	2	2/6*8	2/6*8	2/6*8
	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 instruction (with branching)	CALLT	imm6	2	17	17	17
	CTRET	—	4	8	8	8
	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*8	11, 18*8	11, 18*8
	SYSCALL	vector8	4	17	17	17
Special instruction	DI	—	4	2	2	2
	EI	—	4	2	2	2
	HALT	—	4	*21	*21	*21
	LDSR	reg2,regID,selID	4	1*10	1*10	1
	NOP	—	2	1	1	1
	SNOOZE	—	4	*11	*11	*11
	STSR	regID,reg2,selID	4	1*10	1*10	3
	SYNCE	—	2	1	1	1
	SYNCI	—	2	*12	*12	*12
	SYNCM	—	2	*13	*13	*13
	SYNCP	—	2	*14	*14	*14
Cache instruction	CACHE	cacheop,[reg1]	4	*15	*15	*15
	PREF	prefop,[reg1]	4	*15	*15	*15
Floating-point arithmetic operation (single precision)	ABSF.S	reg2,reg3	4	1	1	4
	ADDF.S	reg1,reg2,reg3	4	1	1	4
	CEILF.SL	reg2,reg3	4	1	1	4
	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

(6/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Floating-point arithmetic operation (single precision)	CVTF.HS	reg2,reg3	4	1	1	4
	CVTF.LS	reg2,reg3	4	1	1	4
	CVTF.SH	reg2,reg3	4	1	1	4
	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*16	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*16	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*16	21	24
	SQRTF.S	reg2,reg3	4	14*16	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

(7/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Floating-point arithmetic operation (double precision)	ABSF.D	reg2,reg3	4	1	1	4
	ADDF.D	reg1,reg2,reg3	4	1	1	4
	CEILF.DL	reg2,reg3	4	1	1	4
	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*16	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*16	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*16	45	48
	SQRTF.D	reg2,reg3	4	30*16	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

(8/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Extended floating-point arithmetic operation	ABSF.S4	wreg2,wreg3	4	1	1	1
	ADDF.S4	wreg1,wreg2,wreg3	4	2	2	5
	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

(9/9)

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Extended floating-point arithmetic operation	SHFLV.W4	imm12,wreg1,wreg2,wreg3	6	1	1	1
	SQRTF.S4	wreg2,wreg3	4	28	28	31
	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. SelID = 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

Renesas Electronics Corporation

R01US0209EJ0220