User's Manual

32

RH850G3M

User's Manual: Software

Renesas microcontroller
RH850 Family

Renesas Electronics
www.renesas.com

Rev.1.40 Dec, 2016

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

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

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

NOTES FOR CMOS DEVICES

(1) VOLTAGE APPLICATION WAVEFORM AT INPUT PIN: Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (MAX) and VIH (MIN) due to noise, etc., the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (MAX) and VIH (MIN).

(2) HANDLING OF UNUSED INPUT PINS: Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

(3) PRECAUTION AGAINST ESD: A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

(4) STATUS BEFORE INITIALIZATION: Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

(5) POWER ON/OFF SEQUENCE: In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current. The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

(6) INPUT OF SIGNAL DURING POWER OFF STATE : Do not input signals or an I/O pull-up power supply while the device is not powered. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

How to Use This Manual

Target and Readers

This manual is intended for users who wish to understand the RH850G3M software and design application systems using these products.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{xxx}$ (overscore over pin or signal name)
Memory map address:	Higher addresses on the top and lower addresses on the bottom
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary ... xxxx or xxxxB
	Decimal ... xxxx
	Hexadecimal ... xxxxH
Prefix indicating power of 2 (address space, memory capacity):	
	K (kilo): 2^{10} = 1,024
	M (mega): 2^{20} = $1,024^2$
	G (giga): 2^{30} = $1,024^3$

Table of Contents

CHAPTER 1 OVERVIEW

1.1 Features of the RH850G3M

The RH850G3M features backward compatibility with the instruction set for the 32-bit RISC microcontroller V850 Series.

The RH850G3M provides basic functions for multi-core systems, such as exclusive control between multiple cores.

Table 1-1 shows the features of the RH850G3M.

Table 1-1 Features of the RH850G3M

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, and V850E2) Long/short type load/store instructions Three-operand instructions Instruction set based on C • CPU operating modes User mode and supervisor mode • Address space: 4-Gbyte linear space for both data and instructions
Coprocessor	• A floating point operation coprocessor (FPU) can be installed. Supports single precision (32-bit) and double precision (64-bit) Supports IEEE754-compliant data types and exceptions Rounding modes: Nearest, 0 direction, +∞ direction, and –∞ direction Handling on non-normalized numbers: These are truncated to 0, or an exception is reported because such numbers do not comply with IEEE754.
Exceptions/interrupts	• Number of scalable interrupt channels • 16-level interrupt priority that can be specified for each channel • Vector selection method that can be selected according to performance requirements and the amount of consumed memory Direct branch method exception vector (direct vector method) Address-table-referencing indirect branch method exception vector (table reference method) • Support for high-speed context backup and restoration processing on interrupt by using dedicated instructions (PUSHSP, POPSP)
Memory management	• A memory protection unit (MPU) can be installed.
Caches	• An instruction cache can be installed.

1.2 Changes from the V850E2M

Table 1-2 Changes from the V850E2M

Item	Changes
CPU	• Defined CPU operating modes • Changed the system registers from the bank selection method to the group specification method • Increased the instruction addressing range to 4 G • Introduced CPU operating modes User mode and supervisor mode • Introduced a CPU virtualization support function • Added new instructions LDL.W, STC.W, BINS, ROTL, LD.DW, ST.DW • High-functional branch instructions Bcond disp17, JARL[reg1], reg3, LOOP reg1, disp16
Exceptions/interrupts	• Changed the vector selection method • Added new exceptions
Coprocessor	• Floating-point operation function coprocessor (FPU) Added IEEE754-2008-compliant FPU version specifications
Memory management	• Permitted requests to access the memory when a memory protection violation is detected

CHAPTER 2 PROCESSOR MODEL

This CPU defines a processor model that has basic operation functions, registers, and an exception management function. This section describes the unique features of the processor model of this CPU.

2.1 CPU Operating Modes

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

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

2.1.1 Definition of CPU Operating Modes

(1) Supervisor mode (SV)

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

(2) User mode (UM)

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

Restriction in user mode (PSW.UM = 1)

- Privileged instruction violations due to SV-privileged-instruction operating restrictions (→ PIE exceptions)

For details about privileged-instruction operating restrictions, see **2.1.3 CPU Operating Modes and Privileges**.

2.1.2 CPU Operating Mode Transition

The CPU operating mode changes due to three events.

(1) Change due to acknowledging an exception

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

(2) Change due to a return instruction

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

(3) Change due to a system register instruction

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

Cautions 1. In supervisor mode, the LDSR instruction can be used to directly change the value of the PSW.UM bit, but system-register-related hazards are defined in the hardware specifications. For the change of this bit, it is recommended to use a return instruction to avoid PSW-register-related hazards.

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

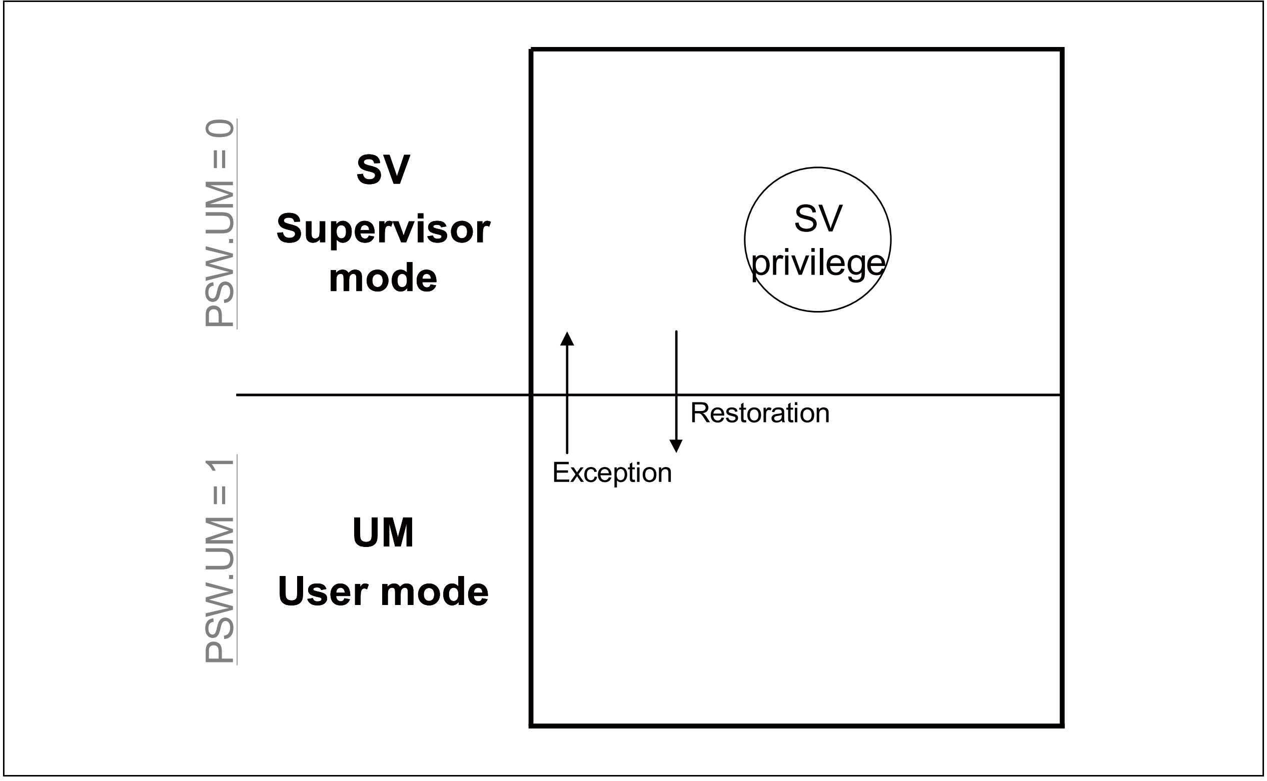
2.1.3 CPU Operating Modes and Privileges

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

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

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

Figure 2-1 CPU Operating Modes and Privileges



(1) Supervisor privilege (SV privilege)

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

(2) Coprocessor use permissions

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

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

(3) Operation when there is a privilege violation

When an attempt is made to execute a privileged instruction by someone who does not have the required privilege, a PIE exception or UCPOP exception occurs. **Table 2-1** shows the relationships between the operating mode, usage permission status, and whether instructions can be executed.

Table 2-1 Operation When There is a Privilege Violation

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

Note This includes the LDSR/STSR instruction for the coprocessor system register.

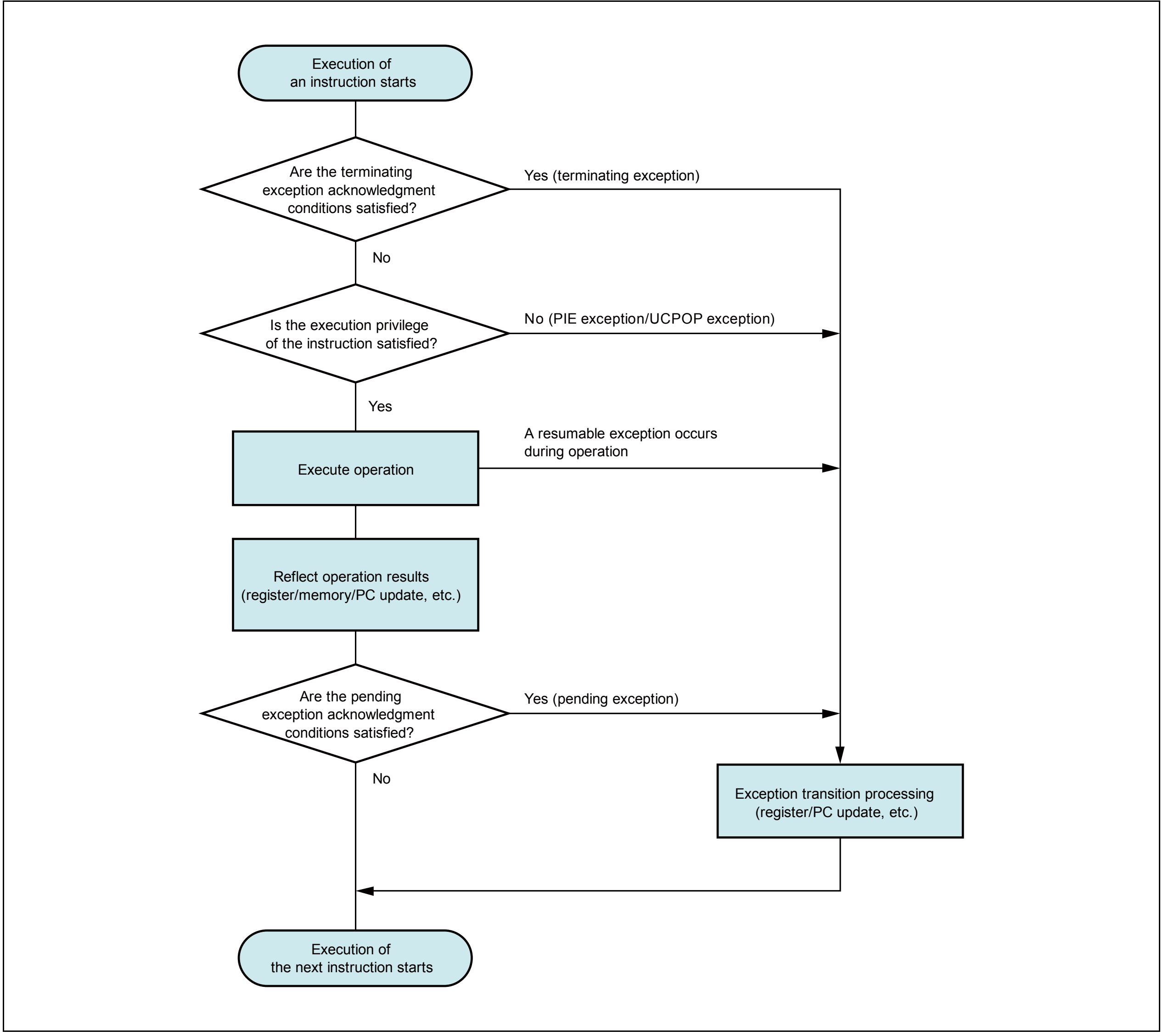
Remark —: 0 or 1

Caution If a register whose access permission is defined as CU*n* or SV is accessed when CU*n* = 0 and UM = 0, a UCPOP exception occurs.

2.2 Instruction Execution

The instruction execution flow of this CPU is shown below.

Figure 2-2 Instruction Execution Flow


Execution of an instruction starts
Are the terminating exception acknowledgment conditions satisfied?
Yes (terminating exception)
No
Is the execution privilege of the instruction satisfied?
No (PIE exception/UCPOP exception)
Yes
Execute operation
A resumable exception occurs during operation
Reflect operation results (register/memory/PC update, etc.)
Are the pending exception acknowledgment conditions satisfied?
Yes (pending exception)
No
Exception transition processing (register/PC update, etc.)
Execution of the next instruction starts

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

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

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

2.3 Exceptions and Interrupts

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

For details, see **CHAPTER 4 EXCEPTIONS AND INTERRUPTS**.

2.3.1 Types of Exceptions

This CPU divides exceptions into the following three types according to their purpose.

- Terminating exceptions
- Resumable exceptions
- Pending exceptions

(1) Terminating exceptions

This is an exception acknowledged by interrupting an instruction before its operation is executed. These exceptions include interrupts and imprecise exceptions.

Interrupts are generated by causes such as an interrupt or a hardware error and start up a program that is unrelated to the program currently executing. Imprecise exceptions are caused by instruction operation, but they do not start executing until the current instruction execution finishes; instead, they start executing during execution of the subsequent instruction.

(2) Resumable exceptions

This is an exception acknowledged during the execution of instruction operation before the execution is finished. Because this kind of an exception is correctly acknowledged without executing the next instruction, it is also called a precise exception.

Unlike terminating-type imprecise exceptions, precise exceptions occur during instruction execution and cause the execution of the instruction to stop. It is therefore possible to resume execution of the same instruction after the exception has been processed. By specifying settings appropriate for the exception handling by using a memory management or other function before resuming execution of the same instruction, complex memory management can be achieved while retaining consistency in the logical behavior of the program.

(3) Pending exceptions

This is an exception acknowledged after the execution of an instruction finishes as a result of executing the instruction operation. Pending exceptions include software exceptions.

Because pending exceptions are defined to occur as part of the normal operation of an instruction, unlike resumable-type exceptions, the instruction that caused the exception finishes normally and is not re-executed. These exceptions are mainly used as call gates for calls made by the management program.

2.3.2 Exception Level

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

- EI level exception
- FE level exception

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

2.4 Coprocessors

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

2.4.1 Coprocessor Use Permissions

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

2.4.2 Correspondences between Coprocessor Use Permissions and Coprocessors

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

Table 2-2 Correspondences Between Coprocessor Use Permissions and Coprocessors

Coprocessor Use Permission	Coprocessor Function	Exception Cause Code
CU0	Single-precision FPU expansion function	80H
	Double-precision FPU expansion function	
CU1	Reserved	81H
CU2	Reserved	82H

2.4.3 Coprocessor Unusable Exceptions

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

2.4.4 System Registers

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

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

2.5 Registers

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

2.5.1 Program Registers

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

Table 2-3 Program Registers

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

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

2.5.2 System Registers

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

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

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

2.5.3 Register Updating

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

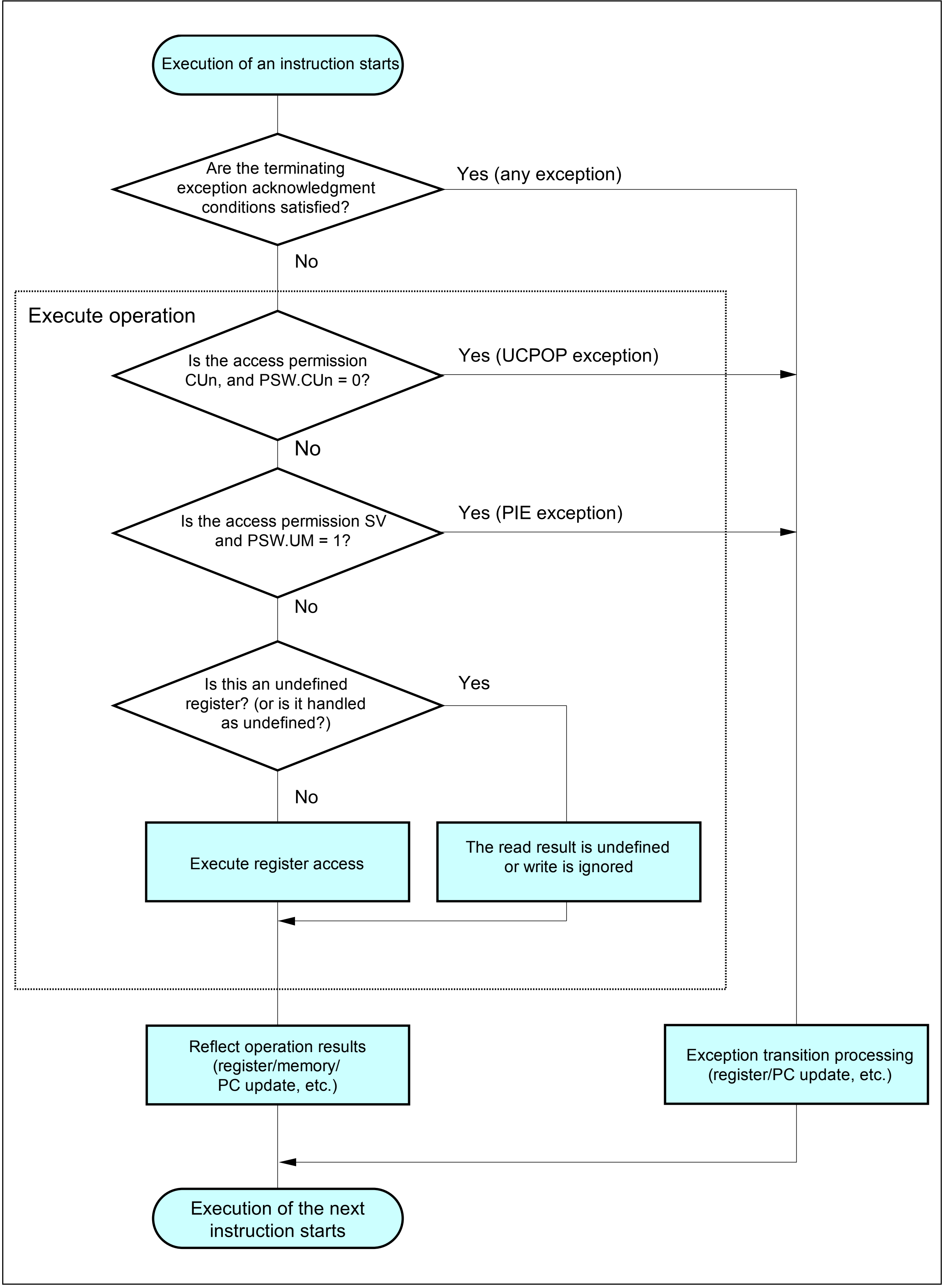
- LDSR
- STSR

(1) LDSR and STSR

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

Figure 2-3 shows the flow of executing the LDSR and STSR instructions.

Figure 2-3 Flow of Executing the LDSR and STSR Instructions


Execution of an instruction starts
Are the terminating exception acknowledgment conditions satisfied?
Yes (any exception)
No
Execute operation
Is the access permission CUn, and PSW.CUn = 0?
Yes (UCPOP exception)
No
Is the access permission SV and PSW.UM = 1?
Yes (PIE exception)
No
Is this an undefined register? (or is it handled as undefined?)
Yes
No
Execute register access
The read result is undefined or write is ignored
Reflect operation results (register/memory/ PC update, etc.)
Exception transition processing (register/PC update, etc.)
Execution of the next instruction starts

2.5.4 Accessing Undefined Registers

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

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

2.6 Data Types

2.6.1 Data formats

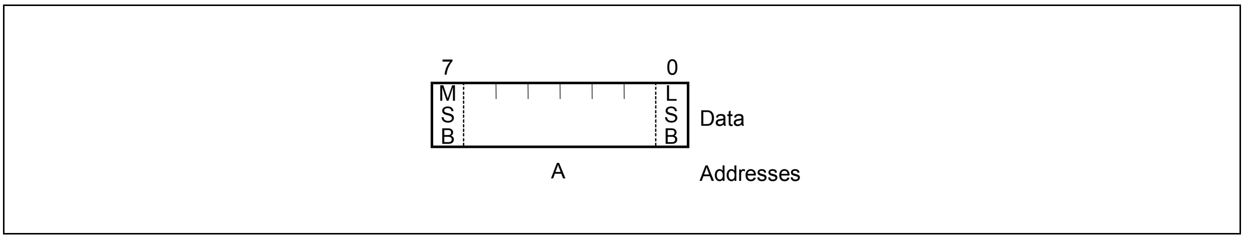
This CPU handles data in little endian format. This means that byte 0 of a halfword or a word is always the least significant (rightmost) byte.

The supported data format is as follows.

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

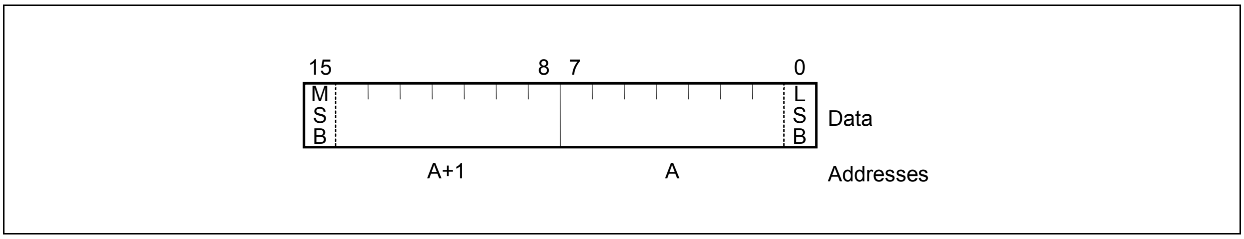
(1) Byte

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



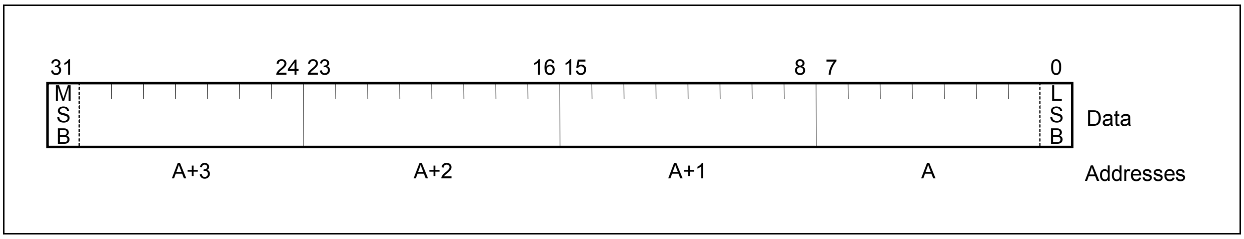
(2) Halfword

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



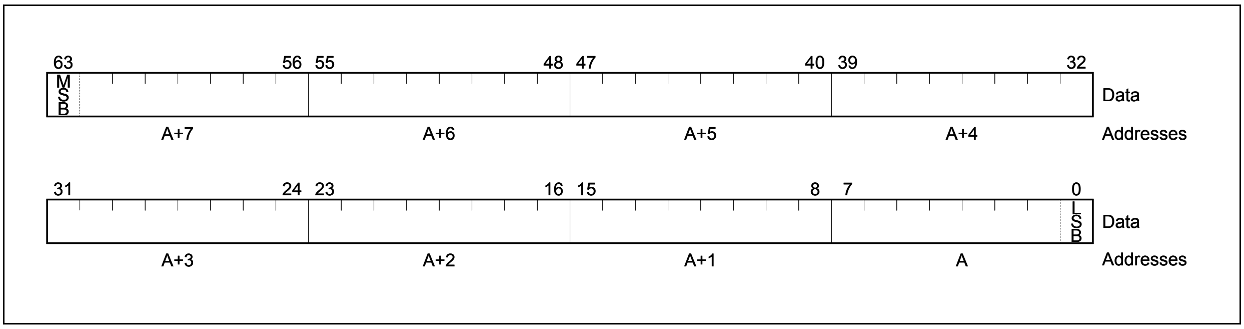
(3) Word

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



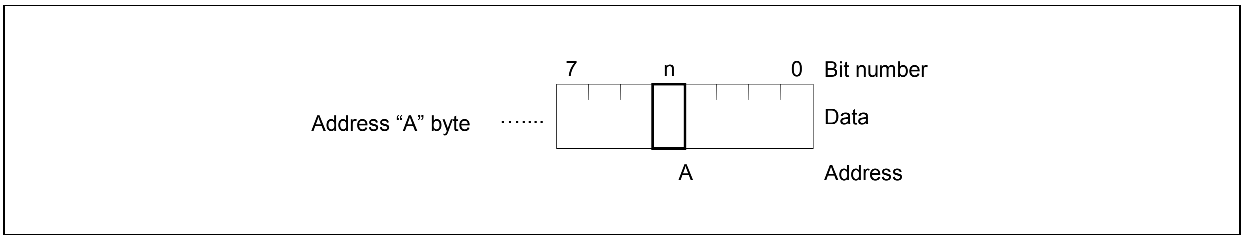
(4) Double-word

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



(5) Bit

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



2.6.2 Data Representation

(1) Integers

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

The integer ranges for various data lengths are as follows.

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

(2) Unsigned integers

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

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

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

(3) Bits

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

- Set
- Clear
- Invert
- Test

2.6.3 Data Alignment

The behavior when the result of address calculation is a misaligned address can be selected by using the MCTL.MA bit. If the MCTL.MA bit has been cleared to 0, a misaligned access exception (MAE) occurs. If the MCTL.MA bit has been set to 1, a misaligned access exception (MAE) does not occur and accessing the address is possible.

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

In addition, even if the MCTL.MA bit is set (1), the double-word access to an address other than the word boundary causes misaligned exception.

Cautions 1. The following instructions might possibly cause misaligned access. For details, see the relevant descriptions in CHAPTER 7 INSTRUCTION.

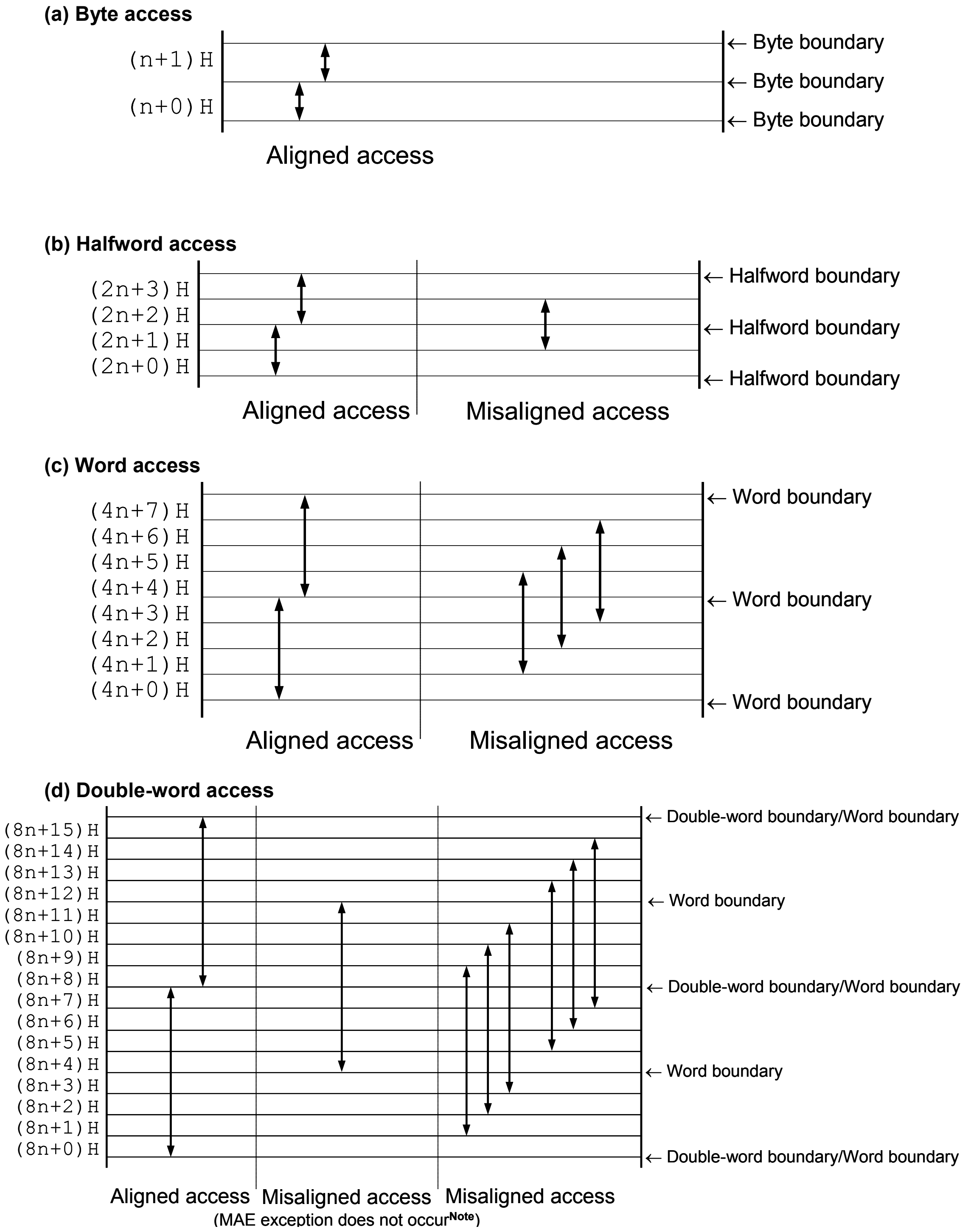
- **LD.H, LD.HU, LD.W, LD.DW**
- **SLD.H, SLD.HU, SLD.W**
- **ST.H, ST.W, ST.DW**
- **SST.H, SST.W**
- **LDL.W, STC.W, CAXI**

2. The following instructions do not cause misaligned access, because the address is rounded in the instruction specification when the alignment specification is incorrect.

- **PREPARE, DISPOSE**
- **PUSHSP, POPSP**

3. For some hardware specifications, a misaligned access exception does not occur and the hardware performs proper misaligned access. For details, see the hardware manual of the product used.

Figure 2-4 Example of Data Placement for Misaligned Access



Note For details, see LD.DW and ST.DW in **CHAPTER 7 INSTRUCTION**.

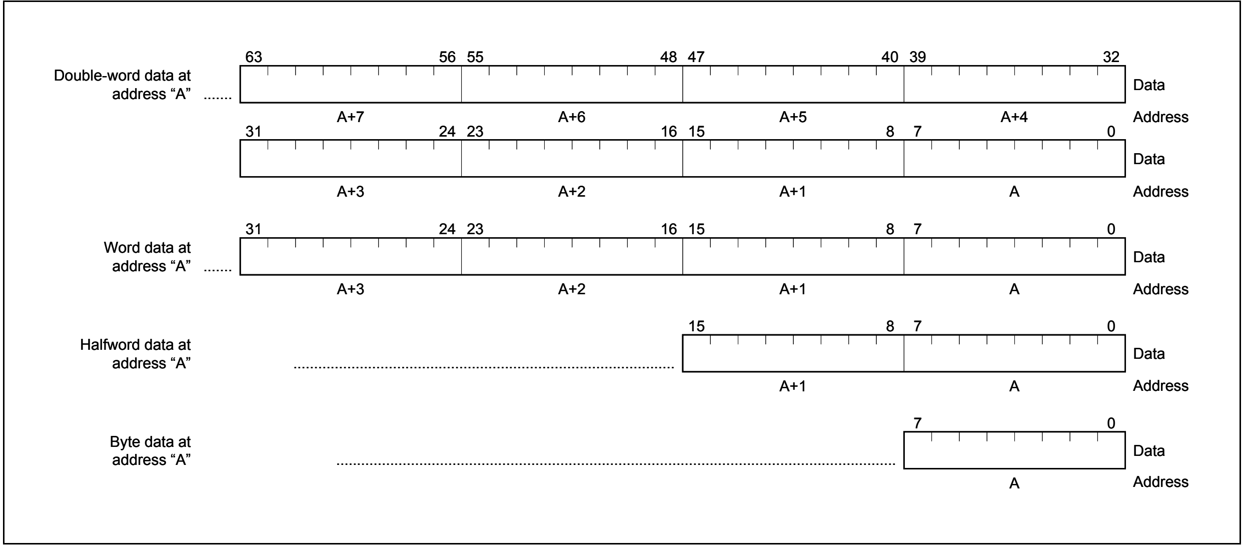
2.7 Address Space

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

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

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

Figure 2-5 Address Space Byte Format

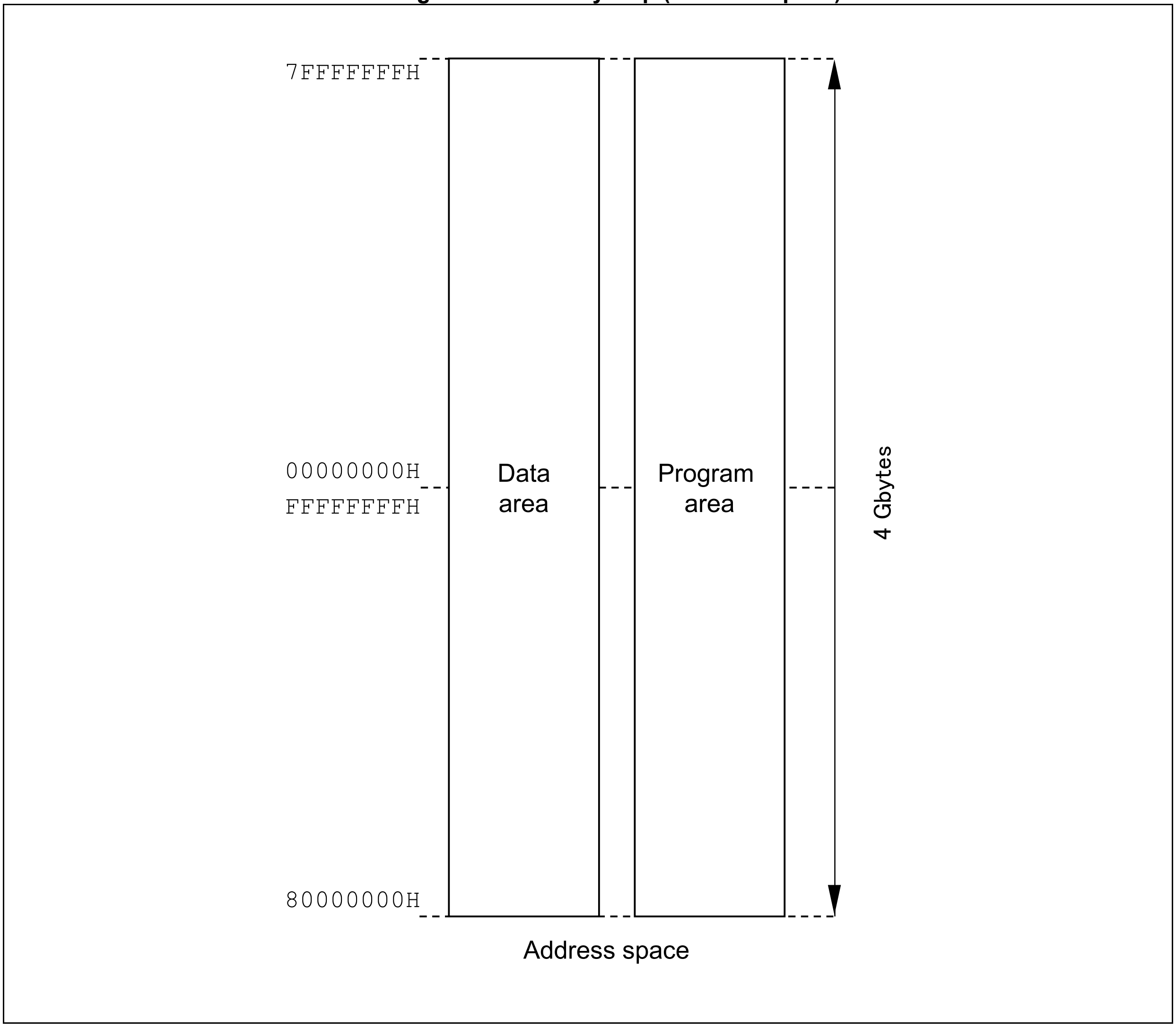


2.7.1 Memory Map

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

A memory map is shown in **Figure 2-6**.

Figure 2-6 Memory Map (Address Space)



2.7.2 Instruction Addressing

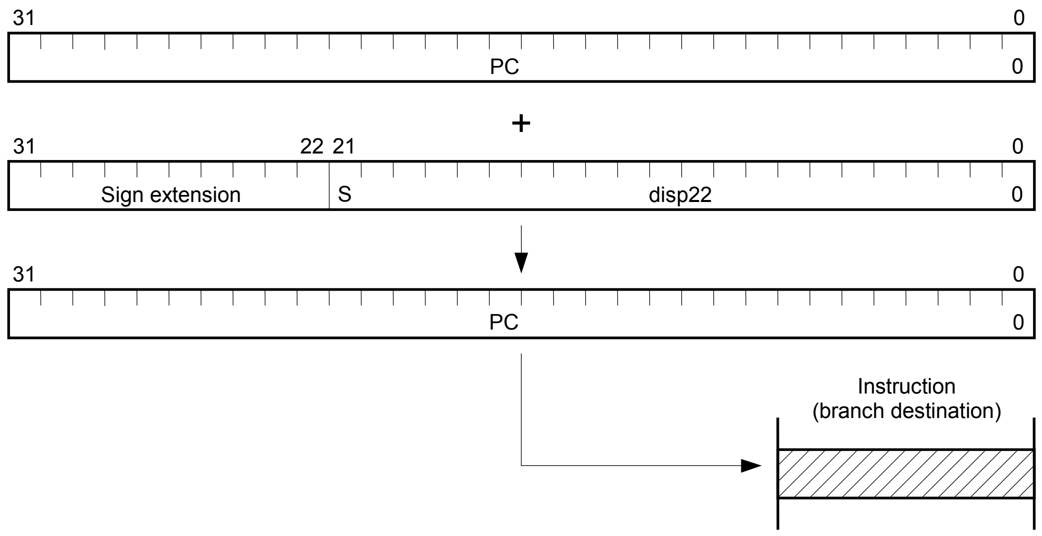
The instruction address is determined based on the contents of the program counter (PC), and is automatically incremented according to the number of bytes in the executed instruction. When a branch instruction is executed, the addressing shown below is used to set the branch destination address to the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

(1) Relative addressing (PC relative)

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

The JARL, JR, and Bcond instructions are used with this type of addressing.

Figure 2-7 Relative Addressing



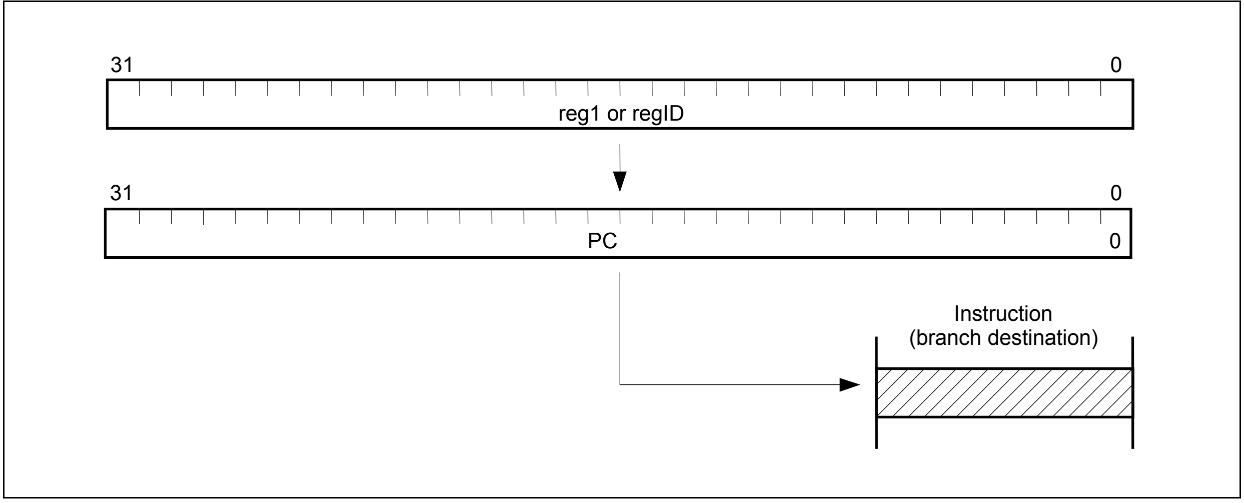
Remark This is an example of 22-bit displacement.

(2) Register addressing (register indirect)

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

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

Figure 2-8 Register Addressing

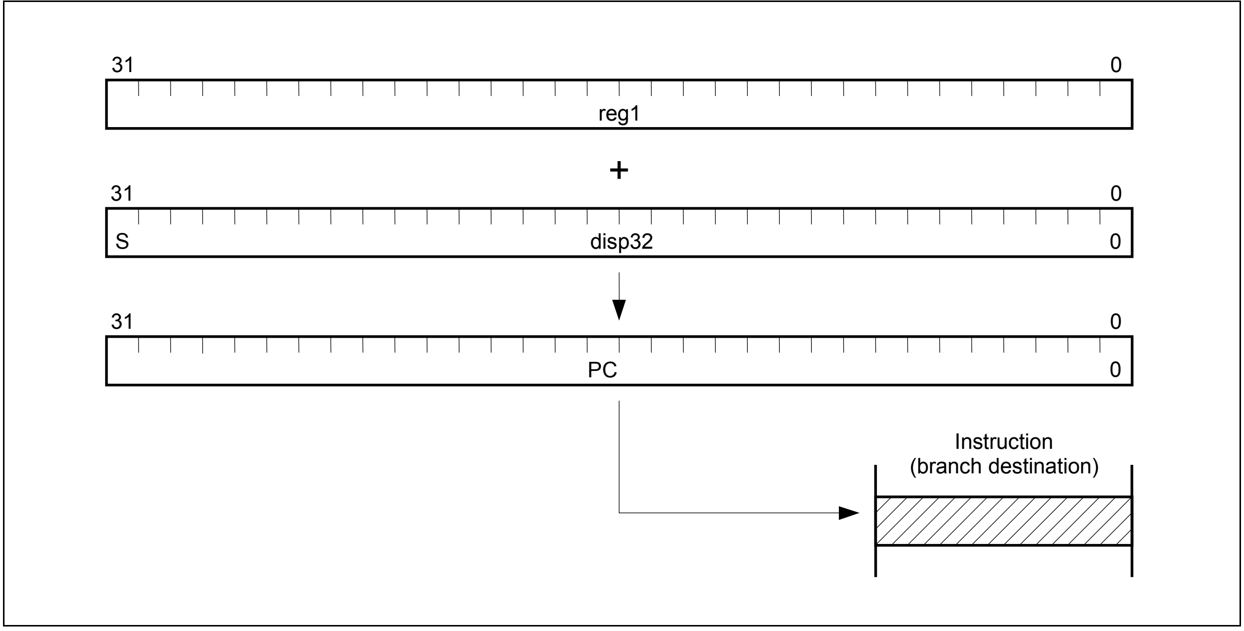


(3) Based addressing

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

The JMP instruction is used with this type of addressing.

Figure 2-9 Based Addressing



(4) Other addressing

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

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

2.7.3 Data Addressing

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

(1) Register addressing

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

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

(2) Immediate addressing

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

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

Remark vector: This is immediate data that specifies the exception vector (00H to 1FH), and is an operand used by the TRAP, FETRAP, and SYSCALL instructions. The data width differs from one instruction to another.

cccc: This is 4-bit data that specifies a condition code, and is an operand used in the CMOV instruction, SASF instruction, and SETF instruction. One bit (0) is added to the higher position and is then assigned to an opcode as a 5-bit immediate data.

(3) Based addressing

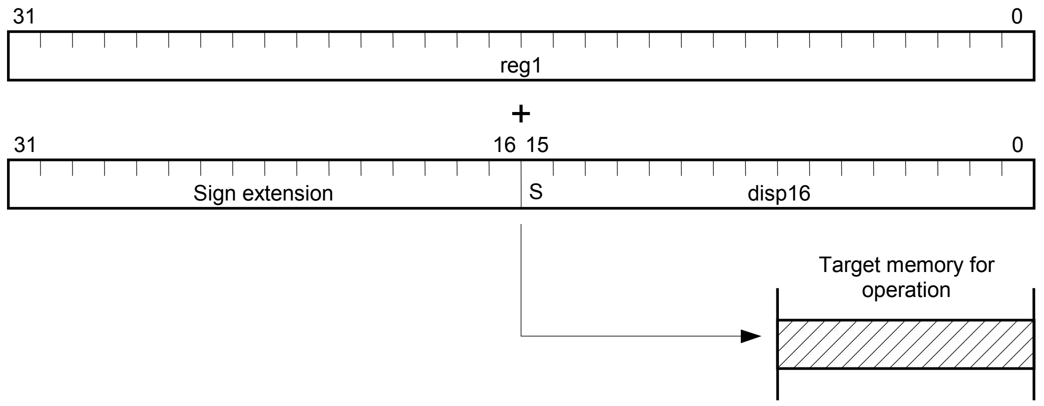
There are two types of based addressing, as described below.

(a) Type 1

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

The LD, ST, and CAXI instructions are used with this type of addressing.

Figure 2-10 Based Addressing (Type 1)



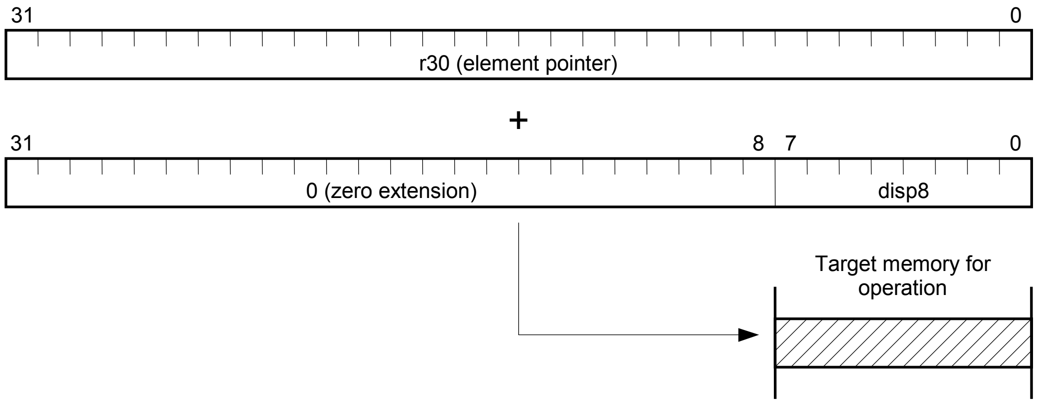
Remark This is an example of 16-bit displacement.

(b) Type 2

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

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

Figure 2-11 Based Addressing (Type 2)



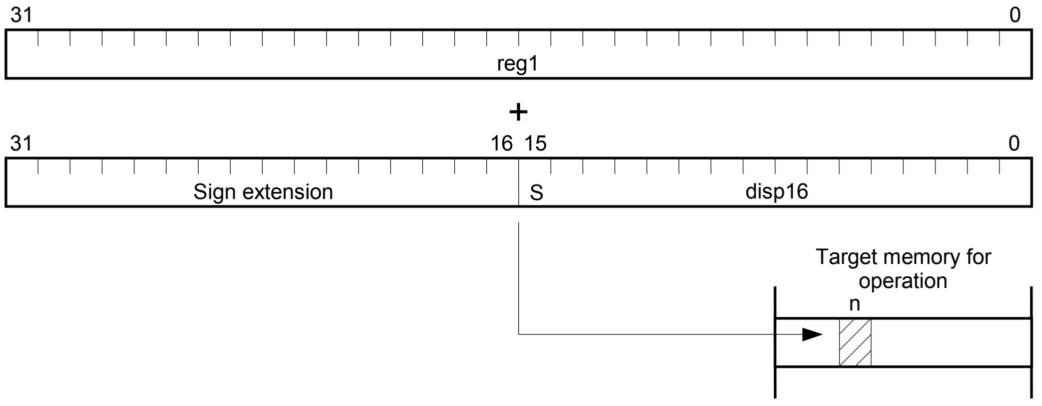
Remark This is an example of 8-bit displacement.

(4) Bit addressing

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

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

Figure 2-12 Bit Addressing



Remark n: Bit position specified by 3-bit data (bit #3) (n = 0 to 7)

This is an example of 16-bit displacement.

(5) Post index increment/decrement addressing

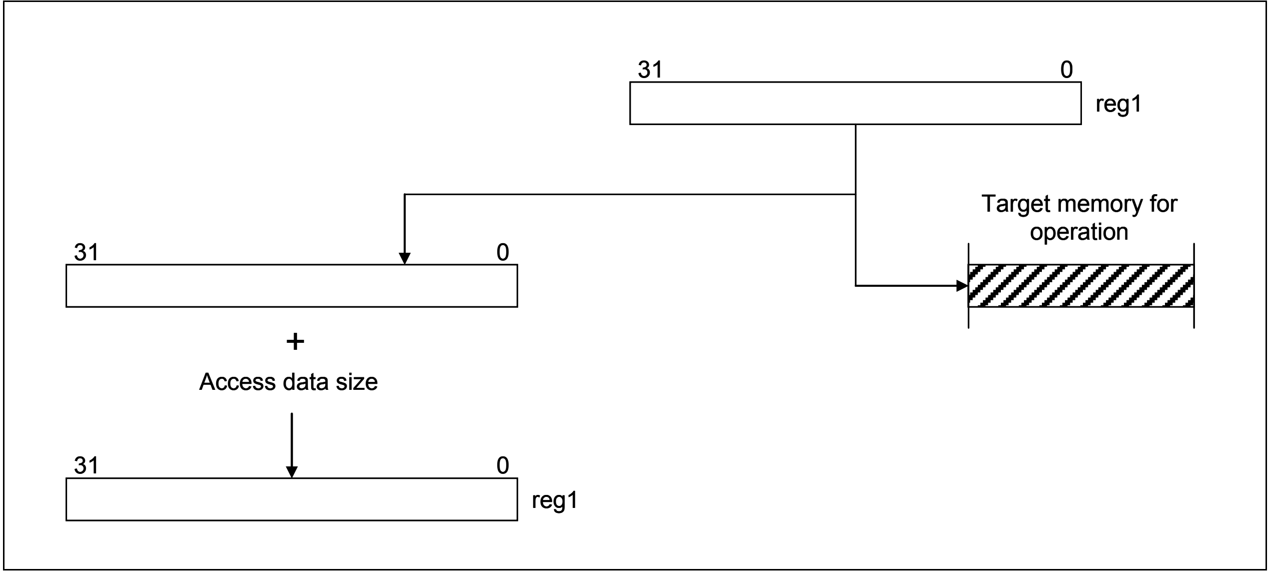
The contents of the general-purpose register (reg1) are used as an operand address to access the target memory, and then the general-purpose register (reg1) is updated. The register is updated by either incrementing or decrementing it, and there are three types (1 to 3).

If the result of incrementing the general-purpose register (reg1) value exceeds the positive maximum value 0xFFFFFFFF, the result wraps around to 0x00000000, and, if the result of decrementing the general-purpose register value is less than the positive minimum value 0x00000000, the result wraps around to 0xFFFFFFFF.

(a) Type 1

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

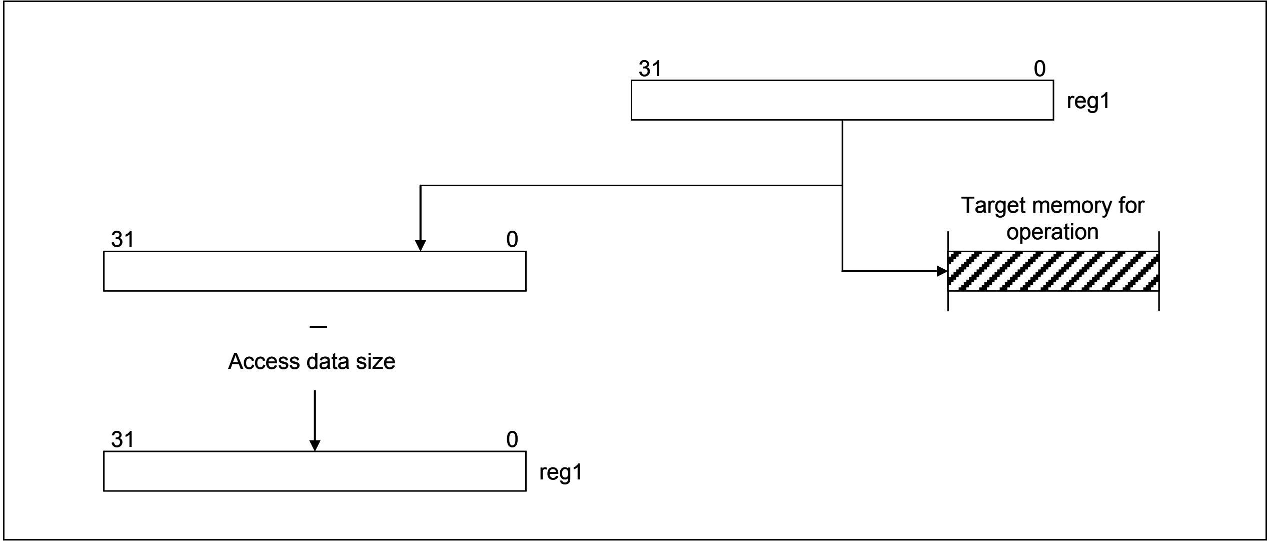
Figure 2-13 Post Index Increment/Decrement Addressing (Type 1)



(b) Type 2

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

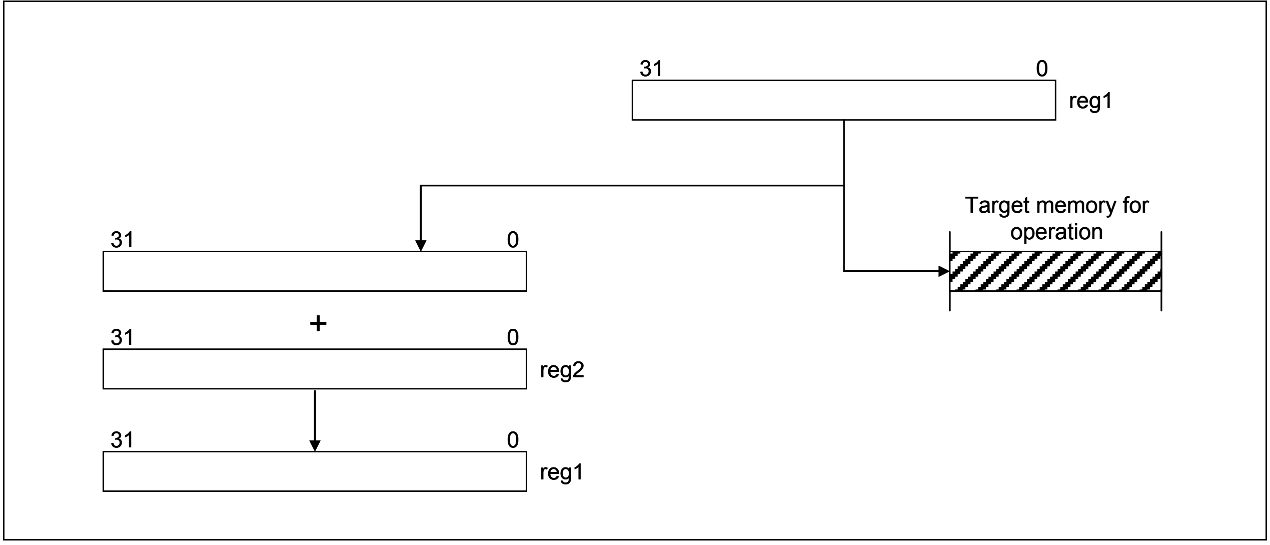
Figure 2-14 Post Index Increment/Decrement Addressing (Type 2)



(c) Type 3

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

Figure 2-15 Post Index Increment/Decrement Addressing (Type 3)



(6) Other addressing

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

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

2.8 Acquiring the CPU Number

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

In the multi-processor configuration, you can identify which CPU core is running a program by referencing HTCFG0.PEID.

With HTCFG0.PEID, unique numbers are assigned within multi-processor systems.

2.9 System Protection Identifier

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

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

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

CHAPTER 3 REGISTER SET

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

3.1 Program Registers

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

Table 3-1 Program Registers

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

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

3.1.1 General-Purpose Registers

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

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

(a) r0, r3, and r30

These registers are implicitly used by instructions.

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

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

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

(b) r1, r4, r5, and r31

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

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

(c) r2

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

3.1.2 PC — Program Counter

The PC retains the address of the instruction being executed.

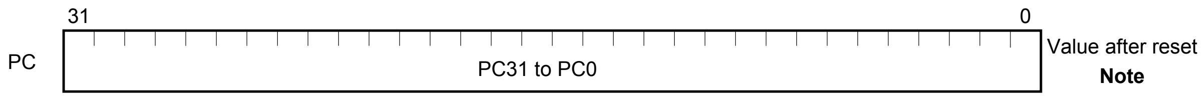


Table 3-2 PC Register Contents

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

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

3.2 Basic System Registers

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

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

Table 3-3 Basic System Registers (1/2)

Register No. (regID, selID)	Symbol	Function	Access Permission
SR0, 0	EIPC	Status save registers when acknowledging EI level exception	SV
SR1, 0	EIPSW	Status save registers when acknowledging EI level exception	SV
SR2, 0	FEPC	Status save registers when acknowledging FE level exception	SV
SR3, 0	FEPSW	Status save registers when acknowledging FE level exception	SV
SR5, 0	PSW	Program status word	**Note 1**
SR6, 0	FPSR	(See **3.4 FPU Function Registers**)	CU0 and SV
SR7, 0	FPEPC	(See **3.4 FPU Function Registers**)	CU0 and SV
SR8, 0	FPST	(See **3.4 FPU Function Registers**)	CU0
SR9, 0	FPCC	(See **3.4 FPU Function Registers**)	CU0
SR10, 0	FPCFG	(See **3.4 FPU Function Registers**)	CU0
SR11, 0	FPEC	(See **3.4 FPU Function Registers**)	CU0 and SV
SR13, 0	EIIC	EI level exception cause	SV
SR14, 0	FEIC	FE level exception cause	SV
SR16, 0	CTPC	CALLT execution status save register	UM
SR17, 0	CTPSW	CALLT execution status save register	UM
SR20, 0	CTBP	CALLT base pointer	UM
SR28, 0	EIWR	EI level exception working register	SV
SR29, 0	FEWR	FE level exception working register	SV
SR31, 0	(BSEL)	(Reserved for backward compatibility with V850E2 series)[Note 2]	SV

Notes 1. The access permission differs depending on the bit. For details, see **(5) PSW — Program status word** in **3.2.1 Basic Registers**.

2. This bit is reserved to maintain backward compatibility with V850E2 series. This bit is always 0 when read. Writing to this bit is ignored.

Table 3-3 Basic System Registers (2/2)

Register No. (regID, selID)	Symbol	Function	Access Permission
SR0, 1	MCFG0	Machine configuration	SV
SR2, 1	RBASE	Reset vector base address	SV
SR3, 1	EBASE	Exception handler vector address	SV
SR4, 1	INTBP	Base address of the interrupt handler table	SV
SR5, 1	MCTL	CPU control	SV
SR6, 1	PID	Processor ID	SV
SR11, 1	SCCFG	SYSCALL operation setting	SV
SR12, 1	SCBP	SYSCALL base pointer	SV
SR0, 2	HTCFG0	Thread configuration	SV
SR6, 2	MEA	Memory error address	SV
SR7, 2	ASID	Address space ID	SV
SR8, 2	MEI	Memory error information	SV

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

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

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

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

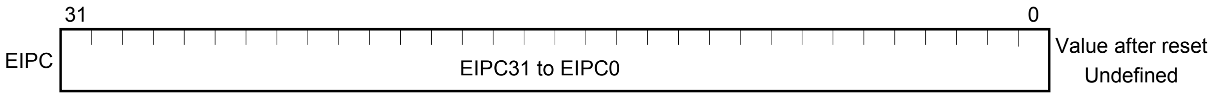


Table 3-4 EIPC Register Contents

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

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

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

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

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

	31	30	29									19	18 17 16	15	14		12	11 10 9	8	7	6	5	4	3	2	1	0	
EIPSW	0	UM	0	0	0	0	0	0	0	0	0	0	CU2 to CU0	EBV	0	0	0	Debug	0	NP	EP	ID	SAT	CY	OV	S	Z	Value after reset 00000020H

Table 3-5 EIPSW Register Contents

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

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

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

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

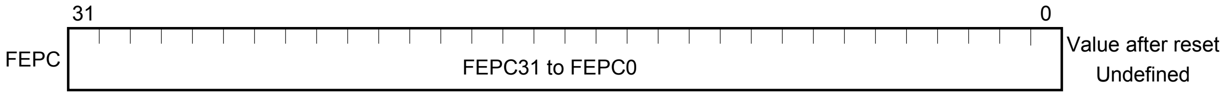


Table 3-6 FEPC Register Contents

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

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

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

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

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

	31	30	29										19	18 17 16	15	14		12	11 10 9	8	7	6	5	4	3	2	1	0	
FEPSW	0	UM	0	0	0	0	0	0	0	0	0	0	0	CU2 to CU0	EBV	0	0	0	Debug	0	NP	EP	ID	SAT	CY	OV	S	Z	Value after reset 00000020H

Table 3-7 FEPSW Register Contents

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

(5) PSW - Program status word

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

Cautions 1. When the LDSR instruction is used to change the contents of bits 7 to 0 in this register, the changed contents become valid from the instruction following the LDSR instruction.

2. The access permission for the PSW register differs depending on the bit. All bits can be read, but some bits can only be written under certain conditions. See **Table 3-8** for the access permission for each bit.

Table 3-8 Access Permission for PSW Register

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

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

	31	30	29									20	19	18 17 16	15	14		12	11 10 9	8	7	6	5	4	3	2	1	0	
PSW	0	UM	0	0	0	0	0	0	0	0	0	0	0	CU2 to CU0	EBV	0	0	0	Debug	0	NP	EP	ID	SAT	CY	OV	S	Z	Value after reset 00000020H

Table 3-9 PSW Register Contents (1/2)

Bit	Name	Description	R/W	Value after Reset
31	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
30	UM	This bit indicates that the CPU is in user mode (in UM mode). 0: Supervisor mode 1: User mode	R/W	0
29 to 19	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
18 to 16	CU2 to CU0	These bits indicate the coprocessor use permissions. When the bit corresponding to the coprocessor is 0, a coprocessor unusable exception occurs if an instruction for the coprocessor is executed or a coprocessor resource (system register) is accessed. CU2 bit 18: (Reserved for future expansion. Be sure to set to 0.) CU1 bit 17: (Reserved for future expansion. Be sure to set to 0.) CU0 bit 16: FPU	R/W	000
15	EBV	This bit indicates the reset vector and exception vector operation. See the description on RBASE and EBASE in this section.	R/W	0
14 to 12	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
11 to 9	Debug	This bit is used for the debug function for the development tool. Always set this bit to 0.	—	0
8	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
7	NP	This bit disables the acknowledgement of FE level exception. When an FE level exception is acknowledged, this bit is set to 1 to disable the acknowledgement of EI level and FE level exceptions. As for the exceptions which the NP bit disables the acknowledgment, see **Table 4-1 Exception Cause List**. 0: The acknowledgement of FE level exception is enabled. 1: The acknowledgement of FE level exception is disabled.	R/W	0
6	EP	This bit indicates that an exception other than an interrupt is being serviced. It is set to 1 when the corresponding exception occurs. This bit does not affect acknowledging an exception request even when it is set to 1. 0: An interrupt is being serviced. 1: An exception other than an interrupt is being serviced.	R/W	0
5	ID	This bit disables the acknowledgement of EI level exception. When an EI level or FE level exception is acknowledged, this bit is set to 1 to disable the acknowledgement of EI level exception. As for the exceptions which the ID bit disables the acknowledgment, see **Table 4-1 Exception Cause List**. This bit is also used to disable EI level exceptions from being acknowledged as a critical section while an ordinary program or interrupt is being serviced. It is set to 1 when the DI instruction is executed, and cleared to 0 when the EI instruction is executed. The change of the ID bit by the EI or ID instruction will be enabled from the next instruction. 0: The acknowledgement of EI level exception is enabled. 1: The acknowledgement of EI level exception is disabled.	R/W	1

Table 3-9 PSW Register Contents (2/2)

Bit	Name	Description	R/W	Value after Reset
4	SAT[Note]	This bit indicates that the operation result is saturated because the result of a saturated operation instruction operation has overflowed. This is a cumulative flag, so when the operation result of the saturated operation instruction becomes saturated, this bit is set to 1, but it is not later cleared to 0 when the operation result for a subsequent instruction is not saturated. This bit is cleared to 0 by the LDSR instruction. This bit is neither set to 1 nor cleared to 0 when an arithmetic operation instruction is executed. 0: Not saturated 1: Saturated	R/W	0
3	CY	This bit indicates whether a carry or borrow has occurred in the operation result. 0: Carry and borrow have not occurred. 1: Carry or borrow has occurred.	R/W	0
2	OV[Note]	This bit indicates whether or not an overflow has occurred during an operation. 0: Overflow has not occurred. 1: Overflow has occurred.	R/W	0
1	S[Note]	This bit indicates whether or not the result of an operation is negative. 0: Result of operation is positive or 0. 1: Result of operation is negative.	R/W	0
0	Z	This bit indicates whether or not the result of an operation is 0. 0: Result of operation is not 0. 1: Result of operation is 0.	R/W	0

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

Operation Result Status	Flag Status			Operation Result after Saturation Processing
	SAT	OV	S	
Exceeded positive maximum value	1	1	0	7FFFFFFFH
Exceeded negative maximum value	1	1	1	80000000H
Positive (maximum value not exceeded)	Value prior to operation is retained.	0	0	Operation result itself
Negative (maximum value not exceeded)			1	

(6) EIIC — EI level exception cause

The EIIC register retains the cause of any EI level exception that occurs. The value retained in this register is an exception code corresponding to a specific exception cause (see **Table 4-1 Exception Cause List**).

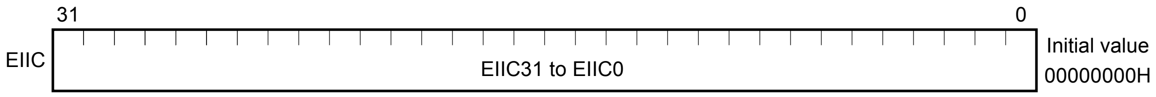


Table 3-10 EIIC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	EIIC31 to EIIC0	These bits store the exception cause code when an EI level exception occurs. The EIIC15-0 field stores the exception cause codes shown in **Table 4-1**. The EIIC31-16 field stores detailed exception cause codes defined individually for each exception. If there is no particular definition, these bits are set to 0.	R/W	0

(7) FEIC — FE level exception cause

The FEIC register retains the cause of any FE level exception that occurs. The value retained in this register is an exception code corresponding to a specific exception cause (see **Table 4-1 Exception Cause List**).

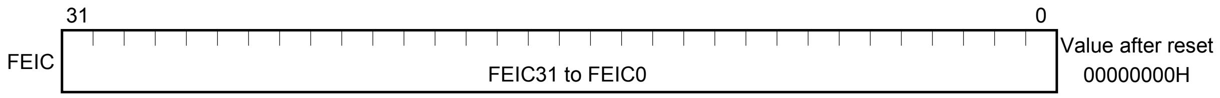


Table 3-11 FEIC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	FEIC31 to FEIC0	These bits store the exception cause code when an FE level exception occurs. The FEIC15-0 field stores the exception cause codes shown in **Table 4-1**. The FEIC31-16 field stores detailed exception cause codes defined individually for each exception. If there is no particular definition, these bits are set to 0.	R/W	0

(8) CTPC — Status save register when executing CALLT

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

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

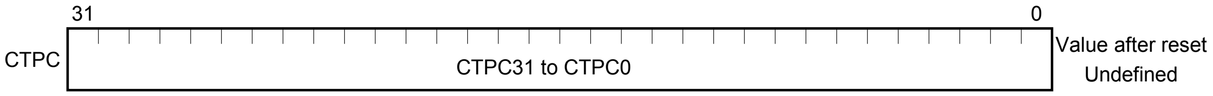


Table 3-12 CTPC Register Contents

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

(9) CTPSW — Status save register when executing CALLT

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

	31																										5	4	3	2	1	0	Value after reset
CTPSW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SAT	CY	OV	S	Z	00000000H

Table 3-13 CTPSW Register Contents

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

(10) CTBP — CALLT base pointer

The CTBP register is used to specify table addresses of the CALLT instruction and generate target addresses. Be sure to set the CTBP register to a halfword address.

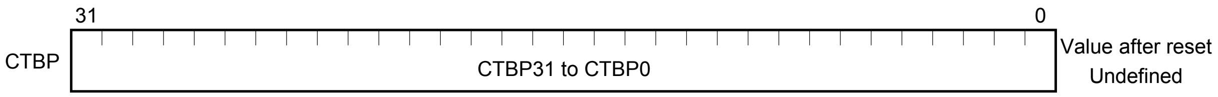


Table 3-14 CTBP Register Contents

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

(11) ASID — Address space ID

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

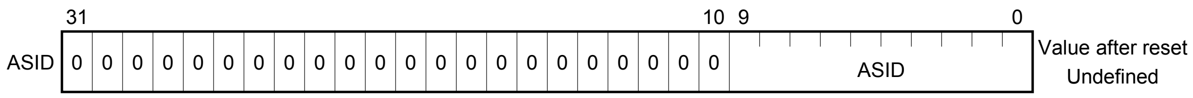


Table 3-15 ASID Register Contents

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

(12) EIWR — EI level exception working register

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

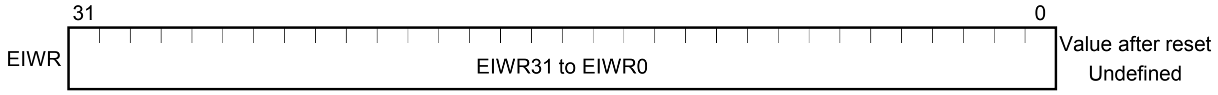


Table 3-16 EIWR Register Contents

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

(13) FEWR — FE level exception working register

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

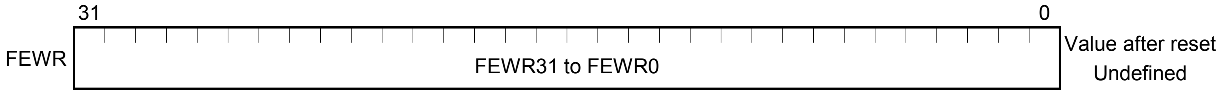


Table 3-17 FEWR Register Contents

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

(14) HTCFG0 — Thread configuration register

	31												19	18 16	15	14														0	
HTCFG0	0	0	0	0	0	0	0	0	0	0	0	0	0	PEID	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Value after reset Undefined

Table 3-18 HTCFG0 Register Contents

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

Note 1. When these bits are read, the CPU processor identifier defined in the product specifications is read. These bits cannot be written. For details, see the hardware manual of the product used.

(15) MEA — Memory error address register

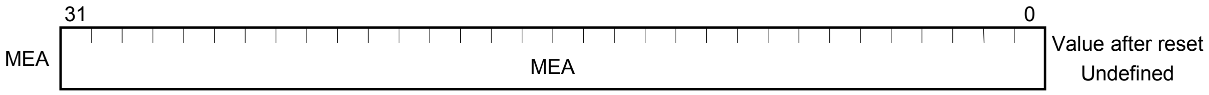


Table 3-19 MEA Register Contents

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

(16) MEI — Memory error information register

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

	31										21	20 16	15				11	10 9	8	7	6	5 1	0	
MEI	0	0	0	0	0	0	0	0	0	0	0	REG	0	0	0	0	0	DS	U	0	0	ITYPE	R W	Value after reset Undefined

Table 3-20 MEI Register Contents

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

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

Table 3-21 Instructions Causing Exceptions and Values of MEI Register

Instruction	REG	DS	U	RW	ITYPE
SLD.B	dst	0 (Byte)	0 (Signed)	0 (Read)	00000b
SLD.BU	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00000b
SLD.H	dst	1 (Half-word)	0 (Signed)	0 (Read)	00000b
SLD.HU	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00000b
SLD.W	dst	2 (Word)	0 (Signed)	0 (Read)	00000b
SST.B	src	0 (Byte)	0 (Signed)	1 (Write)	00000b
SST.H	src	1 (Half-word)	0 (Signed)	1 (Write)	00000b
SST.W	src	2 (Word)	0 (Signed)	1 (Write)	00000b
LD.B (disp16)	dst	0 (Byte)	0 (Signed)	0 (Read)	00001b
LD.BU (disp16)	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00001b
LD.H (disp16)	dst	1 (Half-word)	0 (Signed)	0 (Read)	00001b
LD.HU (disp16)	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00001b
LD.W (disp16)	dst	2 (Word)	0 (Signed)	0 (Read)	00001b
ST.B (disp16)	src	0 (Byte)	0 (Signed)	1 (Write)	00001b
ST.H (disp16)	src	1 (Half-word)	0 (Signed)	1 (Write)	00001b
ST.W (disp16)	src	2 (Word)	0 (Signed)	1 (Write)	00001b
LD.B (disp23)	dst	0 (Byte)	0 (Signed)	0 (Read)	00010b
LD.BU (disp23)	dst	0 (Byte)	1 (Unsigned)	0 (Read)	00010b
LD.H (disp23)	dst	1 (Half-word)	0 (Signed)	0 (Read)	00010b
LD.HU (disp23)	dst	1 (Half-word)	1 (Unsigned)	0 (Read)	00010b
LD.W (disp23)	dst	2 (Word)	0 (Signed)	0 (Read)	00010b
ST.B (disp23)	src	0 (Byte)	0 (Signed)	1 (Write)	00010b
ST.H (disp23)	src	1 (Half-word)	0 (Signed)	1 (Write)	00010b
ST.W (disp23)	src	2 (Word)	0 (Signed)	1 (Write)	00010b
LD.DW (disp23)	dst	3 (Double-word)	0 (Signed)	0 (Read)	00010b
ST.DW (disp23)	src	3 (Double-word)	0 (Signed)	1 (Write)	00010b
LDL.W	dst	2 (Word)	0 (Signed)	0 (Read)	00111b
STC.W	src	2 (Word)	0 (Signed)	1 (Write)	00111b
CAXI	dst	2 (Word)	1 (Unsigned)	0 (Read)Note 1	01000b
SET1	-	0 (Byte)	1 (Unsigned)	0 (Read)Note 1	01001b
CLR1	-	0 (Byte)	1 (Unsigned)	0 (Read)Note 1	01001b
NOT1	-	0 (Byte)	1 (Unsigned)	0 (Read)Note 1	01001b
TST1	-	0 (Byte)	1 (Unsigned)	0 (Read)	01001b
PREPARE	-	2 (Word)	1 (Unsigned)	1 (Write)	01100b
DISPOSE	-	2 (Word)	1 (Unsigned)	0 (Read)	01100b
PUSHSP	-	2 (Word)	1 (Unsigned)	1 (Write)	01101b
POPSP	-	2 (Word)	1 (Unsigned)	0 (Read)	01101b
SWITCH	-	1 (Half-word)	0 (Signed)	0 (Read)	10000b
CALLT	-	1 (Half-word)	1 (Unsigned)	0 (Read)	10001b
SYSCALL	-	2 (Word)	1 (Unsigned)	0 (Read)	10010b
CACHE	-	-	-	0 (Read)	10100b
Interrupt (table reference method)Note 2	-	2 (Word)	1 (Unsigned)	0 (Read)	10101b

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

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

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

(17) RBASE — Reset vector base address

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

	31 9	8							0	
RBASE	RBASE31 to RBASE9	0	0	0	0	0	0	0	0	RINT	Value after reset **Note**

Table 3-22 RBASE Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 9	RBASE31 to RBASE9	These bits indicate the reset vector when there is a reset. When PSW.EBV = 0, this address is also used as the exception vector. The RBASE8-0 bits are not assigned as names because these bits are always 0.	R	**Note**
8 to 1	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	RINT	When the RINT bit is set, the exception handler address for interrupt processing is reduced. See **4.5.1 (1) Direct vector method**. This bit is valid when PSW.EBV = 0.	R	**Note**

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

(18) EBASE — Exception handler vector address

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

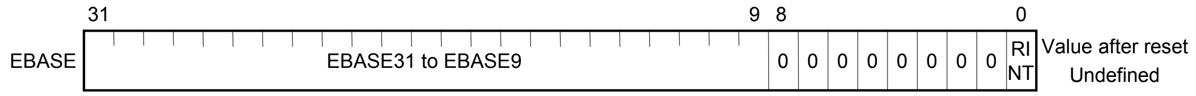


Table 3-23 EBASE Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 9	EBASE31 to EBASE9	The exception handler routine address is changed to the address resulting from adding the offset address of each exception to the base address specified for this register. The EBASE8-0 bits are not assigned as names because these bits are always 0.	R/W	Undefined
8 to 1	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	RINT	When the RINT bit is set, the exception handler address for interrupt processing is reduced. See **4.5.1 (1) Direct vector method**.	R/W	Undefined

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

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

	31 … 9	8								0	
INTBP	INTBP31 to INTBP9	0	0	0	0	0	0	0	0	0	Value after reset Undefined

Table 3-24 INTBP Register Contents

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

(20) PID — Processor ID

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

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

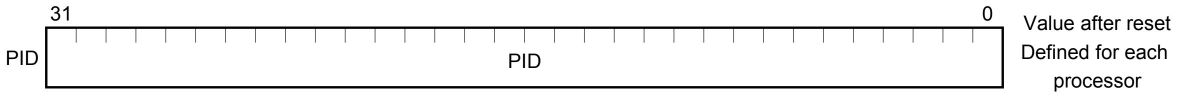


Table 3-25 PID Register Contents

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

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

(21) SCCFG — SYSCALL operation setting

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

	31																							8	7 0	
SCCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SIZE	Value after reset Undefined

Table 3-26 SCCFG Register Contents

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

(22) SCBP — SYSCALL base pointer

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

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

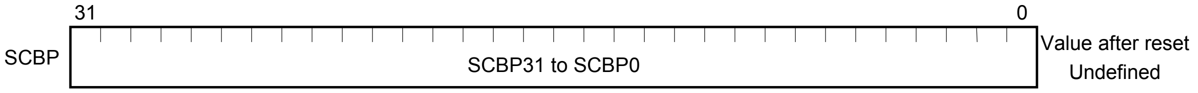


Table 3-27 SCBP Register Contents

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

(23) MCFG0 — Machine configuration

This register indicates the CPU configuration.

	31							24	23 16	15												3	2	1	0	
MCFG0	0	0	0	0	0	0	0	0	SPID	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Value after reset Undefined

Table 3-28 MCFG0 Register Contents

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

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

(24) MCTL — Machine control

This register is used to control the CPU.

	31	30																												2	1	0	
MCTL	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MA	UIC	Value after reset Undefined

Table 3-29 MCTL Register Contents

Bit	Name	Description	R/W	Value after Reset
31	—	(Reserved for future expansion. Be sure to set to 1.)	R	1
30 to 2	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
1	MA	This bit specifies the operation when a misaligned access occurs. 0: When a misaligned access occurs, an exception always occurs. Note 2 1: Hardware operates normally. Note 3	R/W Note 1	0 Note 1
0	UIC	This bit is used to control the interrupt enable/disable operation in user mode. When this bit is set to 1, executing the EI/DI instruction in user mode become possible.	R/W	0

Notes 1. The value after reset and whether each bit can be written depend on the hardware specifications. For details, see the hardware manual of the product used.

2. Excluding LD.DW and ST.DW instructions executed at an address at a word boundary.

3. The case in which an MAE exception occurs and the case in which access is performed by hardware are normally defined separately by the hardware specifications. For details, see the hardware manual of the product used.

3.3 Interrupt Function Registers

3.3.1 Interrupt Function System Registers

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

Table 3-30 Interrupt Function System Registers

Register No. (regID, selID)	Symbol	Function	Access Permission
SR7, 1	FPIPR	FPI exception interrupt priority setting	SV
SR10, 2	ISPR	Priority of interrupt being serviced	SV
SR11, 2	PMR	Interrupt priority masking	SV
SR12, 2	ICSR	Interrupt control status	SV
SR13, 2	INTCFG	Interrupt function setting	SV

(1) FPIPR — FPI exception interrupt priority setting

This register is used to specify the interrupt priority of FPI exceptions.

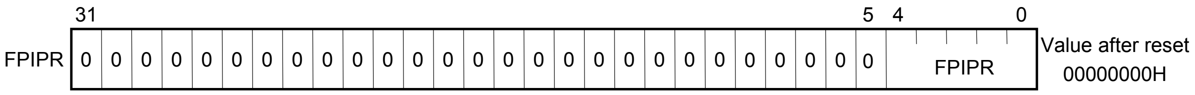


Table 3-31 FPIPR Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 5	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
4 to 0	FPIPR	These bits are used to specify the interrupt priority of floating-point operation exceptions (imprecise) (FPI). Specify values from 0 to 16. Specifying 17 or greater is prohibited.Note 1 FPI exceptions are handled using the specified interrupt priority. If an FPI exception occurs at the same time as an interrupt that has the same priority, the FPI exception is prioritized. **Caution** If 17 or greater is specified, it is handled as 16.	R/W	0

Note 1. The settable value may differ depending on the hardware specifications. For details, see the hardware manual of the product used.

(2) ISPR — Priority of interrupt being serviced

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

	31															16	15 0	
ISPR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ISP15 to ISP0	Value after reset 00000000H

Table 3-32 ISPR Register Contents

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th><th>R/W</th><th>Value after Reset</th></tr>
<tr><td>31 to 16</td><td>—</td><td>(Reserved for future expansion. Be sure to set to 0.)</td><td>R</td><td>0</td></tr>
<tr><td>15 to 0</td><td>ISP15 to ISP0</td><td>These bits indicate the acknowledgment status of an EIINT<i>n</i> interrupt with a priority[Note 1] that corresponds to the relevant bit position.
0: An interrupt request for an interrupt whose priority corresponds to the relevant bit position has not been acknowledged.
1: An interrupt request for an interrupt whose priority corresponds to the relevant position is being serviced by the CPU core.

The bit positions correspond to the following priority levels.
<table>
<tr><th>Bit</th><th>Priority</th></tr>
<tr><td>0</td><td>Priority 0 (highest)</td></tr>
<tr><td>1</td><td>Priority 1</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>14</td><td>Priority 14</td></tr>
<tr><td>15</td><td>Priority 15</td></tr>
</table>
When an interrupt request (EIINT<i>n</i>) is acknowledged, the bit corresponding to the acknowledged interrupt request is automatically set to 1. If PSW.EP is 0 when the EIRET instruction is executed, the bit with the highest priority among the ISP15-0 bits that are set (0 is the highest priority) is cleared to 0[Note 2].
While a bit in this register is set to 1, same or lower priority interrupts (EIINT<i>n</i>) and the FPI exception[Note 3] are masked. Priority level judgment is therefore not performed when the system is determining whether to acknowledge an exception, meaning that exceptions will not be acknowledged. For details, see <b>4.1.5 Exception Acknowledgment Priority and Pending Conditions</b>.
When performing software-based priority control using the PMR register, be sure to clear this register by using the INTCFG.ISPC bit.</td><td>R[Note 4]</td><td>0</td></tr>
</table>

Notes 1. For details, see **4.1.5 Exception Acknowledgment Priority and Pending Conditions**.

2. Interrupt acknowledgment and auto-updating of values when the EIRET instruction is executed are disabled by setting (1) the INTCFG.ISPC bit. It is recommended to enable auto-updating of values, so in normal cases, the INTCFG.ISPC bit should be cleared to 0.

3. The FPI exception has the same priority level as an interrupt (EIINT*n*), so it is affected by the setting of the ISPR register in the same way as an interrupt. The priority level of the FPI exception is specified by the FPIPR register.

4. This is R or R/W, depending on the setting of the INTCFG.ISPC bit. It is recommended to use this register as a read-only (R) register.

(3) PMR — Interrupt priority masking

This register is used to mask the specified interrupt priority.

	31															16	15 0	
PMR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PM15 to PM0	Value after reset 00000000H

Table 3-33 PMR Register Contents

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th><th>R/W</th><th>Value after Reset</th></tr>
<tr><td>31 to 16</td><td>—</td><td>(Reserved for future expansion. Be sure to set to 0.)</td><td>R</td><td>0</td></tr>
<tr><td>15 to 0</td><td>PM15 to PM0</td><td>These bits mask an interrupt request with a priority level that corresponds to the relevant bit position.
0: Servicing of an interrupt request with a priority that corresponds to the relevant bit position is enabled.
1: Servicing of an interrupt request with a priority that corresponds to the relevant bit position is disabled.

The bit positions correspond to the following priority levels:
<table>
<tr><th>Bit</th><th>Priority</th></tr>
<tr><td>0</td><td>Priority 0 (highest)</td></tr>
<tr><td>1</td><td>Priority 1</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>14</td><td>Priority 14</td></tr>
<tr><td>15</td><td>Priority 15 and priority 16 (lowest)[Note 3]</td></tr>
</table>
While a bit in this register is set to 1, interrupts (EIINT<i>n</i>) with the priority corresponding to that bit and the FPI exception[Note 1] are masked. Priority level judgment is therefore not performed when the system is determining whether to acknowledge an exception, meaning that exceptions will not be acknowledged[Note 2].</td><td>R/W</td><td>0</td></tr>
</table>

Notes 1. The FPI exception has the same priority level as an interrupt (EIINT*n*), so it is affected by the setting of the PMR register in the same way as an interrupt. The priority level of the FPI exception is specified by the FPIPR register.

2. Specify the masks by setting the bits to 1 in order from the lowest-priority bit. For example, FF00H can be set, but F0F0H or 00FFH cannot.

3. The corresponding priority level may differ depending on the hardware specifications. For details, see the hardware manual of the product used.

(4) ICSR — Interrupt control status

This register indicates the interrupt control status in the CPU.

	31																													2	1	0	Value after reset
ICSR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PMFP	PMEI	00000000H

Table 3-34 ICSR Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 2	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
1	PMFP	This bit indicates that an FPI exception with the priority level masked by the PMR register exists.	R	0
0	PMEI	This bit indicates that an interrupt (EIINT*n*) with the priority level masked by the PMR register exists.	R	0

(5) INTCFG — Interrupt function setting

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

	31																														1	0	
INTCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ISPC	Value after reset 00000000H

Table 3-35 INTCFG Register Contents

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

3.4 FPU Function Registers

3.4.1 Floating-Point Registers

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

- Single-precision floating-point instruction:
  Thirty-two 32-bit registers can be specified. These general-purpose registers correspond to r0 to r31.
- Double-precision floating-point instruction:
  Sixteen 64-bit registers can be specified. Paired general-purpose registers are used as register pairs ({r1, r0}, {r3, r2} … {r31, r30}). Each register pair is specified in the instruction format with an even numbered register. Because r0 is a zero register (always holds 0), in principle {r1, r0} cannot be used by a double-precision floating-point instruction.

3.4.2 Floating-Point Function System Registers

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

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

Table 3-36 FPU System Registers

Register No. (regID, selID)	Symbol	Function	Access Permission
SR6, 0	FPSR	Floating-point configuration/status	CU0 and SV
SR7, 0	FPEPC	Floating-point exception program counter	CU0 and SV
SR8, 0	FPST	Floating point status	CU0
SR9, 0	FPCC	Floating-point comparison result	CU0
SR10, 0	FPCFG	Floating-point configuration	CU0
SR11, 0	FPEC	Floating-point exception control	CU0 and SV

(1) FPSR — Floating-point configuration/status

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

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPSR	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	FN	IF	PEM	0	RM		FS	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Value after reset
	Cause bits (XC)					Enable bits (XE)					Preservation bits (XP)					
E	V	Z	O	U	I	V	Z	O	U	I	V	Z	O	U	I	**Note**

Table 3-37 FPSR Register Contents (1/2)

Bit	Name	Description	R/W	Value after Reset
31 to 24	CC(7:0)	These are the CC (condition) bits. They store the results of floating-point comparison instructions. The CC7-0 bits are not affected by any instructions except the comparison instruction and LDSR instruction. 0: Comparison result is false 1: Comparison result is true	R/W	Undefined
23	FN	This bit enables flush-to-nearest mode. When the FN bit is set to 1, if the rounding mode is RN and the operation result is a subnormal number, the number is flushed to the nearest number. For details, see **6.1.11 Flush to Nearest**.	R/W	0
22	IF	This bit accumulates and indicates information about the flushing of input operands. For details about flushing subnormal numbers, see **6.1.9 Flushing Subnormal Numbers**.	R/W	0
21	PEM	This bit specifies whether to handle an exception as a precise exception. If the PEM bit is 1, exceptions that are caused by the execution of a floating-point operation instruction are handled as precise exceptions.	R/W	0
20	—	(Reserved for future expansion. Be sure to set to 0.)	R	0

Note See the descriptions of each bit.

Table 3-37 FPSR Register Contents (2/2)

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th><th>R/W</th><th>Value after Reset</th></tr>
<tr><td>19, 18</td><td>RM</td><td>These are the rounding mode control bits. The RM bits define the rounding mode that the FPU uses for all floating-point instructions.
<table>
<tr><th colspan="2">RM Bits</th><th rowspan="2">Mnemonic</th><th rowspan="2">Description</th></tr>
<tr><th>19</th><th>18</th></tr>
<tr><td>0</td><td>0</td><td>RN</td><td>Rounds the result to the nearest representable value. If the value is exactly in-between the two nearest representable values, the result is rounded toward the value whose least significant bit is 0.</td></tr>
<tr><td>0</td><td>1</td><td>RZ</td><td>Rounds the result toward 0. The result is the nearest to the value that does not exceed the absolute value of the result with infinite accuracy.</td></tr>
<tr><td>1</td><td>0</td><td>RP</td><td>Rounds the result toward +∞. The result is nearest to a value greater than the accurate result with infinite accuracy.</td></tr>
<tr><td>1</td><td>1</td><td>RM</td><td>Rounds the result toward −∞. The result is nearest to a value less than the accurate result with infinite accuracy.</td></tr>
</table></td><td>R/W</td><td>00</td></tr>
<tr><td>17</td><td>FS</td><td>This bit enables values that could not be normalized (subnormal numbers) to be flushed. If the FS bit is set, input operands and operation results that are subnormal numbers are flushed without causing an unimplemented operation exception (E). An input operand that is a subnormal number is flushed to 0 with the same sign. Operation results that are subnormal numbers either become 0 or the minimum normalized number, depending on the rounding mode.
<table>
<tr><th rowspan="2">Operation Result that is a Subnormal Number</th><th colspan="4">Rounding Mode and Value after Flushing</th></tr>
<tr><th>RN[Note]</th><th>RZ</th><th>RP</th><th>RM</th></tr>
<tr><td>Positive</td><td>+0</td><td>+0</td><td>$+2^{\mathrm{Emin}}$</td><td>+0</td></tr>
<tr><td>Negative</td><td>−0</td><td>−0</td><td>−0</td><td>-2^{Emin}</td></tr>
</table>
Note If the rounding mode is RN and the FPSR.FN bit is set, flushing will occur in the direction of higher accuracy. For details, see **6.1.11 Flush to Nearest**.</td><td>R/W</td><td>1</td></tr>
<tr><td>16</td><td>—</td><td>(Reserved for future expansion. Be sure to set to 0.)</td><td>R</td><td>0</td></tr>
<tr><td>15 to 10</td><td>XC
(E, V, Z, O, U, I)</td><td>These are the cause bits. For details, see <b>3.4.2 (1) (a) Cause bits (XC)</b>.</td><td>R/W</td><td>Undefined</td></tr>
<tr><td>9 to 5</td><td>XE
(V, Z, O, U, I)</td><td>These are the enable bits. For details, see <b>3.4.2 (1) (b) Enable bits (XE)</b>.</td><td>R/W</td><td>0</td></tr>
<tr><td>4 to 0</td><td>XP
(V, Z, O, U, I)</td><td>These are the preservation bits. For details, see <b>3.4.2 (1) (c) Preservation bits (XP)</b>.</td><td>R/W</td><td>Undefined</td></tr>
</table>

(a) Cause bits (XC)

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

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

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

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

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

(b) Enable bits (XE)

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

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

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

(c) Preservation bits (XP)

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

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

Remark For details about the exception types and how they relate to particular bits, see **Figure 6-6 Cause, Enable, and Preservation Bits of FPSR Register**.

(2) FPEPC — Floating-point exception program counter

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

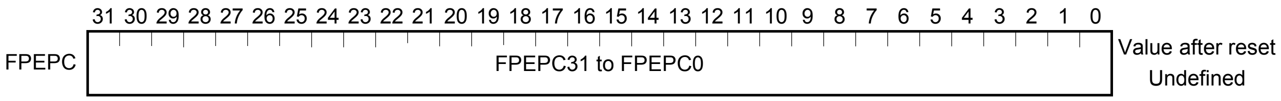


Table 3-38 FPEPC Register Contents

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

(3) FPST — Floating-point operation status

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPST	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
		Cause bits (XC)									Preservation bits (XP)					Value after reset
0	0	E	V	Z	O	U	I	0	0	IF	V	Z	O	U	I	Undefined

Table 3-39 FPST Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 14	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
13 to 8	XC (E, V, Z, O, U, I)	These are cause bits. For details, see **3.4.2 (1) (a) Cause bits (XC)**. Values written to these bits are reflected in FPSR.XC bits.	R/W	Undefined
7, 6	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
5	IF	This bit accumulates and indicates information about the flushing of input operands. For details about flushing subnormal numbers, see **6.1.9 Flushing Subnormal Numbers**. Value written to this bit is reflected in FPSR.IF bit.	R/W	0
4 to 0	XP (V, Z, O, U, I)	These are preservation bits. For details, see **3.4.2 (1) (c) Preservation bits (XP)**. Values written to these bits are reflected in FPSR.XP bits.	R/W	Undefined

(4) FPCC — Floating-point operation comparison result

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPCC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	Value after reset Undefined

Table 3-40 FPCC Register Contents

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

(5) FPCFG — Floating-point operation configuration

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
												Enable bits (XE)				Value after reset
0	0	0	0	0	0	RM		0	0	0	V	Z	O	U	I	00000000H

Table 3-41 FPCFG Register Contents

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th><th>R/W</th><th>Value after Reset</th></tr>
<tr><td>31 to 10</td><td>—</td><td>(Reserved for future expansion. Be sure to set to 0.)</td><td>R</td><td>0</td></tr>
<tr><td>9, 8</td><td>RM</td><td>These are rounding mode control bits. The RM bits define the rounding mode that the FPU uses for all floating-point instructions. Values written to these bits are reflected in RM bits of FPSR.
<table>
<tr><th colspan="2">RM Bits</th><th rowspan="2">Mnemonic</th><th rowspan="2">Description</th></tr>
<tr><th>9</th><th>8</th></tr>
<tr><td>0</td><td>0</td><td>RN</td><td>Rounds the result to the nearest representable value. If the value is exactly in-between the two representable values, the result is rounded toward the value whose least significant bit is 0.</td></tr>
<tr><td>0</td><td>1</td><td>RZ</td><td>Rounds the result toward 0. The result is the nearest to the value that does not exceed the absolute value of the result with infinite accuracy.</td></tr>
<tr><td>1</td><td>0</td><td>RP</td><td>Rounds the result toward +∞. The result is nearest to a value greater than the accurate result with infinite accuracy.</td></tr>
<tr><td>1</td><td>1</td><td>RM</td><td>Rounds the result toward -∞. The result is nearest to a value less than the accurate result with infinite accuracy.</td></tr>
</table></td><td>R/W</td><td>0</td></tr>
<tr><td>7 to 5</td><td>—</td><td>(Reserved for future expansion. Be sure to set to 0.)</td><td>R</td><td>0</td></tr>
<tr><td>4 to 0</td><td>XE
(V, Z, O, U, I)</td><td>These are the enable bits. For details, see <b>3.4.2 (1) (b) Enable bits (XE)</b>. Values written to these bits are reflected in the FPSR.XE bits.</td><td>R/W</td><td>0</td></tr>
</table>

(6) FPEC — Floating-point exception control

This register controls the floating-point operation exception.

Caution For how to handle the FPEC register, see **4.4 Exception Management**.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FPEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FPI VD	Value after reset 00000000H

Table 3-42 FPEC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 1	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	FPIVD[Note]	This bit indicates the status of reporting the FPI exception. If this bit is set to 1, the FPI exception is reported to the CPU but is not acknowledged. It is automatically cleared to 0 when the CPU acknowledges the FPI exception. While this bit is set to 1, all the floating-point instructions are invalidated. Report of the FPI exception can be canceled by clearing (0) this bit by the LDSR instruction while it is set to 1. When report of the FPI exception is canceled, the CPU does not acknowledge the FPI exception. 0: FPI exception is not reported. 1: FPI exception is reported.	R/W	0

Note The FPIVD bit can only be cleared to 0 by the write operation of the LDSR instruction. It cannot be set to 1.

3.5 MPU Function Registers

3.5.1 MPU Function System Registers

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

Table 3-43 MPU Function System Registers (1/2)

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

Table 3-43 MPU Function System Registers (2/2)

Register No. (regID, selID)	Symbol	Function	Access Permission
SR28, 6	MPLA7	Protection area minimum address	SV
SR29, 6	MPUA7	Protection area maximum address	SV
SR30, 6	MPAT7	Protection area attribute	SV
SR0, 7	MPLA8	Protection area minimum address	SV
SR1, 7	MPUA8	Protection area maximum address	SV
SR2, 7	MPAT8	Protection area attribute	SV
SR4, 7	MPLA9	Protection area minimum address	SV
SR5, 7	MPUA9	Protection area maximum address	SV
SR6, 7	MPAT9	Protection area attribute	SV
SR8, 7	MPLA10	Protection area minimum address	SV
SR9, 7	MPUA10	Protection area maximum address	SV
SR10, 7	MPAT10	Protection area attribute	SV
SR12, 7	MPLA11	Protection area minimum address	SV
SR13, 7	MPUA11	Protection area maximum address	SV
SR14, 7	MPAT11	Protection area attribute	SV
SR16, 7	MPLA12	Protection area minimum address	SV
SR17, 7	MPUA12	Protection area maximum address	SV
SR18, 7	MPAT12	Protection area attribute	SV
SR20, 7	MPLA13	Protection area minimum address	SV
SR21, 7	MPUA13	Protection area maximum address	SV
SR22, 7	MPAT13	Protection area attribute	SV
SR24, 7	MPLA14	Protection area minimum address	SV
SR25, 7	MPUA14	Protection area maximum address	SV
SR26, 7	MPAT14	Protection area attribute	SV
SR28, 7	MPLA15	Protection area minimum address	SV
SR29, 7	MPUA15	Protection area maximum address	SV
SR30, 7	MPAT15	Protection area attribute	SV

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

(1) MPM — Memory protection operation mode

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

	31																				11	10	9	8	7					2	1	0
MPM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DX	DW	DR	0	0	0	0	0	0	SVP	MPE

Value after reset
00000000H

Table 3-44 MPM Register Contents (1/2)

Bit	Name	Description	R/W	Value after Reset
31 to 11	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
10	DX	This bit specifies the default operation when an instruction is executed at an address that does not exist in a protection area. “0” is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0: Disable executing an instruction at an address that does not exist in a protection area. 1: Enable executing an instruction at an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see **5.1.4 Caution Points for Protection Area Setup**.	R	0
9	DW	This bit specifies the default operation when writing to an address that does not exist in a protection area. “0” is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0: Disable writing to an address that does not exist in a protection area. 1: Enable writing to an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see **5.1.4 Caution Points for Protection Area Setup**.	R	0
8	DR	This bit specifies the default operation when reading from an address that does not exist in a protection area. “0” is fixed for this bit in this CPU. Default operation is prohibited. Be sure to set to 0. 0: Disable reading from an address that does not exist in a protection area. 1: Enable reading from an address that does not exist in a protection area. The setting of this bit affects the access operation when the protection areas overlap. For details, see **5.1.4 Caution Points for Protection Area Setup**.	R	0
7 to 2	—	(Reserved for future expansion. Be sure to set to 0.)	R	0

Table 3-44 MPM Register Contents (2/2)

Bit	Name	Description	R/W	Value after Reset
1	SVP	In SV mode (when PSW.UM = 0), this bit is used to specify whether to restrict access according to the SX, SW, and SR bits of the MPAT register for each protection area.Note 1 0: As usual, implicitly enable all access in SV mode. 1: Restrict access according to the SX, SW, and SR bits even in SV mode.Note 2	R/W	0
0	MPE	This bit is used to specify whether to enable or disable MPU function. 0: Disable 1: Enable	R/W	0

Notes 1. When the SVP bit is set to 1, access is restricted according to the setting of each protection area even in SV mode. Therefore, specify protection areas before setting the SVP bit to prevent the access of the program itself from being restricted.

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

(2) MPRC — MPU region control

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

	31																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MPRC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E15	E14	E13	E12	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	Value after reset 00000000H

Table 3-45 MPRC Register Contents

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

(3) MPBRGN — MPU base region

This register indicates the minimum usable MPU area number.

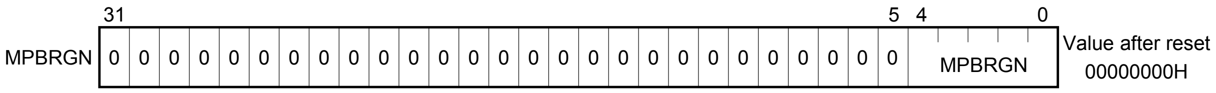


Table 3-46 MPBRGN Register Contents

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

(4) MPTRGN — MPU end region

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

	31																										5	4 0	
MPTRGN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MPTRGN	Value after reset **Note**

Table 3-47 MPTRGN Register Contents

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

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

(5) MCA — Memory protection setting check address

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

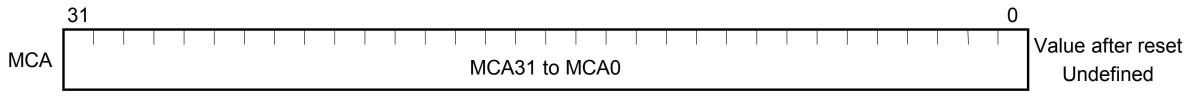


Table 3-48 MCA Register Contents

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

(6) MCS — Memory protection setting check size

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

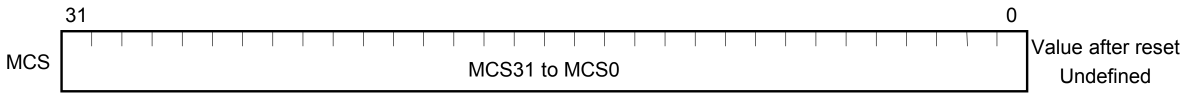


Table 3-49 MCS Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MCS31 to MCS0	These bits are used to specify the size of the memory area which subjects to a memory protection setting check and the size of the target area in bytes. Because the specified size is assumed to represent an unsigned integer, it is not possible to check an area in the direction in which the address value decreases relative to the MCA register value. Do not specify 00000000H for the MCS register.	R/W	Undefined

(7) MCC — Memory protection setting check command

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

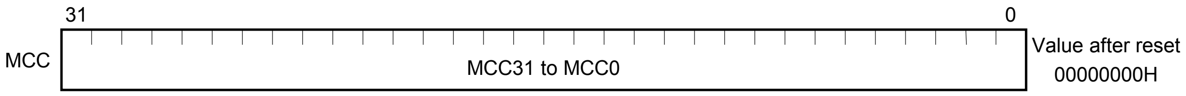


Table 3-50 MCC Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	MCC31 to MCC0	When any value is written to the MCC register, a memory protection setting check starts. By setting up the MCA/MCS register and then writing to the MCC register, results are stored in MCR. Because the check is started by any written value, a check can be started by using r0 as the source register without using any unnecessary registers. Note that, for the check, the results are applied according to each area setting regardless of the state of the PSW.UM bit. When the MCC register is read, value 00000000H is always returned.	R/W	0

(8) MCR — Memory protection setting check result

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

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

Cautions 1. If the specified area to be checked crosses 00000000H or 7FFFFFFFH, it is judged as an area setting error, and the MCR.OV bit is set to 1. This means that the MCR.OV bit must be checked to access the check results. Do not use the check result until it is confirmed that the result is not invalid (OV = 0).

2. When the default set (MPM.DX, DW, DR) is set to 1, it disables sometimes to get the correct result. If enabling the specified default operation, do not use the memory protection setting check function.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	OV	0	0	SXE	SWE	SRE	UXE	UWE	URE	Value after reset Undefined

Table 3-51 MCR Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 9	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
8	OV	If the specified area includes 00000000H or 7FFFFFFFH, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
7, 6	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
5	SXE	If the specified area is contained within one protection area and execution is permitted for that area in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
4	SWE	If the specified area is contained within one protection area and writing to that area is permitted in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
3	SRE	If the specified area is contained within one protection area and reading from that area is permitted in supervisor mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
2	UXE	If the specified area is contained within one protection area and execution is permitted for that area in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
1	UWE	If the specified area is contained within one protection area and writing to that area is permitted in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined
0	URE	If the specified area is contained within one protection area and reading from that area is permitted in user mode, 1 is stored in this bit. In other cases, 0 is stored in this bit.	R/W	Undefined

(9) MPLAn — Protection area minimum address

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

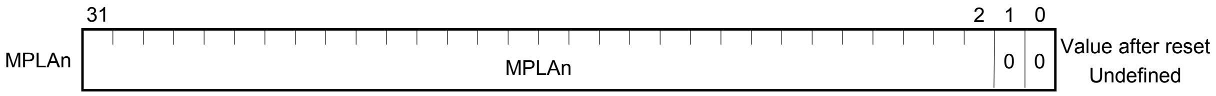


Table 3-52 MPLAn Register Contents

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

(10) MPUAn — Protection area maximum address

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

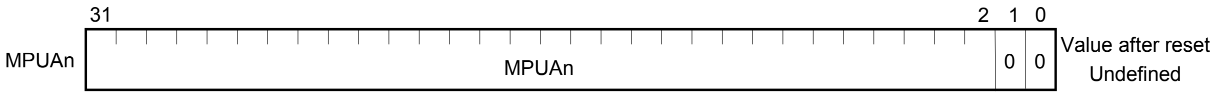


Table 3-53 MPUAn Register Contents

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

(11) MPATn — Protection area attribute

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

	31					26	25 16	15								8	7	6	5	4	3	2	1	0	
MPATn	0	0	0	0	0	0	ASID	0	0	0	0	0	0	0	0	0	E	G	SX	SW	SR	UX	UW	UR	Value after reset Undefined

Table 3-54 MPATn Register Contents

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

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

3.6 Cache Operation Function Registers

3.6.1 Cache Control Function System Registers

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

Table 3-55 Cache Control System Registers

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

(1) ICTAGL — Instruction cache tag Lo access

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

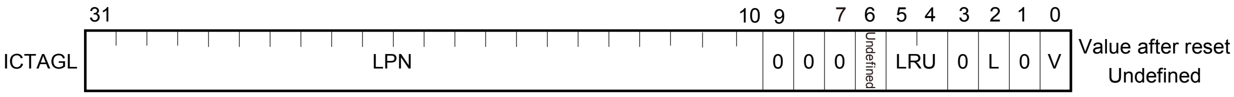


Table 3-56 ICTAGL Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 10	LPN	These bits store physical page number bits 24 to 12. Be sure to set bits 31 to 25, and 10 to 0.	R/W	Undefined
9 to 7	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
6	—	(Reserved for future expansion. Be sure to set to 0.)	R	Undefined
5, 4	LRU	These bits indicate LRU information of specified cache line. LRU information cannot be freely changed to any value by the CIST instruction.	R/W	Undefined
3	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
2	L	This bit stores the lock information.	R/W	Undefined
1	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
0	V	This bit stores valid/invalid information of specified cache line.	R/W	Undefined

(2) ICTAGH — Instruction cache tag Hi access

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

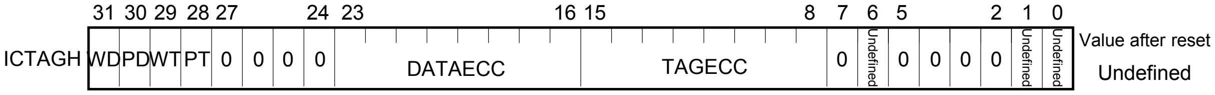


Table 3-57 ICTAGH Register Contents

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

(3) ICDATL — Instruction cache data Lo access

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

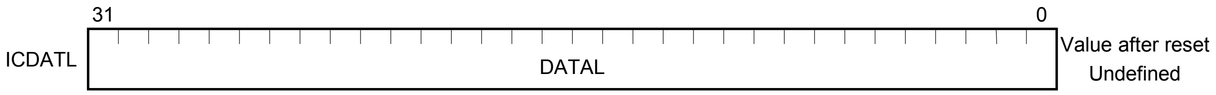


Table 3-58 ICDATL Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	DATAL	Bits 31 to 0 or bits 95 to 64 are stored among the instruction data of a block in the specified cache line. The stored bits are specified by the offset of index. Offset of index = 0000 : Bits 31 to 0 Offset of index = 1000 : Bits 95 to 64	R/W	Undefined

(4) ICDATH — Instruction cache data Hi access

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

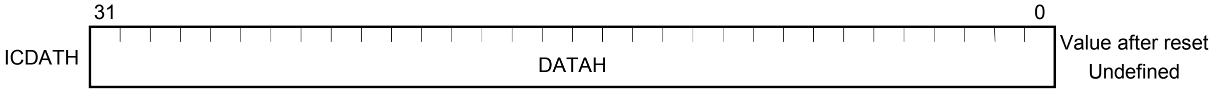


Table 3-59 ICDATH Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 0	DATAH	Bits 63 to 32 or bits 127 to 96 are stored among the instruction data of a block in the specified cache line. The stored bits are specified by the offset of index. Offset of index = 0000 : Bits 63 to 32 Offset of index = 1000 : Bits 127 to 96	R/W	Undefined

(5) ICCTRL — Instruction cache control

This register is used to control the instruction cache.

	31													18	17	16	15						9	8	7				3	2	1	0	
ICCTRL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	D1 EIV	1	0	0	0	0	0	0	0	ICH CLR	0	0	0	0	0	ICH EIV	ICH EM K	ICH EN	Value after reset 0001 0003H

Table 3-60 ICCTRL Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 18	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
17	D1EIV	This bit selects the operation in response to 1-bit errors in the data RAM. 0: After the error is corrected, processing continues, but the entry that had an error is retained. 1: The error is not corrected, the entry is cleared, and fetching is repeated. This bit is read as the previous value until the setting is actually reflected in the instruction cache.	R/W	0
16	—	(Reserved for future expansion. Be sure to set to 1.)	R	1
15 to 9	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
8	ICHCLR	When this bit is set to 1, the entire instruction cache is cleared. After this bit is set to 1, it is read as 1 until clearing is completed. The bit is cleared to 0 once clearing is completed.	R/W	0
7 to 3	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
2	ICHEIV	When this bit is set to 1, the instruction cache is automatically set as invalid (the ICHEN bit is cleared to 0) whenever a cache error occurs.	R/W	0
1	ICHEMK	When this bit is set to 1, it masks notification of cache error exceptions for the CPU after a cache error has occurred.	R/W	1
0	ICHEN	This bit indicates valid/invalid status of instruction cache. 0: Instruction cache is invalid 1: Instruction cache is valid This bit is read as the previous value until the setting is actually reflected in the instruction cache.	R/W	1

(6) ICCFG — Instruction cache configuration

This register indicates the instruction cache configuration.

	31														17	16	15	14 to 8	7 to 4	3 to 0	
ICCFG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	ICHSIZE	ICHLINE	ICHWAY	Value after reset **Note**

Table 3-61 ICCFG Register Contents

Bit	Name	Description	R/W	Value after Reset
31 to 17	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
16	—	(Reserved for future expansion. Be sure to set to 1.)	R	1
15	—	(Reserved for future expansion. Be sure to set to 0.)	R	0
14 to 8	ICHSIZE	These bits indicate the size (in Kbytes) of the instruction cache. 000 0000: No instruction cache 000 0001: 1 Kbyte 000 0010: 2 Kbytes 000 0100: 4 Kbytes 000 1000: 8 Kbytes 001 0000: 16 Kbytes 010 0000: 32 Kbytes 100 0000: 64 Kbytes Other than above: Setting prohibited	R	Note
7 to 4	ICHLINE	These bits indicate the number of lines for each way in the instruction cache. 0000: No instruction cache 0001: 32 lines 0010: 64 lines 0100: 128 lines 1000: 256 lines Other than above: Setting prohibited	R	Note
3 to 0	ICHWAY	These bits indicate the number of ways in the instruction cache. 0000: No instruction cache 0001: 1 way 0010: 2 ways 0100: 4 ways 1000: 8 ways Other than above: Setting prohibited	R	Note

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

(7) ICERR — Instruction cache error

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

After the ICHERR bit is set to 1, any subsequent cache error information that is generated is not stored until this setting is explicitly cleared to 0. In addition, the ICERR register is not updated while cache operation by the CILD instruction is executed.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	5	4	3	2	1	0	
ICERR	CISTW	0	ESMH	ESPBSE	ESTE1	ESTE2	ESDC	ESDE	0	0	ERMMH	ERMPBSE	ERMTE1	ERMTE2	ERMDC	ERMDE	0	ICHEWY		ICHEIX		ICHERQ	ICHED	ICHET	0	ICHERR	Value after reset Undefined

Table 3-62 ICERR Register Contents

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

3.7 Data Buffer Operation Registers

3.7.1 Data Buffer Control System Registers

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

Table 3-63 List of Data Buffer Operation Registers

Register No.	Name	Function	Access Permission
SR 24, 13	CDBCR	Data buffer control register	SV

(1) CDBCR — Data buffer control register

This is the register for controlling data buffer.

	31																												2	1	0	
CDBCR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CDBCLR	CDBEN	Initial value 00000001H

Table 3-64 CDBCR Register Contents

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

CHAPTER 4 EXCEPTIONS AND INTERRUPTS

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

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

Caution This CPU handles interrupts as types of exceptions.

4.1 Outline of Exceptions

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

4.1.1 Exception Cause List

Table 4-1 Exception Cause List

							Priority Order[Note2]		Acknowledgment Condition (PSW)		Update(PSW)				
Exception	Name	Source	Type[Note1]	Saved Resource	Return/ Restoration	Exception Cause Code [Note5]	Priority Level	Priority	ID	NP	UM	ID	NP	EP	EBV
RESET	Reset	Reset input[Note3]	Terminating	—	—	None	1	—	x	x	0	1	0	0	0
FENMI	FENMI interrupt	Interrupt controller[Note3]	Terminating	FE	No	E0H	3	1	x	x	0	1	1	0	s
SYSERR	System error	System error input[Note3]	Terminating	FE	No	10H-1FH [Note3]	3	2	x	x	0	1	1	1	s
FEINT	FEINT interrupt	Interrupt controller[Note3]	Terminating	FE	Yes	F0H	3	3	x	0	0	1	1	0	s
FPI	FPU exception (imprecise)	Execution of an FPU instruction	Terminating	EI	Return: Yes, Restoration: No	72H	4	Note4	0	0	0	1	s	1	s
EIINT0-511	User interrupt	Interrupt controller[Note3]	Terminating	EI	Yes	1000H-11FFH [Note6]	4	Note4	0	0	0	1	s	0	s
MIP	Memory protection exception (execution privilege)	Memory protection violation	Resumable	FE	Yes	90H	10	1	x	x	0	1	1	1	s
SYSERR	System error	Error input during instruction fetch [Note3]	Resumable	FE	No	10H-1FH[Note3]	10	3	x	x	0	1	1	1	s
RIE	Reserved instruction exception	Execution of a reserved instruction	Resumable	FE	Yes	60H	10	4	x	x	0	1	1	1	s
UCPOP	Coprocessor unusable exception	Execution of a coprocessor instruction/ access permission violation	Resumable	FE	Yes	80H-82H [Note9]	10	5	x	x	0	1	1	1	s
PIE	Privilege instruction exception	Execution of a privileged instruction/ access permission violation	Resumable	FE	Yes	A0H	10	6	x	x	0	1	1	1	s
MAE	Misalignment exception	Misaligned access occurrence	Resumable	FE	Yes	C0H	11	Note7	x	x	0	1	1	1	s
MDP	Memory protection exception (access privilege)	Memory protection violation	Resumable	FE	Yes	91H	11	Note7	x	x	0	1	1	1	s
FPP	FPU exception (precise)	Execution of an FPU instruction	Resumable	EI	Yes	71H	11	Note7	x	x	0	1	s	1	s
SYSCALL	System call	Execution of the SYSCALL instruction	Pending	EI	Yes	8000H-80FFH	12	Note8	x	x	0	1	s	1	s
FETRAP	FE level trap	Execution of the FETRAP instruction	Pending	FE	Yes	31H-3FH	12	Note8	x	x	0	1	1	1	s
TRAP0	EI level trap 0	Execution of the TRAP instruction	Pending	EI	Yes	40H-4FH	12	Note8	x	x	0	1	s	1	s
TRAP1	EI level trap 1	Execution of the TRAP instruction	Pending	EI	Yes	50H-5FH	12	Note8	x	x	0	1	s	1	s

Remark s: Retained, x: Not an acknowledgment condition

Notes 1. For details, see **4.1.3 Types of Exceptions**.

2. The acknowledgment priority for exceptions is checked by the priority level, and then priority. A smaller value has a higher priority. For details, see **4.1.4 Exception Acknowledgment Conditions and Priority Order**.

3. For details, see the hardware manual of the product used.

4. The priorities of EIINT0 to EIINT511 and FPI vary depending on the register setting. For details, see **4.1.5 Interrupt Exception Priority and Priority Masking**.

5. The lower 16 bits of the exception cause code are shown. The higher 16 bits of the exception cause code contain the detailed code defined for each exception. These bits are 0000H unless otherwise specified in the description of the function.

6. 1000H to 11FFH (channels 0 to 511) are selected according to the channel.

7. This depends on the operation order of instructions.

8. These exceptions occur exclusively because they occur due to instruction execution. There is no priority within the same priority level.

9. 80H to 82H correspond to the coprocessor use permission (CU0 to CU2), respectively.

4.1.2 Overview of Exception Causes

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

(1) RESET

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

(2) FENMI, FEINT, and EIINT

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

In addition, be sure to place a SYNCP instruction at the start of the exception handlers for these exceptions (in case of EIINT, this only applies if the direct vector method is in use). For details, see **4.2.2 Points for Caution on the Acceptance of Exceptions**.

(3) SYSERR

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

In addition, be sure to place a SYNCP instruction at the start of the exception handler for this exception. For details, see **4.2.2 Points for Caution on the Acceptance of Exceptions**.

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

(4) FPI and FPP

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

In addition, be sure to place the SYNCP instruction at the start of the exception handlers for these exceptions. For details, see **4.2.2 Points for Caution on the Acceptance of Exceptions**.

(5) MIP and MDP

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

(6) RIE

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

(7) PIE

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

(8) UCPOP

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

(9) MAE

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

(10) TRAP, FETRAP, and SYSCALL

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

4.1.3 Types of Exceptions

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

- Terminating exceptions
- Resumable exceptions
- Pending exceptions

(1) Terminating exceptions

This is an exception acknowledged by interrupting an instruction before its operation is executed. These exceptions include interrupts and imprecise exceptions.

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

Caution The PSW.EP bit is cleared to 0 only when an interrupt (INT0 to INT511, FEINT, or FENMI) is acknowledged. It is set to 1 when any other exception occurs.
If an instruction to return execution from the exception handler routine that has been started by generation of an interrupt is executed while the PSW.EP bit is set to 1, the resources on the external interrupt controller might not be released, causing malfunctioning.

If the result of executing the instruction before the interrupted instruction was invalid, there is a delay, and then an imprecise exception occurs. For an imprecise exception, because instructions following the instruction that caused the exception might have already finished executing, resulting in the CPU state at the time of the exception cause not being saved, it is not possible to restore the original processing for re-execution after the processing of this kind of exception.

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

(2) Resumable exceptions

This is an exception acknowledged during the execution of instruction operation before the execution is finished. Because this kind of an exception is correctly acknowledged without executing the next instruction, it is also called a precise exception. General-purpose registers or system registers are not updated because of the occurrence of this exception. The return PC from the exception also points to the instruction where the exception occurred, so execution can be restarted from the state of before the exception occurred.

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

(3) Pending exceptions

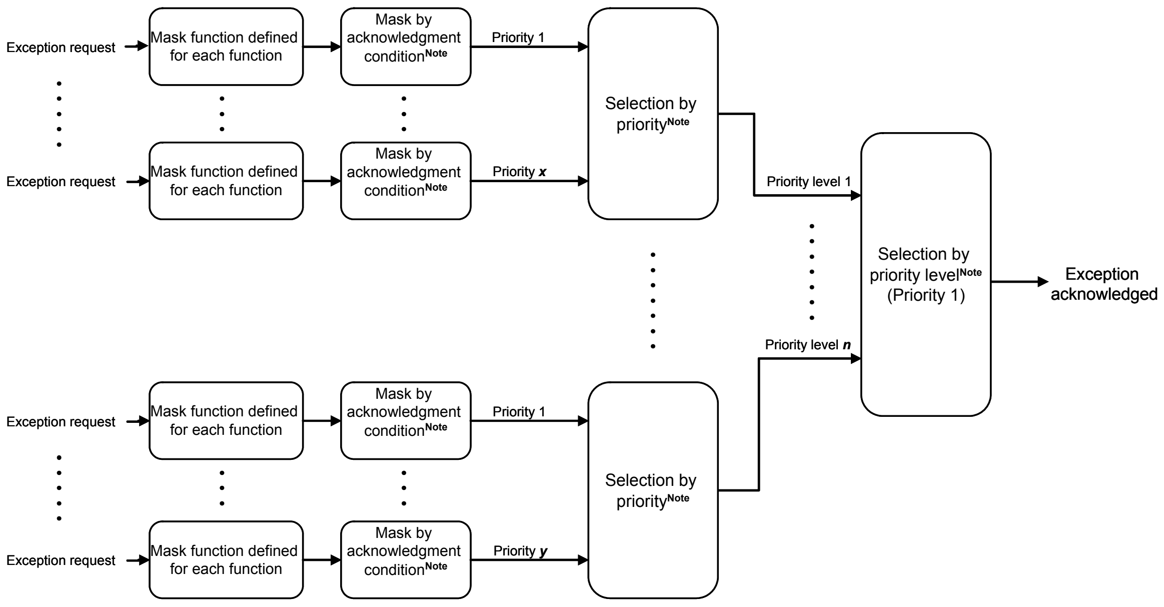
This is an exception acknowledged after the execution of an instruction finishes as a result of executing the instruction operation. Pending exceptions include software exceptions. Because pending exceptions occur as a result of normal instruction execution, the processing resumes with the instruction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

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

4.1.4 Exception Acknowledgment Conditions and Priority Order

The CPU acknowledges only one exception at specific timing based on the exception acknowledgment conditions and priority order. The exception to be acknowledged is determined based on the exception acknowledgment conditions and priority order, as shown in **Figure 4-1** below.

Figure 4-1 Exception Acknowledgment Conditions and Priority Order


Exception request
Mask function defined for each function
Mask by acknowledgment condition Note
Priority 1
Exception request
Mask function defined for each function
Mask by acknowledgment condition Note
Priority *x*
Selection by priority Note
Priority level 1
Selection by priority level Note (Priority 1)
Exception acknowledged
Priority level *n*
Exception request
Mask function defined for each function
Mask by acknowledgment condition Note
Priority 1
Exception request
Mask function defined for each function
Mask by acknowledgment condition Note
Priority *y*
Selection by priority Note

Note See **Table 4-1**.

In **Table 4-1**, an exception with "0" in the acknowledgment condition column can be acknowledged when the corresponding bit is "0." For this kind of exception, acknowledgment is held pending when the corresponding bit is "1." When it changes to "0" and the acknowledgment conditions are met, acknowledgment of the exception becomes possible. If no value is specified for a bit, it is not an acknowledgment condition. If multiple bits are specified as conditions, all the conditions must be met simultaneously.

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

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

For details about acknowledgment conditions, priority level, and priority, see **Table 4-1 Exception Cause List**.

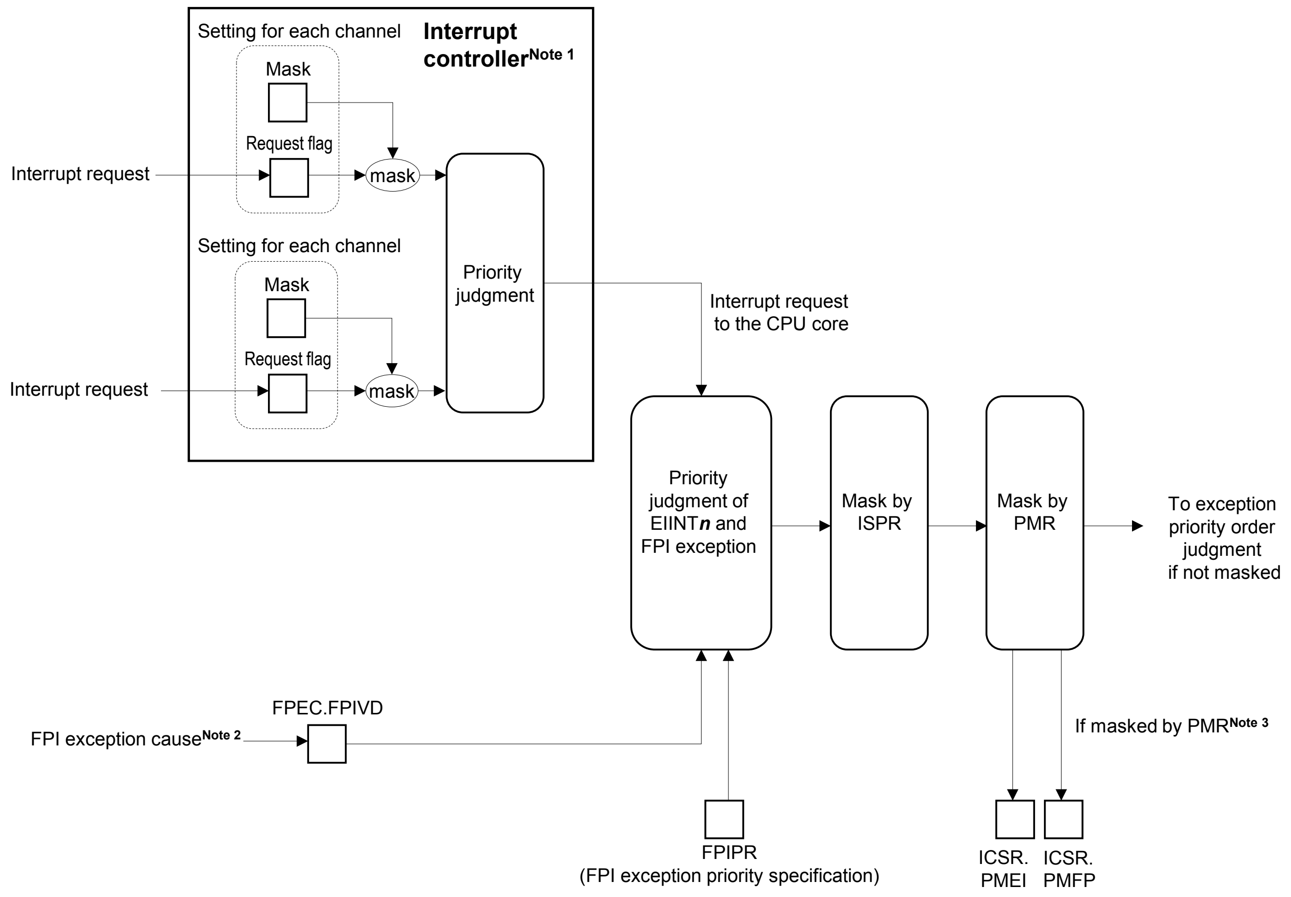
4.1.5 Interrupt Exception Priority and Priority Masking

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

Caution In V850E2 products, the ISPR, PMR, and ICSR registers were defined as functions of the interrupt controller. In this CPU, they are defined as functions of the CPU, but their functions are basically equivalent. Note that there are some differences in functionality.

Figure 4-2 shows an overview of the functions of interrupt exception priority and priority masking.

Figure 4-2 Interrupt Exception Priority and Priority Masking


Setting for each channel
Interrupt controllerNote 1
Mask
Request flag
Interrupt request
mask
Setting for each channel
Mask
Request flag
Interrupt request
mask
Priority judgment
Interrupt request to the CPU core
Priority judgment of EIINTn and FPI exception
Mask by ISPR
Mask by PMR
To exception priority order judgment if not masked
FPEC.FPIVD
FPI exception causeNote 2
If masked by PMRNote 3
FPIPR (FPI exception priority specification)
ICSR. PMEI
ICSR. PMFP

Notes 1. For details about the interrupt controller, see the hardware manual of the product used.

2. An FPI exception cause might occur if it is allowed by the FPU and if imprecise exceptions are specified. For details, see **6.1.5 Floating-Point Calculation Exception** and **6.1.7 Precise Exceptions and Imprecise Exceptions**.

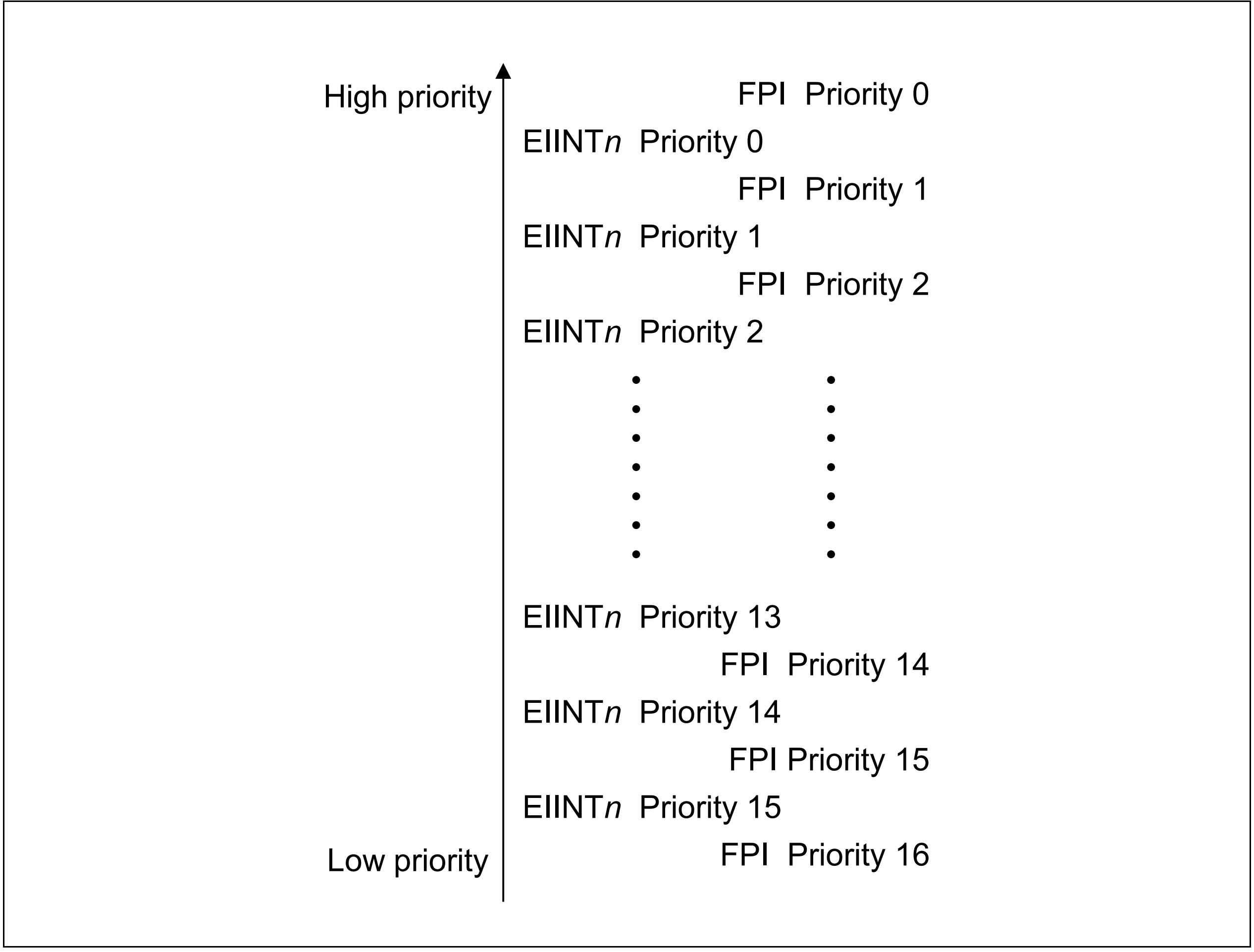
3. The PMEI and PMFP bits in the ICSR register show EIINT*n* or FPI masked by PMR. If EIINT*n* or FPI is masked by the ISPR register or the masking specification of another function before masked by the PMR register, the PMEI and PMFP bits are not affected.

(1) Interrupt priority

For an interrupt (EIINT*n*) and imprecise floating-point exception (FPI), the exception priority can be changed by setting registers. EIINT*n* and FPI are defined with the same priority level, and you can control the priority relationship between EIINT*n* and FPI by changing their exception priorities.

The priority relationship between EIINT*n* and FPI is shown in **Figure 4-3**. If they have the same priority, FPI has precedence. The priority of FPI can be set by using the FPIPR register.

Figure 4-3 Priority Relationship between EIINT*n* and FPI



(2) Interrupt priority mask

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

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

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

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

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

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

(3) Differences in operation between EIINT*n* and FPI

EIINT*n* and FPI behave in the same way up to acknowledgment of an exception. However, their operations partly differ after acknowledgment.

For acknowledgment of an FPI exception, the ISPR register is not updated. As a result, multiple interrupts with a lower priority than the FPI exception might occur when the PSW.ID bit is cleared (0) by the EI instruction during FPI exception handling, releasing the interrupt disabled state.

Generally, an FPI exception is used by setting a higher priority than programs using the FPU. As a result, when an interrupt with a lower priority is acknowledged during an FPI exception, another FPI exception might occur before the FPI exception handling is complete. Therefore, interrupt priority masking must be specified properly by using the PMR register before releasing the interrupt disabled state during an FPI exception.

4.1.6 Return and Restoration

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

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

An exception that cannot be returned or restored from ("No" in **Table 4-1**) might cause the return PC to be lost, making it impossible to return from the exception to the original processing by using a return instruction. An exception whose trigger cannot be selected is an unreturnable or unrestorable exception.
For an unrestorable exception, it is possible to return to the original program flow. However, because the state before the occurrence of the exception cannot be restored at that point, care must be taken in continuing subsequent program operation.

4.1.7 Context Saving

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

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

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

(1) Context saving

Exceptions with certain acknowledgment conditions might not be acknowledged at the start of exception handling, based on the pending bits (PSW.ID and NP bits) that are automatically set when another exception is acknowledged.
To enable processing of multiple exceptions of the same level that can be acknowledged again, certain information about the corresponding return registers and exception causes must be saved, such as to a stack. This information that must be saved is called the “context”.
In principle, before saving the context, caution is needed to avoid the occurrence of exceptions at the same level. The work system registers that can be used for work to save the context, and the system registers that must be at least saved to enable multiple exception handling are called basic context registers. These basic context registers are provided for each level.

Table 4-2 Basic Context Registers

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

4.2 Operation When Acknowledging an Exception

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

<1> Check whether the acknowledgment conditions are satisfied and whether exceptions are acknowledged according to their priority.

<2> Calculate the exception handler address according to the current PSW value[Note 1].

<3> For FE level exceptions, the following processing is performed.

- Saving the PC to FEPC
- Saving the PSW to FEPSW
- Storing the exception cause code in FEIC
- Updating the PSW[Note 2]
- Store the exception handler address calculated in <2> in the PC, and then pass control to the exception handler.

<4> For EI level exceptions, the following processing is performed.

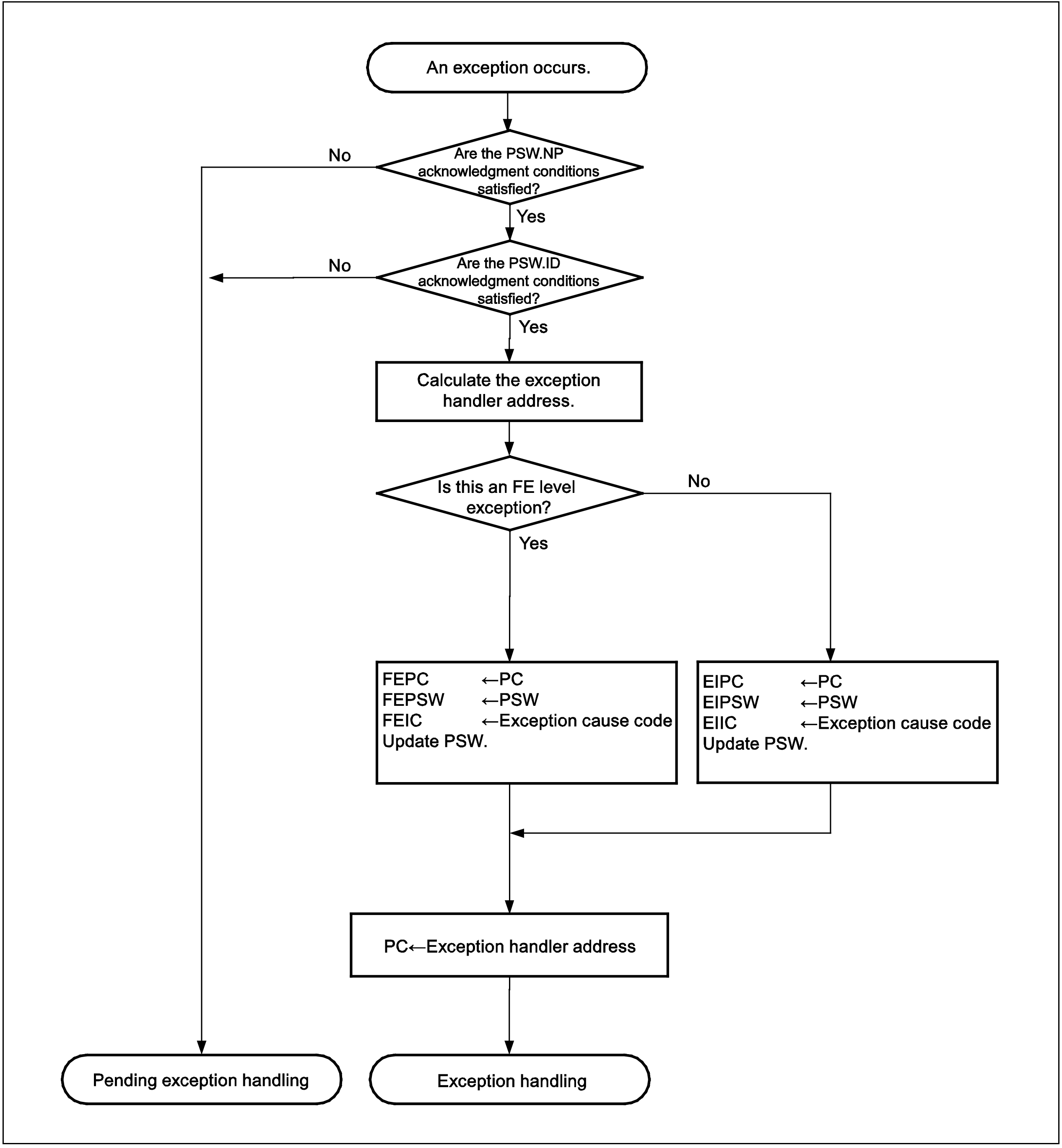
- Saving the PC to EIPC
- Saving the PSW to EIPSW
- Storing the exception cause code in EIIC
- Updating the PSW[Note 2]
- Store the exception handler address calculated in <2> in the PC, and then pass control to the exception handler.

Notes 1. For details, see **4.5 Exception Handler Address**.

2. For the values to be updated, see **Table 4-1 Exception Cause List**.

The following figure shows steps <1> to <4>.

Figure 4-4 Operation When Acknowledging an Exception


An exception occurs.
Are the PSW.NP acknowledgment conditions satisfied?
No
Yes
Are the PSW.ID acknowledgment conditions satisfied?
No
Yes
Calculate the exception handler address.
Is this an FE level exception?
No
Yes
FEPC ←PC
FEPSW ←PSW
FEIC ←Exception cause code
Update PSW.
EIPC ←PC
EIPSW ←PSW
EIIC ←Exception cause code
Update PSW.
PC←Exception handler address
Pending exception handling
Exception handling

4.2.1 Special Operations

(1) EP bit of PSW register

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

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

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

(2) Coprocessor unusable exception

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

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

For details, see **2.4.3 Coprocessor Unusable Exceptions**.

(3) Reserved instruction exception

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

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

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

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

(4) Reset

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

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

For details, see **CHAPTER 8 RESET**.

4.2.2 Points for Caution on the Acceptance of Exceptions

Be sure to place a SNYCP instruction at the start of handers for exceptions of set E listed below. If this is not done, instructions of set B listed below, which are to be executed after an exception of set E has been acknowledged, may not be executed normally.
In cases where it is difficult to place the SYNCP instruction at the start of the handler, the alternative methods listed below are available.

- Place the SYNCP instruction between instructions of set B and the start of the handler for the exception of set E.
- Do not include instructions of set A listed below in programs that may be interrupted by exceptions of set E.

- Instructions of set A:	CALLT, SYSCALL, and SWITCH instructions, and handling in response to EIINT interrupts with the table reference method (not including cases where the table for EIINT interrupts is located at the code flash memory)
- Instructions of set B:	PUSHSP, PREPARE, POPSP, DISPOSE, CALLT, SYSCALL, and SWITCH instructions, and handling in response to EIINT interrupts with the table reference method
- Exceptions of set E:	FENMI, FEINT, EIINT (with the direct vector method), SYSERR, and FPI

4.3 Return from Exception Handling

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

When a context has been saved, such as to a stack, the context must be restored before executing the return instruction.

When execution is returned from an irrecoverable exception, the status before the exception occurs in the original program cannot be restored. Consequently, the execution result might differ from that when the exception does not occur.

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

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

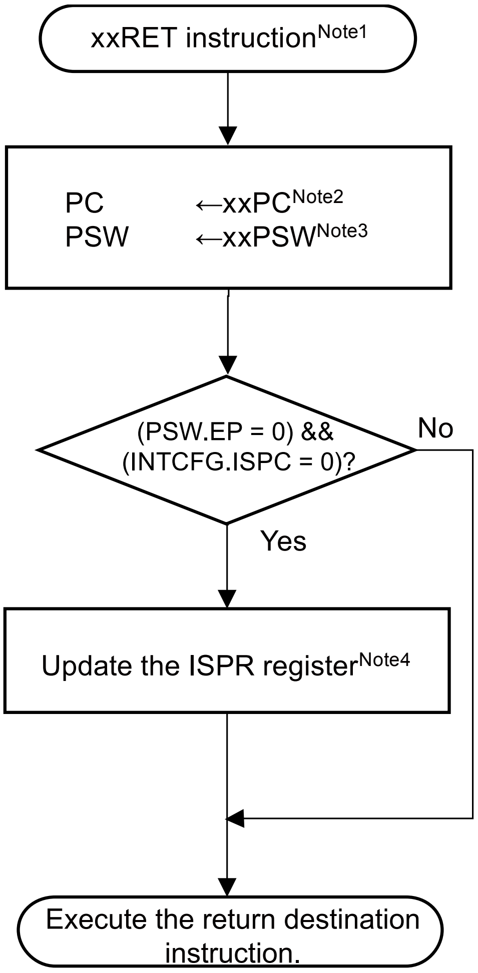
<1> When the EIRET instruction is executed, return PC and PSW are loaded from the EIPC and EIPSW registers. When the FERET instruction is executed, return PC and PSW are loaded from the FEPC and FEPSW registers.

<2> Control is passed to the address indicated by the return PC that were loaded.

<3> When the EIRET instruction is executed while EP = 0 and INTCFG.ISPC = 0, the CPU updates the ISPR register.
When the FERET instruction is executed, the CPU does not update the ISPR register.

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

Figure 4-5 Return Instruction-Based Exception Return Flow



Notes 1. It is the EIRET instruction when returning from an EI level exception, or the FERET instruction when returning from an FE level exception.

2. It is EIPC when returning from an EI level exception, or FEPC when returning from an FE level exception.

3. It is EIPSW when returning from an EI level exception, or FEPSW when returning from an FE level exception.

4. Only for the EIRET instruction.

4.4 Exception Management

This CPU has the following functions to manage exceptions in order to prevent mutual interference between programs during multi-programming.

- Exception synchronization instruction (SYNCE)
- Function to check pending exception
- Function to cancel pending exception

This CPU defines imprecise exceptions that have a delay time until the exception handling is started after the cause of the exception has been generated.
This CPU has an exception management function to wait for all exceptions caused by a program before the program is changed or terminated, so that the exceptions are sequentially processed. This prevents the influence of illegal processing of a certain program from reaching the other programs. It also prevents termination processing of a program from being completed without the exceptions being processed.

4.4.1 Exception Synchronization Instruction

Imprecise exceptions can be synchronized using the SYNCE instruction. In this CPU, this is equivalent to an imprecise floating-point operation exception (FPI). To acknowledge imprecise exceptions at any time, perform the following procedure.

(1) Mask the acknowledgment conditions of the imprecise exception to be acknowledged (by clearing PSW.ID and NP).

(2) Execute the exception synchronization instruction (SYNCE). At this point, all the imprecise exceptions that are generated by the instructions preceding the SYNCE instruction have always been reported to the CPU. However, acknowledging an exception might be masked by the acknowledgment condition set in (1) and the exception might have been held pending.

(3) As a result of (2), an exception that is not masked is acknowledged. If there are two or more sources of exceptions, the exceptions are sequentially acknowledged in accordance with their priority.

4.4.2 Checking and Canceling Pending Exception

To check if there is an exception that is held pending, follow this procedure.

(1) Set a mask so that the acknowledgment conditions of the imprecise exception to be checked are not satisfied (by setting PSW.ID and NP).

(2) Execute the exception synchronization instruction (SYNCE). At this time, all the imprecise exceptions that are generated by the instructions preceding the SYNCE instruction have always been reported to the CPU. The exception to be checked is not acknowledged but held pending because of the mask set in (1). However, the other exceptions might be acknowledged.

(3) Read the exception report bit of the exception to be checked. If the bit is 1, the exception has been held pending.

(4) Clear the mask set in (1) as necessary.

To not acknowledge but cancel a pending exception without executing exception handling, follow this procedure.

(1) Set a mask so that the acknowledgment conditions of the imprecise exception to be canceled are not satisfied (by setting PSW.ID and NP).

(2) Execute the exception synchronization instruction (SYNCE). At this time, all the imprecise exceptions that are generated by the instructions preceding the SYNCE instruction have always been reported to the CPU. The exception to be canceled is not acknowledged but held pending because of the mask set in (1). However, the other exceptions might be acknowledged.

(3) Clear the exception report bit of the exception to be canceled.

(4) Perform waiting processing until cancellation is completed. The instruction sequence of the waiting processing is defined as the hardware specifications. See the hardware manual of the product used.

(5) When cancellation has been completed, clear the mask set in (1) as necessary.

The function to cancel each exception is provided by the following registers.

Table 4-3 Checking and Canceling Pending Exception

Exception	Cause	Canceling Bit	Remark
FPI exception	FPU instruction	FPIVD bit in the FPEC register	If this bit is cleared, disabling the succeeding FPU instruction is canceled.

4.5 Exception Handler Address

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

4.5.1 Resets, Exceptions, and Interrupts

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

Caution 1. The exception handler address of EIINT*n* selected using the direct vector method differs from that of V850E2 products. In V850E2 products, a different exception handler address is individually assigned to each interrupt channel (EIINT*n*). In this CPU, one exception handler address is assigned to each interrupt priority. Consequently, interrupts that have the same priority level branch to the same exception handler.

(1) Direct vector method

The CPU uses the result of adding the exception cause offset shown in **Table 4-4 Selection of Base Register/Offset Address** to the base address indicated by the RBASE or EBASE register as the exception handler address.

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

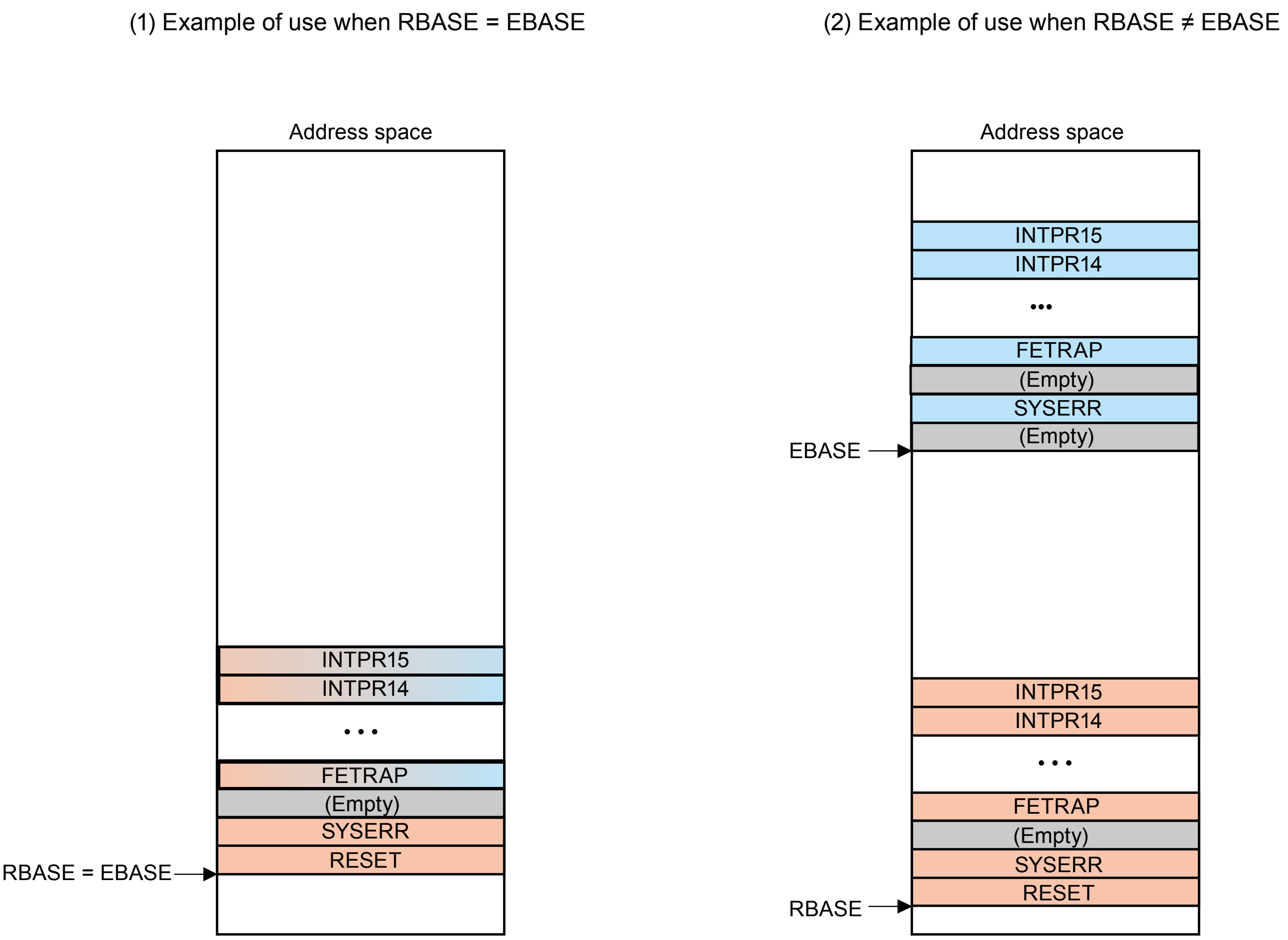
However, reset input and some exceptions[Note 2] always refer to the RBASE register.

In addition, user interrupts refer to the RINT bit of the corresponding base register, and reduce the offset address according to the bit status. If the RBASE.RINT bit or EBASE.RINT bit is set to 1, all user interrupts are handled using an offset of 100H. If the bit is cleared to 0, the offset address is determined according to **Table 4-4 Selection of Base Register/Offset Address**.

Notes 1. Exception acknowledgment itself sometimes updates the status of the PSW.EBV bit. In this case, the base register is selected based on the new bit value. For details, see **4.5 Exception Handler Address**.

2. The exceptions that always reference RBASE are determined according to the hardware specifications.

Figure 4-6 Direct Vector Method



Remark INTPRx is the same as EIINT*n* (priority x) in **Table 4-4 Selection of Base Register/Offset Address**.

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

Table 4-4 Selection of Base Register/Offset Address

	PSW.EBV = 0	PSW.EBV = 1	RINT = 0	RINT = 1
	Base Register		Offset Address	
RESET	RBASE	None[Note 1]	000H	000H
SYSERR[Note 3]		EBASE	010H	010H
FETRAP			030H	030H
TRAP0			040H	040H
TRAP1			050H	050H
RIE			060H	060H
FPP/FPI[Note 3]			070H	070H
UCPOP			080H	080H
MIP/MDP			090H	090H
PIE			0A0H	0A0H
Debug[Note 2]			0B0H	0B0H
MAE			0C0H	0C0H
(R.F.U.)			0D0H	0D0H
FENMI[Note 3]			0E0H	0E0H
FEINT[Note 3]			0F0H	0F0H
EIINT*n* (priority 0)[Note 3]			100H	100H
EIINT*n* (priority 1)[Note 3]			110H	
EIINT*n* (priority 2)[Note 3]			120H	
EIINT*n* (priority 3)[Note 3]			130H	
EIINT*n* (priority 4)[Note 3]			140H	
EIINT*n* (priority 5)[Note 3]			150H	
EIINT*n* (priority 6)[Note 3]			160H	
EIINT*n* (priority 7)[Note 3]			170H	
EIINT*n* (priority 8)[Note 3]			180H	
EIINT*n* (priority 9)[Note 3]			190H	
EIINT*n* (priority 10)[Note 3]			1A0H	
EIINT*n* (priority 11)[Note 3]			1B0H	
EIINT*n* (priority 12)[Note 3]			1C0H	
EIINT*n* (priority 13)[Note 3]			1D0H	
EIINT*n* (priority 14)[Note 3]			1E0H	
EIINT*n* (priority 15)[Note 3]			1F0H	

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

2. The exception for debug function.

3. Be sure to place a SYNCP instruction at the start of handlers for these exceptions. For details, see **4.2.2 Points for Caution on the Acceptance of Exceptions**.

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

(2) Table reference method

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

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

<1> In any of the following cases, the exception handler address is determined by using the direct vector method.

- When PSW.EBV = 0 and RBASE.RINT = 1
- When PSW.EBV = 1 and EBASE.RINT = 1
- When the interrupt channel setting is not the table reference method

<2> In cases other than <1>, calculate the table reference position.
Exception handler address read position = INTBP register + channel number * 4 bytes

<3> Read word data starting at the interrupt handler address read position calculated in <2>.

<4> Use the word data read in <3> as the exception handler address.

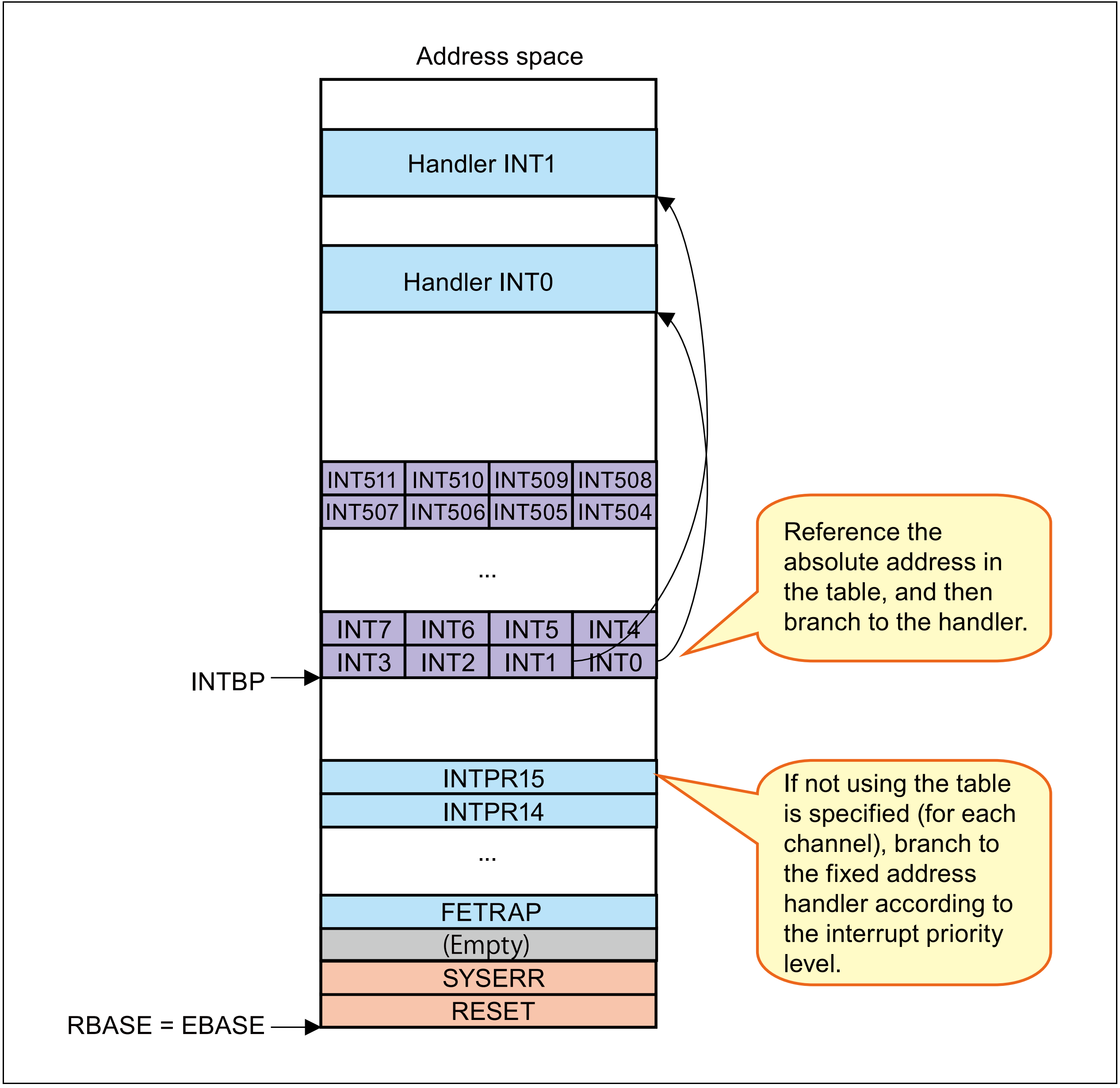
Caution For details about the interrupt channel settings, see the hardware manual of the product used.

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

Table 4-5 Exception Handler Address Expansion

Type	Exception Handler Address Read Position
EIINT interrupt channel 0	INTBP + 0 * 4
EIINT interrupt channel 1	INTBP + 1 * 4
...	...
EIINT interrupt channel 510	INTBP + 510 * 4
EIINT interrupt channel 511	INTBP + 511 * 4

Figure 4-7 Overview of Using the Table Reference Method


Address space
Handler INT1
Handler INT0
INT511
INT510
INT509
INT508
INT507
INT506
INT505
INT504
...
INT7
INT6
INT5
INT4
INT3
INT2
INT1
INT0
INTBP
INTPR15
INTPR14
...
FETRAP
(Empty)
SYSERR
RESET
RBASE = EBASE
Reference the absolute address in the table, and then branch to the handler.
If not using the table is specified (for each channel), branch to the fixed address handler according to the interrupt priority level.

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

4.5.2 System Calls

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

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

Table 4-6 System Calls

Vector	Exception Cause Code	Referenced Table Entry
0	0000 8000H	Table entry 0
1	0000 8001H	Table entry 1
2	0000 8002H	Table entry 2
...	...	...
n – 1	0000 8000H + (n – 1)H	Table entry n – 1
n	0000 8000H + nH	Table entry n
n + 1	0000 8000H + (n + 1)H	Table entry 0
...	...	...
254	0000 80FEH	Table entry 0
255	0000 80FFH	Table entry 0

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

4.5.3 Models for Application

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

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

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

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

Notes 1. Normally, an EIRET or FERET instruction should be used to set the PSW.EBV bit to 1.

2. The hardware specifications determine which exception has which cause, and whether or not an exception is needed to clear PSW.EBV to 0.

Figure 4-8 Example of Model for Application (Operation Flow)

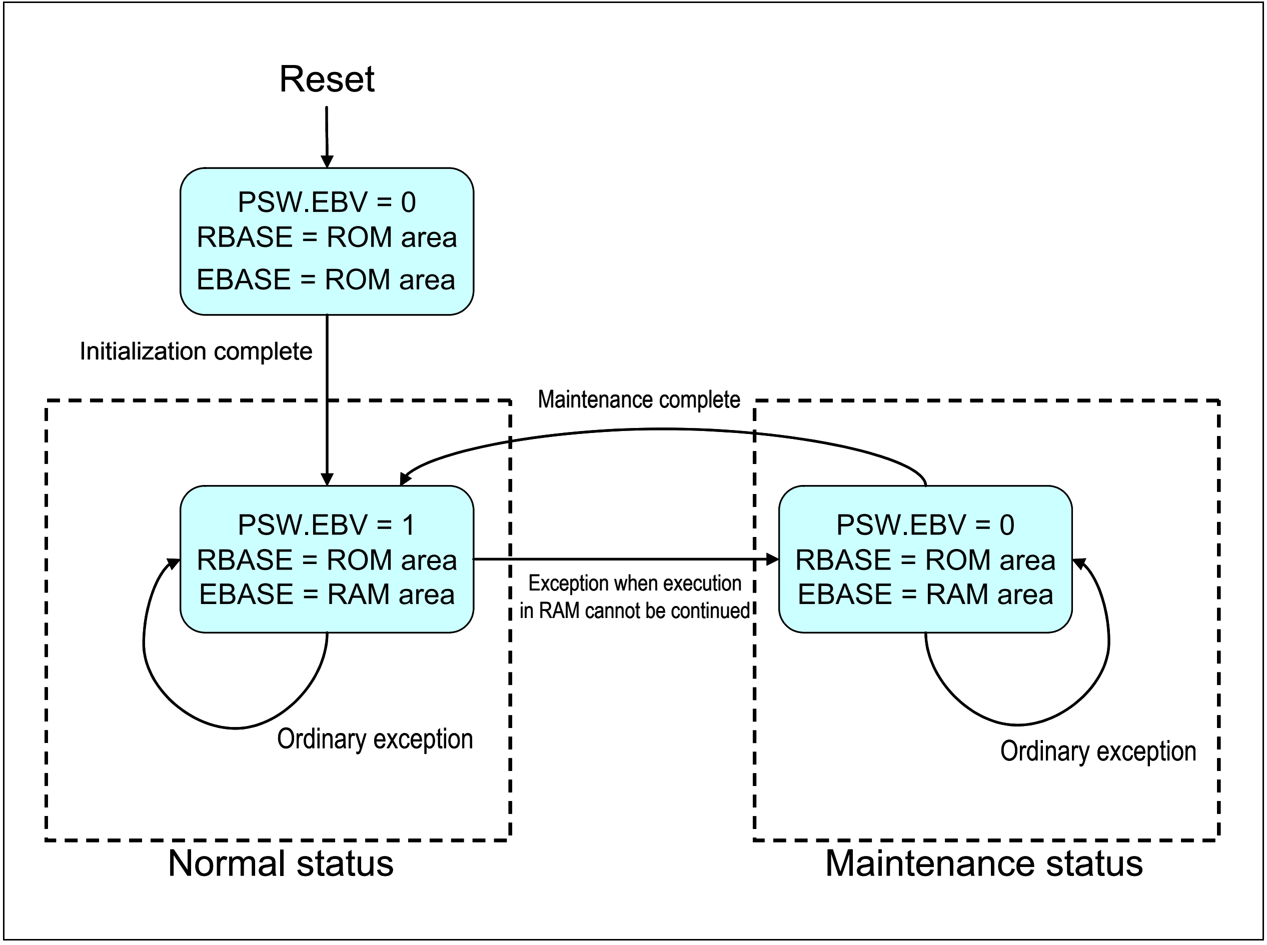
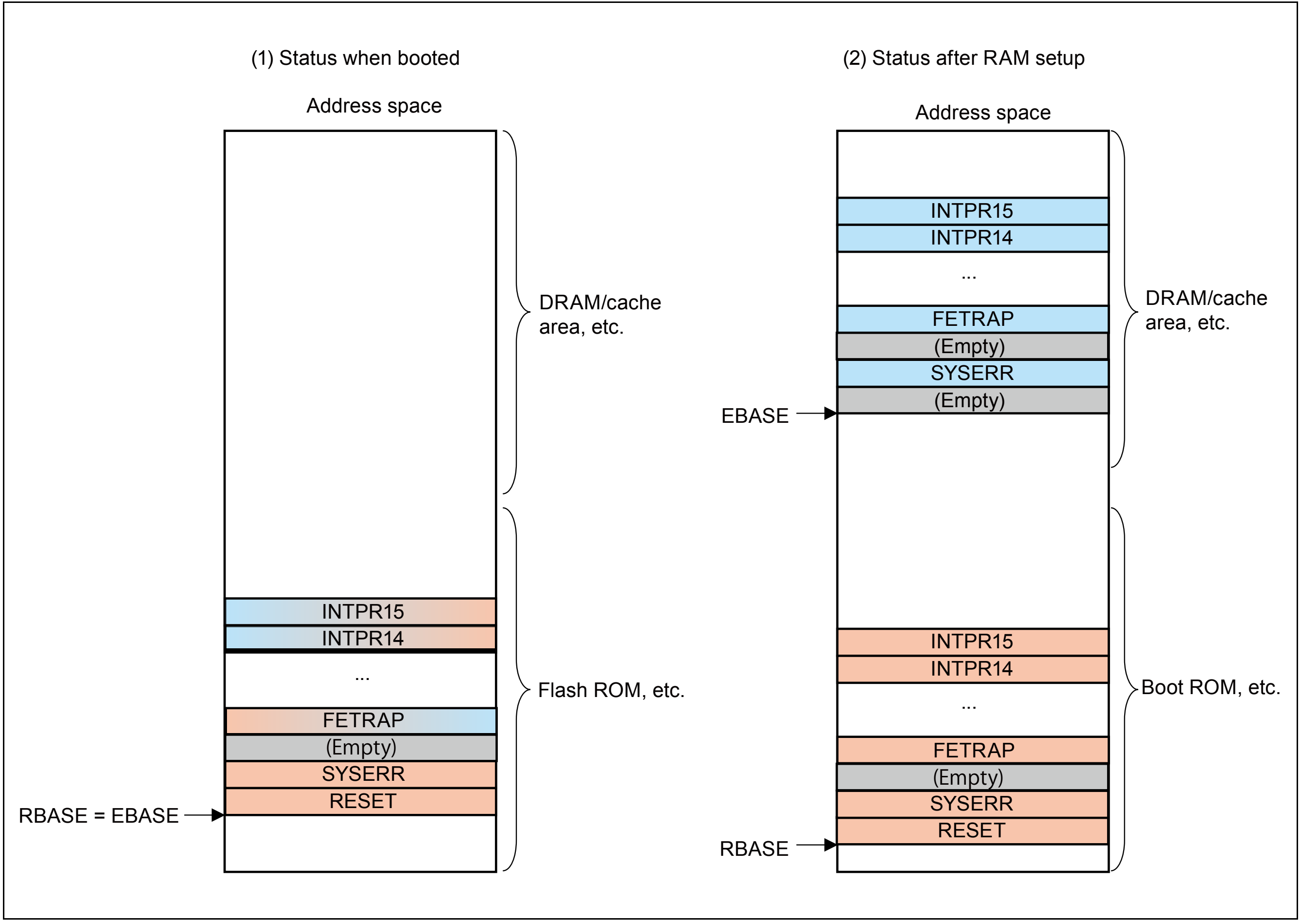


Figure 4-9 Example of Model for Application (Address Map)



CHAPTER 5 MEMORY MANAGEMENT

This CPU provides the following functions for managing the memory.

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

5.1 Memory Protection Unit (MPU)

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

5.1.1 Features

(1) Memory access control

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

(2) Access management for each CPU operation mode

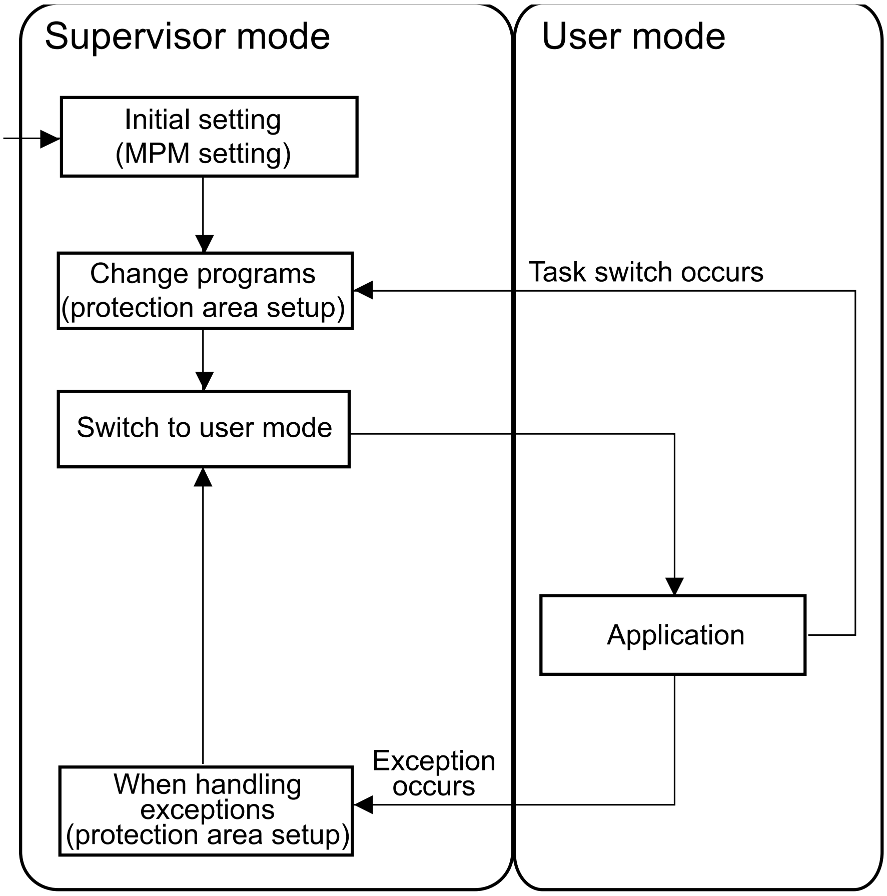
In this CPU, several status bits are used to control access to resources, and these bits are used in combination to perform protection that is appropriate, according to each program's level of reliability.

5.1.2 MPU Operation Settings

Before using a protection area, set up operation of the MPU function in supervisor mode. Normally, it is assumed that this setting is performed by management software, such as the OS.

Settings in supervisor mode fall into three types: initial settings, settings to change programs, and settings that are changed when handling exceptions. The processing flow is illustrated below.

Figure 5-1 Example of MPU Processing Flow


Supervisor mode
User mode
Initial setting (MPM setting)
Change programs (protection area setup)
Task switch occurs
Switch to user mode
Application
Exception occurs
When handling exceptions (protection area setup)

The initial settings are set as appropriate values in the MPM register. Always use the MPE bit to validate the MPU.

The SVP bit should be set to 1 only when protection is also being performed by a supervisor such as an OS.

Caution Perform the following procedures in advance only when the SVP bit will be set to 1.

- Before setting (to 1) the SR, SW, or SX bit in the protection area, correctly set up the MPUAn and MPLAn registers in the same protection area.
- No procedures are necessary if the SR, SW, and SX bits will not be set to 1.
- Note with caution that when the SVP bit is set to 1, the management program (OS, etc.) that sets the SVP itself cannot be executed. If a setting error is made, continued execution might become impossible due to recursive occurrence of MIP or MDP exceptions.

When switching programs, the protection area for the target program might need to be set up. For details about protection area settings, see **5.1.3 Protection Area Settings**.

During exception handling, unlike processing that sets a recovery as part of ordinary error processing, a management program determines whether or not the address where the exception occurred can be used and, when demand paging is performed to continue execution, the protection area might be changed.

As when programs are switched, protection area settings are changed, as described in **5.1.3 Protection Area Settings** below.

5.1.3 Protection Area Settings

(1) Protection area settings

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

Some additional description is provided below regarding certain caution points.

(a) E bit

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

(b) UX, UR, and UW bits

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

(c) SX, SR, and SW bits

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

(d) G bit and ASID field

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

5.1.4 Caution Points for Protection Area Setup

(1) Crossing protection area boundaries

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

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

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

(2) Invalid protection area settings

Protection area settings are invalid in the following case.

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

Caution Note, however, that addresses are handled as unsigned integers (0H to FFFFFFFFH).

5.1.5 Access Control

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

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

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

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

Cautions 1. Even when access is enabled, there might be cases where access is blocked by another function that prohibits it.

2. In some cases, access judged to be prohibited may be executed for a memory or I/O device. The cases are as listed below.

   ・Reading local RAM

   ・Reading of code flash memory by an instruction prefetched from the instruction cache

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

5.1.6 Violations and Exceptions

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

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

(1) Execution protection violation (MIP exception)

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

(2) Data protection violation (MDP exception)

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

(3) Exception cause code and exception address

When an instruction protection violation or data protection violation has been detected, the exception cause code is determined as shown in **Table 5-1**. The determined exception cause code is set to the FEIC register.
The MEA register is used to store either the PC of the instruction that detected the instruction protection violation or the access address used when the data protection violation occurred. The MEA register is shared in order to prevent simultaneous occurrence of MIP and MDP exceptions. Also, when a data protection violation occurs, the information of the instruction that caused the violation is stored in the MEI register.

Table 5-1 Exception Cause Code of Memory Protection Violation

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

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

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

3. This bit is set to 1 when a violation is caused by the SET1, NOT1, CLR1, or CAXI instruction.

4. This bit is set to 1 when a violation is caused by the PREPARE, DISPOSE, PUSHSP, or POPSP instruction.

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

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

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

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

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

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

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

RMW: Set to 1 when the instruction causing the violation contains a read-modify-write operation (SET1, NOT1, CLR1, or CAXI).

BL: Set to 1 when the instruction causing the violation performs a block transfer (PREPARE, DISPOSE, PUSHSP, or POPSP).

MS: Set to 1 when the instruction causing the violation performs a misaligned access.

5.1.7 Memory Protection Setting Check Function

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

(1) Procedure

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

Cautions 1. If the specified area to be checked crosses 00000000H or 7FFFFFFFH, it is judged as an area setting error, and the MCR.OV bit is set to 1. This means that the MCR.OV bit must be checked to access the check results. Do not use the check result until it is confirmed that the result is not invalid (OV = 0).

2. If the default operations specified by using the MPM.DX, DW, and DR bits are enabled (1), the correct result might not be able to be obtained. If enabling the specified default operation, do not use the memory protection setting check function.

(2) Sample code

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

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

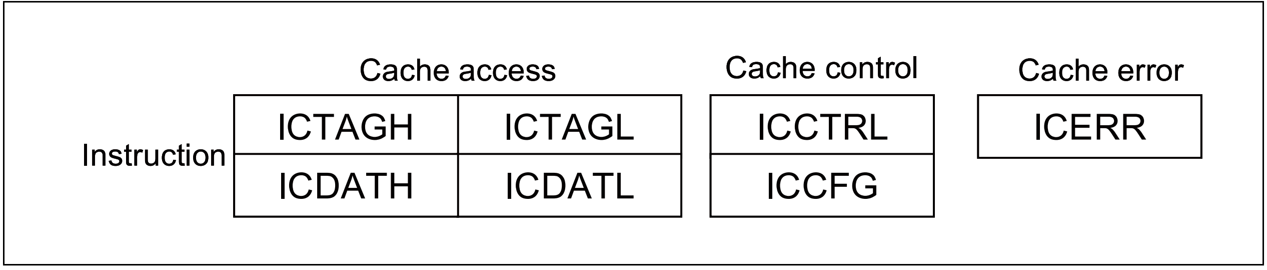
5.2 Cache

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

5.2.1 Cache Operation Registers

Figure 5-2 shows the system registers for cache operation. The supervisor privilege is required for the operation.

Figure 5-2 Cache Operation Registers



5.2.2 Change Cache Use Mode

(1) Change use mode of instruction cache

When an instruction cache has been mounted, execution of instructions can be made more efficient by enabling an instruction cache. By contrast, if the instruction code is placed in RAM and will be overwritten during processing, it might be more advisable to not use an instruction cache.

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

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

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

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

5.2.3 Cache Operations using CACHE Instruction

The CACHE instruction manipulates data in the specified cache memory.

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

(1) Specification method for target of CACHE instruction

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

- Directly specify the address to be accessed:

In this CPU, this is called the address specification method. In this case, the cache line containing the specified address is subject to operation.

- Directly specify the cache memory's way number and line number:

In this CPU, this is called the index specification method. In this case, no hit judgment for the cache is performed, and the operation is performed on the specified cache index. For details about the cache index specification method, see **5.2.5 Cache Index Specification Method**.

(2) Operations performed using the CACHE instruction

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

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

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

(b) Cache Fetch And Lock (CFAL)

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

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

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

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

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

(d) Other operations

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

5.2.4 Cache Operation when the PREF Instruction is Executed

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

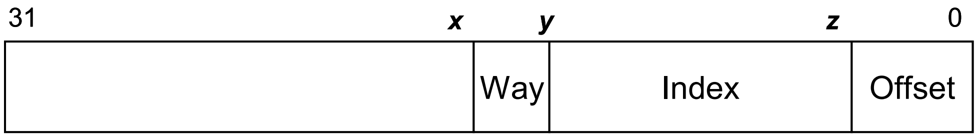
Assuming support by compilers and other tools, the PREF instruction can be executed regardless of the CPU operating mode. Execution of the PREF instruction does not cause an exception generated by the MPU, and has no effect on logical operations, just like a NOP instruction.

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

5.2.5 Cache Index Specification Method

For a cache instruction that uses the index specification method, explicitly specify the cache memory subject to operation in the format shown in **Figure 5-3**, instead of specifying an address. The bit positions (x, y, z) of each field depend on the size of the cache memory incorporated in the CPU core. Information about the incorporated cache memory and size can be read from the ICCFG register.

Figure 5-3 Cache Memory Index Specification Method



Remark Way: Specify the way within the cache.

Index: Specify the cache index.

Offset: Specify the offset within the cache line.

Caution Bits 31 to x: Be sure to clear to 0.

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

5.2.6 Execution Privilege of the CACHE/PREF Instruction

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

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

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

(a) Operations allowed with the user privilege

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

(b) Operations requiring the supervisor privilege

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

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

5.2.7 Memory Protection for CACHE and PREF Instructions

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

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

Table 5-2 shows the correspondence between operations and access privileges.

Table 5-2 Relationship Between Cache Operations and Permissions

Instruction	Target	Address/Index	Instruction Execution Privilege	Access Permission
CHBII	Instruction	Address	UM	Read
CIBII	Instruction	Index	SV	—
CFALI	Instruction	Address	SV	Read
CISTI	Instruction	Index	SV	—
CILDI	Instruction	Index	SV	—
(CLL instruction)[Note]	—	—	—	—

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

5.3 Mutual Exclusion

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

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

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

5.3.1 Shared Data that does not Require Mutual Exclusion Processing

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

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

With some exceptions, mutual exclusion is achieved by using these types of data access. In other words, it is guaranteed that while one CPU is executing the instructions that perform the above data accesses, another CPU is not accessing the data in question. This is known as an instruction being executed atomically or an instruction providing an atomic guarantee. Note that the atomic execution of an instruction means that a data access bus transaction completes with no disruption; it does not necessarily mean that a series of transactions has been completed.

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

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

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

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

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

(1) Link

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

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

Links are deleted when certain event or address conditions are satisfied. **Table 5-3** shows the link deletion conditions. A link is deleted if any of the conditions shown in **Table 5-3** are satisfied.

Table 5-3 Link Deletion Conditions

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

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

(2) Sample code

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

Lock acquisition

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

Lock release

```
          st.w    r0, 0[r20]
```

5.3.3 Performing Mutual Exclusion by Using the SET1 Instruction

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

Cautions 1. Depending on the hardware specifications, the system performance might drop if exclusive control is executed frequently by using the SET1 instruction, because this causes the bus to be occupied for a long time. It is therefore recommended to execute exclusive control by using the LDL/STC instructions as much as possible.

2. When performing mutual exclusion by using the SET1 instruction, to prevent the problem of excessive bus occupancy described in Caution 1 above, execute the snooze instruction before attempting to acquire a lock again after lock acquisition has failed, and adjust the lock acquisition loop execution interval.

(1) Sample code

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

Lock acquisition

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

Lock release

```
  clr1 0, 0[r20]
```

5.3.4 Performing Mutual Exclusion by Using the CAXI Instruction

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

Cautions 1. Depending on the hardware specifications, the system performance might drop if exclusive control is executed frequently by using the CAXI instruction, because this causes the bus to be occupied for a long time. It is therefore recommended to execute exclusive control by using the LDL/STC instructions as much as possible.

2. When performing mutual exclusion by using the CAXI instruction, to prevent the problem of excessive bus occupancy described in Caution 1 above, execute the snooze instruction before attempting to acquire a lock again after lock acquisition has failed, and adjust the lock acquisition loop execution interval.

(1) Sample code

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

Lock acquisition

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

Lock release

```
       st.w  r0, 0[r20]
```

5.4 Synchronization Function

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

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

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

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

The SYNCE instruction is the special instruction, which synchronizes all preceding imprecise exceptions (FPI exceptions). Execute the SYNCE instruction when all preceding FPI exceptions need to be accepted. The SYNCE instruction can be used to guarantee completion of exception handling by the preceding task before a task is changed or terminated in a multi-processing environment.

Table 5-4 shows the effect of the synchronization instructions.
For the hazard resolution procedure for system registers, see **APPENDIX A, HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS**.

Table 5-4 Effect of Synchronization Instructions

SYNC Instruction	Synchronization Guaranteed by the SYNC Instruction					
	Synchronization of Instruction Fetch		Synchronization of Execution of the Preceding Instruction			
	Re-fetch of Subsequent Instructions	Cache Instruction/Instruction to Update Cache Operation Function Register	Calculation Instruction	Load Instruction	Store Instruction	FPI Exception
SYNCP	—	—	Completion of execution	Completion of execution Note 1	—	—
SYNCM	—	—	Completion of execution	Completion of execution Note 1	Completion of execution Note 2	—
SYNCI	Re-fetch after synchronization of execution of the preceding instruction	Completion of execution Note 3	Completion of execution	—	—	—
SYNCE	—	—	—	—	—	Acceptance of exception

Remark: "—": Not guaranteed

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

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

3. When completion of instruction cache clearance is confirmed, check the read value of the ICCTRL.ICHCLR bit.

CHAPTER 6 COPROCESSOR

6.1 Floating-Point Operation

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

Either single-precision (32-bit) or double-precision (64-bit) data can be used. In addition, floating-point values and fixed-point values can be converted.

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

(1) Floating-point instructions

This FPU inherits the V850EFC and VR Series floating-point instruction sets, so software properties from both the V850EFC and VR Series can be easily ported.

- Instructions equivalent to the floating-point instructions available in the VR Series are supported.
- The following four V850EFC instructions are supported.
  MAXF.S, MINF.S, MAXF.D, and MINF.D
- Supports the flag transfer instruction which transfers the floating-point configuration/status register’s condition bits to the Z flag of the PSW register.
  TRFSR
- Supports conditional move instruction, to accelerate conditional branch.
  CMOVF.S, CMOVF.D
- Supports unsigned conversion instructions which efficiently execute format conversions with unsigned integers.
- Supports the CEIL and FLOOR instructions, which efficiently execute conversion of the format to the nearest integer.
- Supports fused-multiply-add instructions that execute multiply-add operations with high accuracy.
- Supports half-precision floating-point format conversion instructions for storing data efficiently.

Supports condition bits (8 bits) for storing floating-point comparison results.

Supports two FPU execution modes: precise mode and imprecise mode

(2) Register set

- Floating-point operation registers: Uses general-purpose registers
  (not special-purpose register for floating-point operations)
- Floating-point system registers:
  FPSR – Floating-point configuration/status
  FPEPC – Floating-point exception program counter
  FPST – Floating-point status
  FPCC – Floating-point comparison result
  FPCFG – Floating-point configuration
  FPEC – Floating-point exception control

6.1.1 Configuration of Floating -Point Operation Function

(1) Not implemented

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

(2) Implementing only single precision

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

Notes 1. The single-precision floating-point instruction is the instruction described as (Single) in the description of each instruction in **7.4.4 Floating-Point Instruction Set**.

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

(3) Implementing single precision and double precision

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

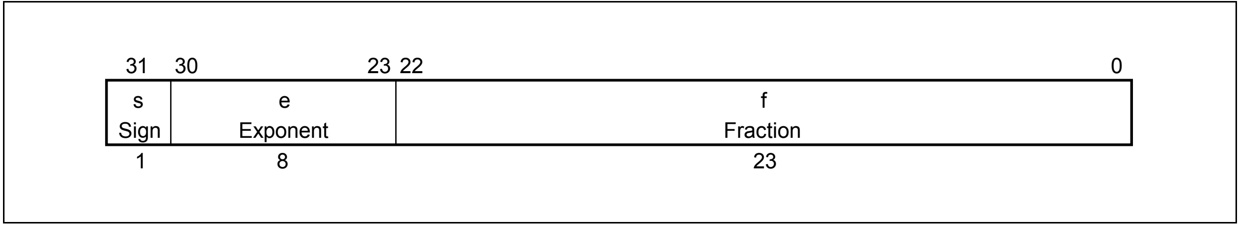
6.1.2 Data Types

(1) Floating-point format

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

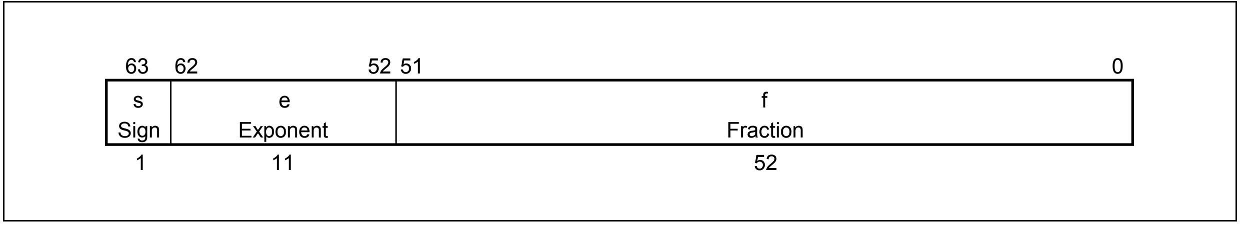
The single-precision floating-point format consists of a 24-bit signed fraction (s + f) and an 8-bit exponent (e), as shown in **Figure 6-1**.

Figure 6-1 Single-precision Floating-point Format



The double-precision floating-point format consists of a 53-bit signed fraction (s + f) and an 11-bit exponent (e), as shown in **Figure 6-2**.

Figure 6-2 Double-precision Floating-Point Format



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

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

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

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

The numeric value (v) represented in this format can be calculated by the expression shown in **Table 6-1**.

Table 6-1 Calculation Expression of Floating-Point Value

Type	Calculation Expression	
NaN (not-a-number)	If $E = E_{max} + 1$ and $f \neq 0$	then v = NaN regardless of s
±∞ (infinite number)	If $E = E_{max} + 1$ and $f = 0$	then $v = (-1)^s\infty$
Normalized number	If $E_{min} \leq E \leq E_{max}$	then $v = (-1)^s2^E$ (1.f)
Subnormal number	If $E = E_{min} - 1$ and $f \neq 0$	then $v = (-1)^s2^{Emin}$ (0.f)
±0 (zero)	If $E = E_{min} - 1$ and $f = 0$	then $v = (-1)^s0$

- NaN (not-a-number)

IEEE754 defines a floating-point value called NaN (not-a-number). Because this value is not a numerical value, it does not have any “greater than” or “less than” relationships to other values.

If v is NaN in all of the floating-point formats, it might be either SignalingNaN (S-NaN) or QuietNaN (Q-NaN), depending on the value of the most significant bit of f. If the most significant bit of f is set, v is QuietNaN; if the most significant bit is cleared, it is SignalingNaN.

Table 6-2 shows the value of each parameter defined in floating-point formats.

Table 6-2 Floating-Point Formats and Parameter Values

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

Table 6-3 shows the minimum and maximum values that can be represented in floating-point formats.

Table 6-3 Floating-Point Minimum and Maximum Values

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

(2) Fixed-point formats

The value of a fixed point is held in the format of 2’s complement. **Figure 6-3** shows a 32-bit fixed-point format and **Figure 6-4** shows a 64-bit fixed-point format. No signed bits exist in the unsigned fixed-point format, and all bits represent the integer value.

Figure 6-3 32-bit Fixed-Point Format

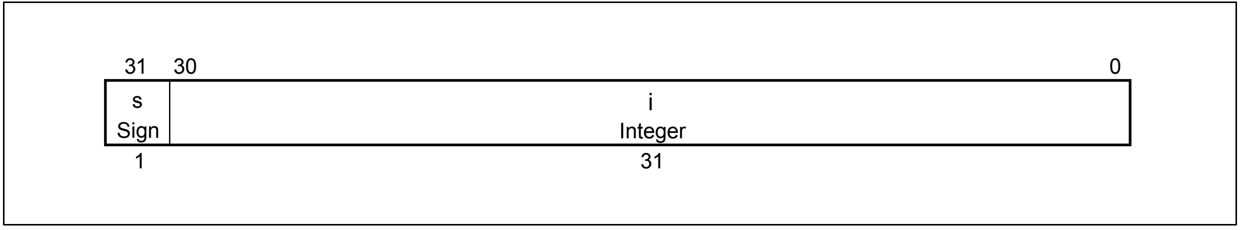
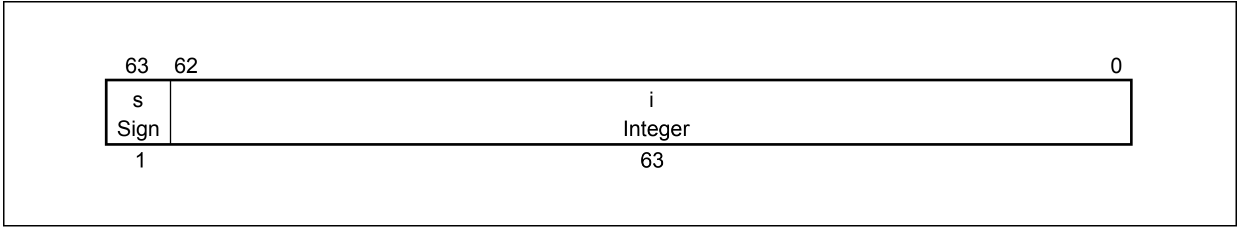


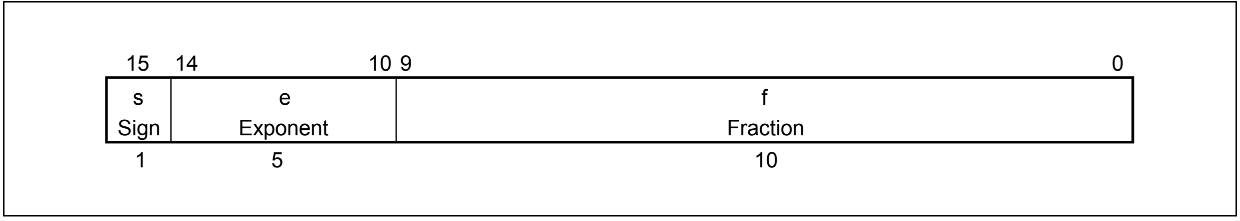
Figure 6-4 64-bit Fixed-Point Format



(3) Expanded floating-point format

This CPU supports the 16-bit (half-precision) IEEE754 floating-point format as a floating-point format for storing data. The half-precision floating-point format is used to decrease the amount of data; it is not supported for arithmetic operations. Instructions are available for converting single-precision floating-point format data into half-precision floating-point data and vice-versa. The half-precision floating-point format consists of an 11-bit signed fraction (s + f) and a 5-bit exponent (e), as shown in **Figure 6-5**.

Figure 6-5 Half-Precision Floating-Point Format



Like other floating-point formats, the numeric values represented in this format can be calculated by using the expressions shown in **Table 6-1**. The values of the parameters defined by the half-precision floating-point format are shown in **Table 6-4**.

Table 6-4 Half-Precision Floating-Point Format and Parameter Values

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

Table 6-5 shows the minimum and maximum values that can be represented in the half-precision floating-point format.

Table 6-5 Half-Precision Floating-Point Minimum and Maximum Values

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

6.1.3 Register Set

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

- Single-precision floating-point instruction:

  32 registers (32 bits each) can be specified. These general-purpose registers correspond to r0 to r31.
- Double-precision floating-point instruction:

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

(1) Floating-point system registers

Six system registers can be used by the FPU.

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

For details about the floating-point system registers, see **3.4 FPU Function Registers**.

6.1.4 Floating-Point Instructions

Floating-point instructions are divided into single-precision instructions (single) and double-precision instructions (double), and include the following instructions (mnemonics).

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

6.1.5 Floating-Point Operation Exceptions

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

(1) Types of exceptions

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

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

- When exceptions are enabled

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

- When exceptions are prohibited

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

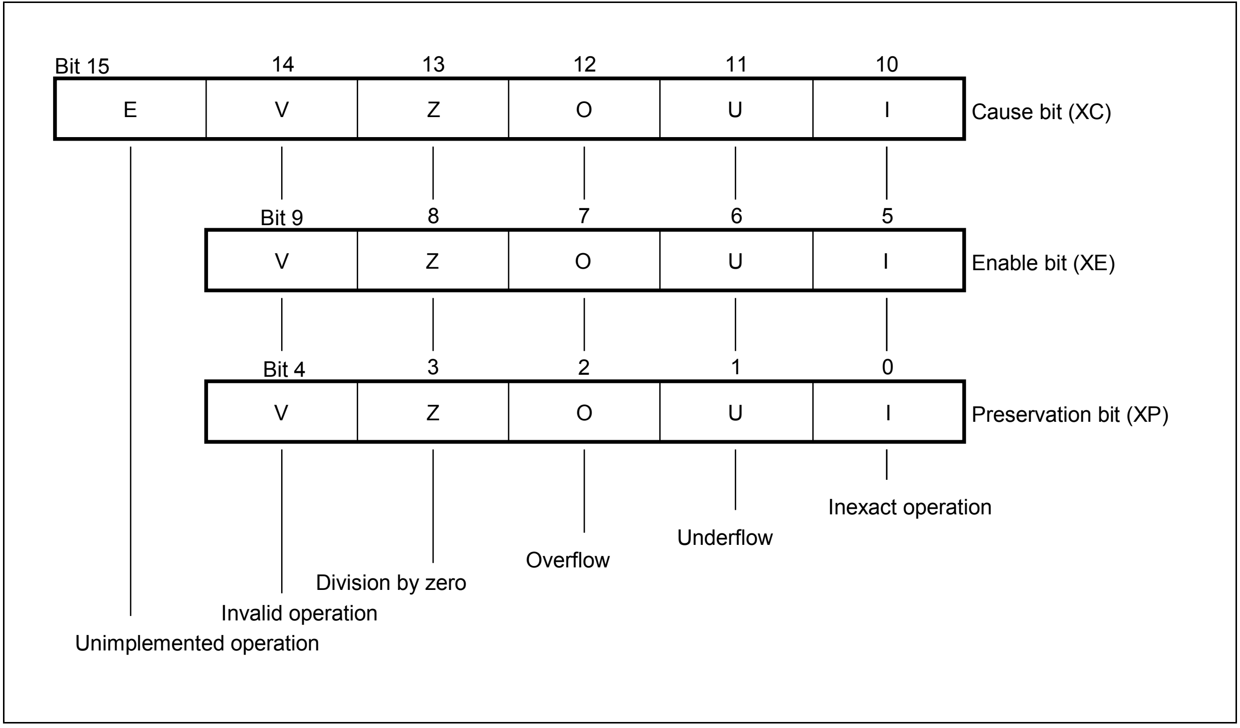
The FPU uses cause bits, enable bits, and preservation bits (status flags) to support the following five types of IEEE754-defined exception causes.

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

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

Figure 6-6 shows the FPSR register bits that are used to support exceptions.

Figure 6-6 Cause, Enable, and Preservation Bits of FPSR Register



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

(2) Exception handling

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

(a) Status flag

A corresponding preservation bit is available for each IEEE754-defined exception. The preservation bit is set when the corresponding exception is prohibited but the exception condition has been detected. The preservation bit is set or reset whenever new values are written to the FPSR register by the LDSR instruction. If an exception is prohibited by an enable bit, predetermined processing is performed by the FPU. This processing provides an initial value as the result, rather than a floating-point operation result. This initial value is determined according to the type of exception. For an overflow exception or underflow exception, the initial value also differs depending on the current rounding mode. **Table 6-6** shows the initial values provided for each of the FPU IEEE754-defined exceptions.

Table 6-6 FPU Initial Values for IEEE754-Defined Exceptions

Area	Description	Rounding Mode	Initial Value
V	Invalid operation	—	Quiet not-a-number (Q-NaN)
Z	Division by zero	—	Correctly signed ∞
O	Overflow	RN	∞ with sign of intermediate result
		RZ	Maximum normalized number with sign of intermediate result
		RP	Negative overflow: Maximum negative normalized number Positive overflow: +∞
		RM	Positive overflow: Maximum positive normalized number Negative overflow: –∞
U	Underflow[Note 1]	RN[Note 2]	0 with sign of intermediate result
		RZ	0 with sign of intermediate result
		RP	Positive underflow: Minimum positive normalized number Negative underflow: 0
		RM	Negative underflow: Minimum negative normalized number Positive underflow: 0
I	Inexact operation	—	Rounded result

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

2. If the rounding mode is RN and the FN bit of the FPSR register is set, flushing will occur in the direction of higher accuracy. For details, see **6.1.11 Flush to Nearest**.

6.1.6 Exception Details

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

(1) Inexact exception (I)

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

- When the precision of the rounded result is dropped
- When the rounded result overflows while overflow exceptions are prohibited
- When the rounded result underflows while underflow exceptions are prohibited
- When the operand that is a subnormal number is flushed, neither an invalid operation exception (V) nor a division by zero exception (Z) is detected, and the other operands are not Q-NaN

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

(a) If exception is enabled

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

(b) If exception is not enabled

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

(2) Invalid operation exception (V)

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

- Arithmetic operation with S-NaN included in operands. The conditional move instruction (CMOV), absolute value (ABS), and arithmetic negation (NEG) are not handled as arithmetic operations, but minimum value (MIN) and maximum value (MAX) are handled as arithmetic operations.
- Multiplication: ±0 × ±∞ or ±∞ × ±0
- Fused-multiply-add: (±0 × ±∞) + c or (±∞ × ±0) + c. But only if c is not Q-NaN.
- Addition/subtraction or multiply-add operation**Note**:

  Addition of infinite values with different signs or subtraction of infinite values with the same sign
- Division: ±0 ÷ ±0 or ±∞ ÷ ±∞
- Square root: When operand is less than 0
- Conversion to integer when source is outside of integer range.
- Comparison: With condition codes 8 to 15, if the operand is unordered (see **Table 7-10 Definitions of Condition Code Bits and Their Logical Inversions**)

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

(a) If exception is enabled

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

(b) If exception is not enabled

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

(3) Division by zero exception (Z)

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

(a) If exception is enabled

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

(b) If exception is not enabled

If no other exception occurs, a correctly signed infinite number (±∞) is stored in the destination register.

(4) Overflow exception (O)

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

(a) If exception is enabled

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

(b) If exception is not enabled

If no other exception occurs, the initial value that is determined by the rounding mode and the sign of the intermediate result is stored in the destination register (see **Table 6-6 FPU Initial Values for IEEE754-defined Exceptions**).

(5) Underflow exception (U)

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

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

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

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

In this CPU, an underflow is detected before rounding.

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

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

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

(a) If exception is enabled

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

(b) If exception is not enabled

If the FS bit of the FPSR register has been set, the initial value determined according to the rounding mode and intermediate result value is stored in the destination register (see **Table 6-6 FPU Initial Values for IEEE754-defined Exceptions**).

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

(6) Unimplemented operation exception (E)

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

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

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

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

Cautions 1. For details about the processing when an unimplemented operation exception (E) occurs, see **6.1.10 Selection of Floating-Point Operation Model**.

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

6.1.7 Precise Exceptions and Imprecise Exceptions

Each floating-point operation exception can be specified as an exception that occurs precisely (precise exception) or imprecisely (imprecise exception).

The default setting is that imprecise exceptions occur. To generate precise exceptions, the exception mode must be changed.

This CPU specifies precise exception mode by setting the PEM bit of the FPSR register.

(1) Precise exceptions

When a precise exception is specified, the CPU does not start execution of any subsequent instructions until the already started floating-point instruction has been completed. Consequently, when an exception occurs, the program can continue after emulation by software.

The program counter for the instruction where a floating-point operation exception has occurred is stored in the EIPC register and FPEPC register. When returning from emulation processing, an EIRET instruction is executed.

Any floating-point operation exception that has occurred during precise exception mode is acknowledged immediately, regardless of the status of the ID bit or NP bit of PSW.

(2) Imprecise exceptions

When an imprecise exception is specified, the CPU is able to start execution of subsequent instructions even before the already started floating-point instruction has been completed. Consequently, when an exception occurs, the subsequent instructions are executed speculatively, so if an exception occurs, emulation becomes difficult but the throughput of instruction execution can be greatly increased.

When a floating-point operation exception occurs for a floating-point instruction executed in imprecise exception mode, the results of subsequent floating-point instructions (except for a TRFSR instruction) are not reflected in the general-purpose register after the exception is acknowledged and until processing of the exception handler routine starts, and no other floating-point operation exceptions occur. This is called an “invalidating instruction”.

The program counter for the instruction where a floating-point operation exception has occurred is stored in the FPEPC register, and the program counter for an instruction that is interrupted when an exception is acknowledged is stored in the EIPC register.

A floating-point operation exception that has occurred in imprecise exception mode is held pending when the ID bit of PSW = 1 or when the NP bit = 1. In such cases, when an LDSR instruction is used to set the NP and ID bits of the PSW register as “0”, the pending exception is acknowledged.

6.1.8 Saving and Returning Status

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

A floating-point operation exception code is 71H for a precise exception and 72H for an imprecise exception.

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

When a floating-point instruction is used in a floating-point operation exception handler routine, the FPSR and FPEPC registers will be overridden if another floating-point operation exception occurs. In such cases, the FPSR and FPEPC registers should be saved at the start of the floating-point operation exception handler processing, and should be returned at the end of the handler processing. When precise exception mode is specified, execute the SYNCI instruction after the FPSR and FPEPC registers are saved. When imprecise exception mode is specified and it is necessary to return to the original program, execute the SYNCE instruction at the end of the handler processing, and then execute the EIRET instruction.

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

6.1.9 Flushing Subnormal Numbers

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

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

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

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

By setting the FS bit of the FPSR register to 1, this CPU can normalize the operand or operation result to a specific value and continue executing arithmetic processing if a subnormal number is input as the operand or obtained as the operation result. At this time, extremely small differences in values might not appear in the operation result. For the operand and operation result, the values to which subnormal numbers are flushed when the FS bit is set (1) are shown in **Tables 6-7** and **6-8** below.

Table 6-7 Rounding Mode and Flush Value of Input Operand

Sign of Subnormal Operand	Rounding Mode and Value to Which Input Operand Is Flushed			
	RN	RZ	RP	RM
+	+0			
–	–0			

Table 6-8 Rounding Mode and Flush Value of Operation Result

Sign of Subnormal Operation Result	Rounding Mode and Value to Which Operation Result Is Flushed			
	RN[Note]	RZ	RP	RM
+	+0	+0	$+2^{Emin}$	+0
–	–0	–0	–0	-2^{Emin}

Note If the rounding mode is RN and the FN bit of the FPSR register is set, flushing will occur in the direction of higher accuracy. For details, see **6.1.11 Flush to Nearest**.

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

Cautions 1. In control systems that require a real-time performance, it is recommended to always set the FS bit to 1.

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

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

4. The IF bit of the FPSR register also accumulates and indicates information about flushing instructions that have caused a floating-point operation exception.

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

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

Caution To obtain an accurate result when using software processing, floating-point operation exceptions must be able to be acknowledged when an unimplemented operation exception occurs. Be sure, therefore, to set the PEM bit of the FPSR register to 1 to enable the correct acknowledgment of floating-point operation exceptions.

(3) Instructions that can handle subnormal numbers

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

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

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

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

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

6.1.10 Selection of Floating-Point Operation Model

This CPU has three recommended floating-point operation models that can be selected according to whether you want to process floating point operations focusing on speed or accuracy.

If you want to focus on processing speed, select the *do not generate exceptions* model, in which processing performance is prioritized by minimizing the occurrence of exceptions during the execution of floating point operations. By using this model, the emulation processing overhead generated by exception handling can be removed, making it ideal for applications that require a real-time performance, but that do not require such a high level of accuracy.

It is also possible to select an *imprecise exception* model, in which high-speed processing is executed as long as no exceptions occur, but which switches to exception handling when an exception does occur. If you anticipate using this model in applications such as those mentioned above that require high-speed processing, debugging can be made easier by designing the software so that exceptions are detected and processed early, thus preserving an internal status close to the status of when the event that caused the exception occurred.

For applications that require a high level of accuracy and that you want to manage by using software, select the *precise exception* model, in which the system shifts to exception handling as soon as a floating-point operation exception is detected. This model uses software-based processing to generate more accurate operation results.

(1) Do not generate exceptions model

If you want to use this model, which prioritizes processing speed and minimizes the occurrence of exceptions, specify the following settings:

- Clear the enable bit of the FPSR register to 0 to suppress the occurrence of floating-point operation exceptions.
- Set the FS bit of the FPSR register to 1 to flush subnormal numbers.
- Use the single-precision floating-point format for processing that does not require a high level of accuracy.

By disabling the generation of floating-point operation exceptions that can be ignored during arithmetic processing, arithmetic processing can continue to be executed using default values. Also, if progressive underflows can be ignored when flushing subnormal numbers, arithmetic processing can continue to be executed using flushed values. The use of single-precision instructions also generally reduces the number of execution clock cycles (latency) required to complete the processing.

Detect exception events that occur during arithmetic processing by explicitly referencing cause flags set by using a separate software program.

(2) Imprecise exception model

If you want to use this model, which prioritizes speed but also allows exceptions to be generated, specify the following settings.

- Set the enable bit of the FPSR register to an appropriate value according to the necessity of exception handling
- Set the FS bit of the FPSR register to 1 to flush subnormal numbers.
- Use the single-precision floating-point format for processing that does not require a high level of accuracy.
- Clear the PEM bit of the FPSR register to 0 to specify the imprecise exception mode.

By disabling the generation of floating-point operation exceptions that can be ignored during arithmetic processing, arithmetic processing can continue to be executed using default values. Also, if progressive underflows can be ignored when flushing subnormal numbers, arithmetic processing can continue to be executed using flushed values. The use of single-precision instructions also generally reduces the number of execution clock cycles (latency) required to complete the processing. The processing throughput in imprecise exception mode is therefore higher than that in precise exception mode.

(3) Precise exception model

If you want to use this model, which allows exceptions to be processed as soon as they occur so that the processing accuracy can be managed by using software, specify the following settings:

- Set the enable bit of the FPSR register to an appropriate value according to the necessity of exception handling
- Clear the FS bit of the FPSR register to 0 to generate an exception if a subnormal number exists.
- Use the single-precision floating-point format or double-precision floating-point format as required
- Set the PEM bit of the FPSR register to 1 to specify the precise exception mode.

Specifying these settings enables exceptions to be acknowledged immediately, at the instruction that caused the exception. Subsequent instructions are not executed and the processor status of before the exception-causing instruction was executed is retained. This enables software-based emulation in cases where extremely accurate arithmetic operations are required. If an IEEE754 exception is triggered by the emulated operation, also emulate that exception.

By searching for instructions by using the FPEPC register, the exception handler can determine the following.

- The instruction being executed
- The destination format

To obtain an accurately rounded result when an overflow exception, underflow exception (except one caused by a conversion instruction), or inexact exception occurs, include program code that searches for the source register and emulates the instruction in the exception handler routine.

To obtain an accurate result when an invalid operation exception or division by zero exception occurs, or an overflow or underflow exception occurs during floating-point conversion, include program code in the exception handler routine that searches for the instruction’s source register and obtains the operand value.

In the IEEE754 standard, it is recommended to prioritize overflow and underflow exceptions over inexact exceptions. The exception priority can be specified by using software. Be sure to set the hardware enable bits of the overflow, underflow, and inexact exceptions.

Note that if an attempt is made to execute an instruction with an invalid data format in the FPU, or if an operand input or operation result is a subnormal number while the FS bit of the FPSR register is cleared (0), an unimplemented operation exception (E) will occur (except for some instructions). In this case, neither the operand nor the contents of the destination register will be changed.

6.1.11 Flush to Nearest

This CPU provides flush-to-nearest mode, a feature for flushing to the nearest number with higher accuracy when a flushing operation results subnormal number. Flush-to-nearest mode is enabled when the rounding mode is RN and the FN bit of the FPSR register is set (1). When this mode is used, the FPU determines the value to which to flush the subnormal number based on the number of the operation result and not just the sign. However, the result is flushed to $\pm 2^{Emin}$, which is different from the value shown in Table 6-9, when the operation result of the subtract operation by SUBF, FMSF, FNMSF instructions and the add operation of a negative value by ADDF, FMAF, FNMAF instructions becomes $\pm 2^{(Emin-2)}$. This feature has no effect in rounding modes other than RN or on the result of flushing an input operand.

Table 6-9 Rounding Mode and Value to Which Operation Result is Flushed

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

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

CHAPTER 7 INSTRUCTION

7.1 Opcodes and Instruction Formats

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

7.1.1 CPU Instructions

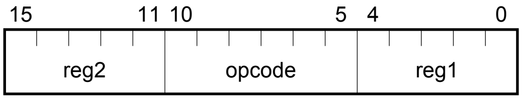
Instructions classified as CPU instructions are allocated in the opcode area other than the area used in the format of the coprocessor instructions shown in **7.1.2 Coprocessor Instructions**.

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

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

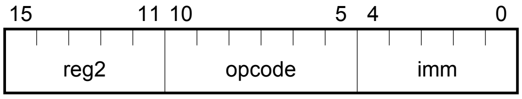
(1) reg-reg instruction (Format I)

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



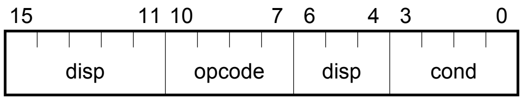
(2) imm-reg instruction (Format II)

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



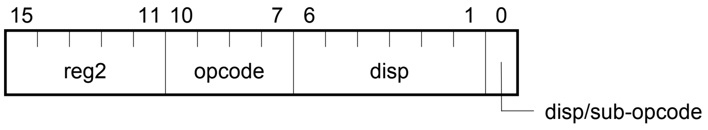
(3) Conditional branch instruction (Format III)

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

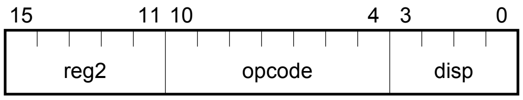


(4) 16-bit load/store instruction (Format IV)

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

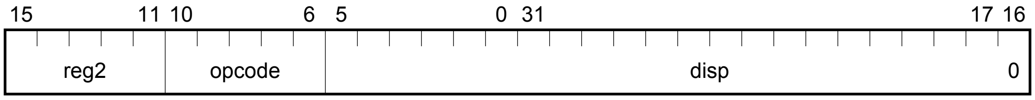


In addition, a 16-bit instruction format consists of a 7-bit opcode field, a general-purpose register specification field, and a 4-bit displacement field.



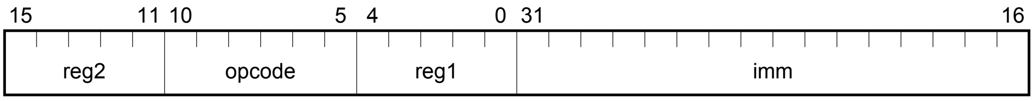
(5) Jump instruction (Format V)

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



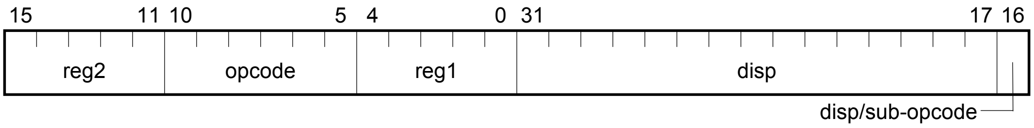
(6) 3-operand instruction (Format VI)

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



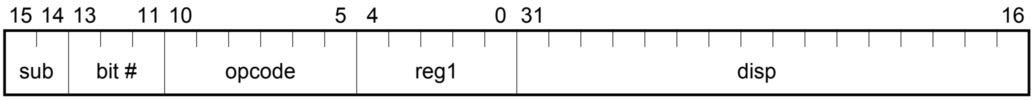
(7) 32-bit load/store instruction (Format VII)

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



(8) Bit manipulation instruction (Format VIII)

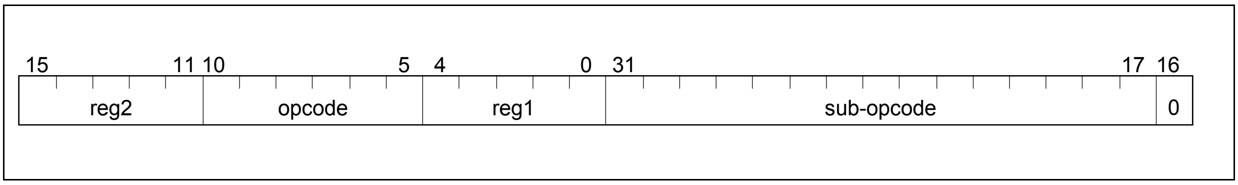
A 32-bit instruction format consists of a 6-bit opcode field, 2-bit sub-opcode field, 3-bit bit specification field, a general-purpose register specification field, and a 16-bit displacement field.



(9) Extended instruction format 1 (Format IX)

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

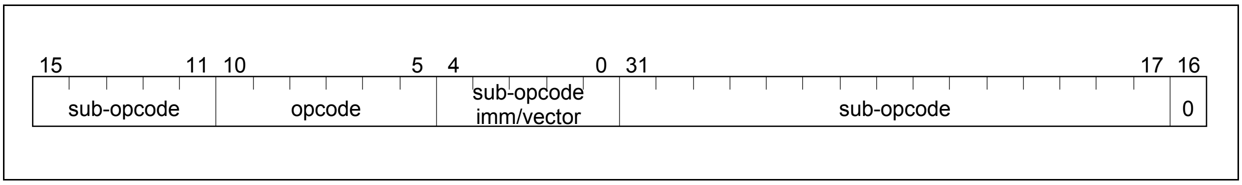
> **Caution** Extended instruction format 1 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in **7.2.2 Basic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



(10) Extended instruction format 2 (Format X)

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

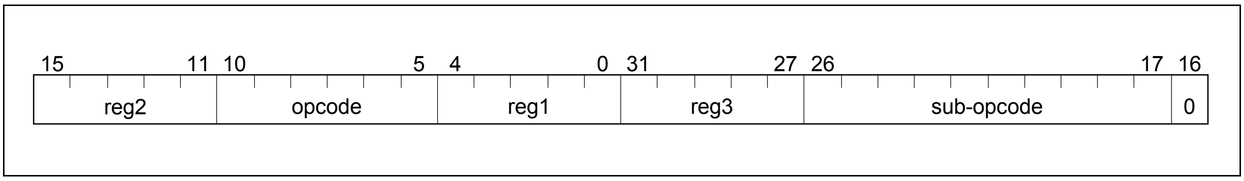
> **Caution** Extended instruction format 2 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in **7.2.2 Basic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



(11) Extended instruction format 3 (Format XI)

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

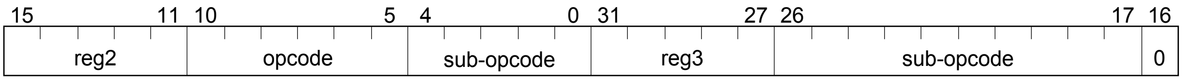
> **Caution** Extended instruction format 3 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in **7.2.2 Basic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



(12) Extended instruction format 4 (Format XII)

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

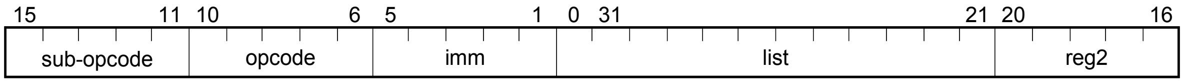
Caution Extended instruction format 4 might use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, see the description of each instruction in **7.2.2 Basic Instruction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



(13) Stack manipulation instruction format (Format XIII)

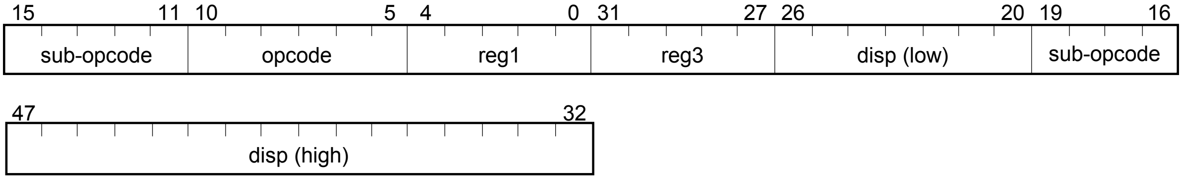
A 32-bit instruction format consists of a 5-bit opcode field, 5-bit immediate field, 12-bit register list field, 5-bit sub-opcode field, and one general-purpose register specification field (or 5-bit sub-opcode field).

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



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

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



7.1.2 Coprocessor Instructions

Instructions in the following format are defined as coprocessor instructions.

15 11	10	9	8	7	6	5	4 0	31 27	26	25 17	16
reg2	1	1	1	1	1	1	opcode or reg1	reg3	1	opcode	0

Coprocessor instructions define the functions of each coprocessor.

(1) Coprocessor unusable exception

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

For details, see **2.4.3 Coprocessor Unusable Exceptions**.

7.1.3 Reserved Instructions

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

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

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

- 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)

7.2 Basic Instructions

7.2.1 Overview of Basic Instructions

(1) Load instructions

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

(a) LD instructions

• LD.B:	Load byte
• LD.BU:	Load byte unsigned
• LD.DW:	Load double word
• LD.H:	Load halfword
• LD.HU:	Load halfword unsigned
• LD.W:	Load word

(b) SLD instructions

• SLD.B:	Short format load byte
• SLD.BU:	Short format load byte unsigned
• SLD.H:	Short format load halfword
• SLD.HU:	Short format load halfword unsigned
• SLD.W:	Short format load word

(2) Store instructions

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

(a) ST instructions

• ST.B:	Store byte
• ST.DW:	Store double word
• ST.H:	Store halfword
• ST.W:	Store word

(b) SST instructions

• SST.B:	Short format store byte
• SST.H:	Short format store halfword
• SST.W:	Short format store word

(3) Multiply instructions

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

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

(4) Multiply-accumulate instructions

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

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

(5) Arithmetic instructions

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

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

(6) Conditional arithmetic instructions

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

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

(7) Saturated operation instructions

Execute saturated addition and subtraction. If the operation result exceeds the maximum positive value (7FFFFFFFH), 7FFFFFFFH returns. If the operation result exceeds the maximum negative value (80000000H), 80000000H returns. The following instructions (mnemonics) are provided.

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

(8) Logical instructions

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

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

(9) Data manipulation instructions

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

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

(10) Bit search instructions

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

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

(11) Divide instructions

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

• DIV:	Divide word
• DIVH:	Divide halfword
• DIVHU:	Divide halfword unsigned
• DIVU:	Divide word unsigned

(12) High-speed divide instructions

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

• DIVQ:	Divide word quickly
• DIVQU:	Divide word unsigned quickly

(13) Branch instructions

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

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

(14) Loop instruction

• LOOP:	Loop

(15) Bit manipulation instructions

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

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

(16) Special instructions

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

• CALLT:	Call with table look up
• CAXI:	Compare and exchange for interlock
• CLL:	Clear load link
• CTRET:	Return from CALLT
• DI:	Disable interrupt
• DISPOSE:	Function dispose
• EI:	Enable interrupt
• EIRET:	Return from trap or interrupt
• FERET:	Return from trap or interrupt
• FETRAP:	Software trap
• HALT:	Halt
• LDSR:	Load system register
• LDL.W:	Load linked word
• NOP:	No operation
• POPSP:	Pop registers from stack
• PREPARE:	Function prepare
• PUSHSP:	Push registers from stack
• RIE:	Reserved instruction exception
• SNOOZE:	Snooze
• STSR:	Store system register
• STC.W:	Store conditional word
• SWITCH:	Jump with table look up
• SYNCE:	Synchronize exceptions
• SYNCI:	Synchronize memory for instruction writers
• SYNCM:	Synchronize memory
• SYNCP:	Synchronize pipeline
• SYSCALL:	System call
• TRAP:	Trap

7.2.2 Basic Instruction Set

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

- Instruction format: Indicates how the instruction is written and its operand(s) (for symbols, see **Table 7-1**).
- Operation: Indicates the function of the instruction (for symbols, see **Table 7-2**).
- Format: Indicates the instruction format (see **7.1 Opcodes and Instruction Formats**).
- Opcode: Indicates the bit field of the instruction opcode (for symbols, see **Table 7-3**).
- Flag: Indicates the change of flags of PSW (program status word) after the instruction execution.
  "0" is to clear (reset), "1" to set, and "--" to remain unchanged.
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.
- Caution: Provides precautionary notes.

Table 7-1 Conventions of Instruction Format

Symbol	Meaning
reg1	General-purpose register (as source register)
reg2	General-purpose register (primarily as destination register with some as source registers)
reg3	General-purpose register (primarily used to store the remainder of a division result and/or the higher 32 bits of a multiplication result)
bit#3	3-bit data to specify bit number
imm×	×-bit immediate data
disp×	×-bit displacement data
regID	System register number
selID	System register group number
vector×	Data to specify vector (× indicates the bit size)
cond	Condition code (see **Table 7-4 Condition Codes**)
cccc	4-bit data to specify condition code (see **Table 7-4 Condition Codes**)
sp	Stack pointer (r3)
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*.

Table 7-2 Conventions of Operation

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*.
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 ≥ 7FFFFFFFH, n = 7FFFFFFFH. If n ≤ 80000000H, n = 80000000H.
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

Table 7-3 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 7-4 Condition Codes**)
bbb	3-bit data for bit number specification
L	1-bit data of code specifying general-purpose register in register list
S	1-bit data of code specifying EIPC/FEPC, EIPSW/FEPSW in register list
P	1-bit data of code specifying PSW in register list

Table 7-4 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

<Arithmetic instruction>

ADD

Add register/immediate

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]

```
    15              0
(1) rrrrr001110RRRRR
```

```
    15              0
(2) 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.

<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.

<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	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	0110	LT	(S xor OV) = 1
0010	Z	Z = 1	1110	GE	(S xor OV) = 0
1010	NZ	Z = 0	0111	LE	((S xor OV) or Z) = 1
0011	NH	(CY or Z) = 1	1111	GT	((S xor OV) or Z) = 0
1011	H	(CY or Z) = 0	(1101)	Setting prohibited	

<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.

<Logical instruction>

ANDI

AND immediate

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	“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 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.

<Branch instruction>

Bcond

Branch on condition code with 9-bit displacement

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 (see **Table 7-5 Bcond Instructions**).

(2)

15 0	31 16
00000111111DCCCC	ddddddddddddddd1

dddddddddddddddd is the higher 16 bits of disp17.

cccc is the condition code of the condition indicated by cond. (For details, see **Table 7-5 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 7-5 Bcond Instructions

Instruction		Condition Code (cccc)	Flag Status	Branch Condition
Signed integer	BGE	1110	(S xor OV) = 0	Greater than or equal to signed
	BGT	1111	((S xor OV) or Z) = 0	Greater than signed
	BLE	0111	((S xor OV) or Z) = 1	Less than or equal to signed
	BLT	0110	(S xor OV) = 1	Less than signed
Unsigned 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.
2. For Bcond disp17 (instruction format (2)), BR (0101) cannot be specified as the condition code.

<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.

<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.

<Data manipulation instruction>

BSW

Byte swap word

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.

<Special instruction>

CALLT

Call with table look up

Subroutine call with table look up

[Instruction format] CALLT imm6

[Operation]

CTPC ← PC + 2 (return PC)

CTPSW(4:0) ← PSW(4:0)

adr ← CTBP + zero-extend (imm6 logically shift left by 1)[Note]

PC ← CTBP + zero-extend (Load-memory (adr, Half-word))

Note 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) Transfers the contents of both return PC and PSW to CTPC and CTPSW.

(2) Adds the CTBP value to the 6-bit immediate data, logically left-shifted by 1, and zero-extended to word length, to generate a 32-bit table entry address.

(3) Loads the halfword entry data of the address generated in step (2) and zero-extend to word length.

(4) Adds the CTBP value to the data generated in step (3) to generate a 32-bit target address.

(5) Jumps to the target address.

Cautions 1. When an exception occurs during CALLT instruction execution, the execution is aborted after the end of the read/write cycle.

2. Memory protection is performed when executing a memory read operation to read the CALLT instruction table. When memory protection is enabled, the data for generating a target address from a table allocated in an area to which access from a user program is prohibited cannot be loaded.

<Special instruction>

Compare and exchange for interlock

CAXI

Comparison and swap

[Instruction format] CAXI [reg1], reg2, reg3

[Operation]

```
adr ← GR[reg1]Note
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 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 occurs in the *result* operation; otherwise, “0”
OV	“1” if overflow occurs in the *resul*t operation; otherwise, “0”
S	“1” if result is negative; otherwise, “0”
Z	“1” if result is 0; otherwise, “0”
SAT	—

[Description] Word data is read from the specified address and compared with the word data in general-purpose register reg2, and the result is indicated by flags in the PSW. Comparison is performed by subtracting the read word data from the word data in general-purpose register reg2. If the comparison result is “0”, word data in general-purpose register reg3 is stored in the generated address, otherwise the read word data is stored in the generated address. Afterward, the read word data is stored in general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.

Cautions 1. This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.

2. The CAXI instruction is included for backward compatibility. If you are using a multi-core system and require an atomic guarantee, use the LDL.W and STC.W instructions.

3. According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<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] The thread link generated by the LDL.W instruction is deleted.

For details about the link operation between the thread and core, see 5.3.2 Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions.

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

<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]

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]

token ← Load-memory (adr, Byte)

Z flag ← Not (extract-bit (token, reg2))

token ← clear-bit (token, reg2)

Store-memory (adr, token, Byte)

Note 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.

(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the bits indicated by the lower three bits of reg2 are cleared (0), and the data is written back to the original address.

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

Caution This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.

<Data manipulation instruction>

Conditional move

CMOV

Conditional move

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

(2) CMOV cccc, imm5, reg2, reg3

[Operation] (1) if conditions are satisfied

then GR [reg3] ← GR [reg1]

else GR [reg3] ← GR [reg2]

(2) if conditions are satisfied

then GR [reg3] ← sign-extended (imm5)

else GR [reg3] ← GR [reg2]

[Format] (1) Format XI

(2) Format XII

[Opcode]

	15 0	31 16
(1)	rrrrr111111RRRRR	wwwww011001cccc0

	15 0	31 16
(2)	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.

<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]

```
     15              0
(1)  rrrrr001111RRRRR

     15              0
(2)  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.

<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.

(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.

<Special instruction>

DI

Disable interrupt

Disable EI level maskable exception

[Instruction format] DI

[Operation] PSW.ID ← 1 (Disables EI level maskable interrupt)

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000101100000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—
ID	1

[Description] Sets “1” to the ID flag of the PSW to disable the acknowledgement of EI level maskable exceptions after the execution of this instruction.

[Supplement] Overwrite of flags in the PSW by this instruction becomes valid as of the next instruction.
If the MCTL.UIC bit has been cleared to 0, this instruction is a supervisor-level instruction.
If the MCTL.UIC bit has been set to 1, this instruction can always be executed.

<Special instruction>

DISPOSE

Function dispose

Stack frame deletion

[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 ← $\text{tmp}^{\text{Notes 1, 2}}$
        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 ← $\text{tmp}^{\text{Notes 1, 2}}$
        GR[reg in list12] ← Load-memory (adr, Word)
        tmp ← tmp + 4
 }
 PC ← GR[reg1]
 sp ← tmp
```

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

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 LLLLLLLLLLL 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”: 08000001H
- When all of the register’s non-corresponding bits are “1”: 081FFFFFH

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description]

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

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

[Supplement]

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

Cautions 1. If an exception occurs while this instruction is being executed, execution of the instruction might be stopped after the read/write cycle and the register value write operation are completed, but sp will retain its original value from before the start of execution. The instruction will be executed again later, after a return from the exception.

2. For instruction format (2) DISPOSE imm5, list12, [reg1], do not specify r0 for reg1.

<Divide instruction>

DIV

Divide word

Division of (signed) word data

[Instruction format] DIV reg1, reg2, reg3

[Operation] GR [reg2] ← 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. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] Overflow occurs when the maximum negative value (80000000H) is divided by −1 with the quotient = 80000000H and when the data is divided by 0 with quotient being undefined.

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

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

Caution 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.

<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. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

(2) Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] (1) The remainder is not stored. Overflow occurs when the maximum negative value (80000000H) is divided by −1 with the quotient = 80000000H and when the data is divided by 0 with quotient being undefined.

When an exception occurs during the DIVH instruction execution, the execution is aborted to process the exception. 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 (80000000H) is divided by −1 with the quotient = 80000000H and when the data is divided by 0 with quotient being undefined.

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

When an exception occurs during the DIVH instruction execution, the execution is 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).

<Divide instruction>

DIVHU

Divide halfword unsigned

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” when the operation result quotient word data is “1”; otherwise, “0”
Z	“1” if the operation result quotient is “0”; otherwise, “0”.
SAT	—

[Description] Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1 and stores the quotient to general-purpose register reg2 with the remainder set to general-purpose register reg3. General-purpose register reg1 is not affected. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] Overflow occurs by division by zero (with the operation result being undefined).
If reg2 and reg3 are the same register, the remainder is stored in that register.
When an exception occurs during the DIVHU instruction execution, the execution is aborted to process the exception. The execution resumes at the original instruction address upon returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

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

<High-speed divide instructions>

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. When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] (1) Overflow occurs when the maximum negative value (80000000H) is divided by −1 (with the quotient = 80000000H) and when the data is divided by 0 with the quotient being undefined.

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

When an exception occurs during execution of this instruction, the execution is aborted. After exception handling is completed, the execution resumes at the original instruction address when returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

(2) The smaller the difference in the number of valid bits between reg1 and reg2, the smaller the number of execution cycles. In most cases, the number of instruction cycles is smaller than that of the ordinary division instruction. If data of 16-bit integer type is divided by another 16-bit integer type data, the difference in the number of valid bits is 15 or less, and the operation is completed within 20 cycles.

Cautions

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

<High-speed divide instructions>

DIVQU

Divide word unsigned quickly

Division of (unsigned) word data (variable steps)

[Instruction format] DIVQU reg1, reg2, reg3

[Operation] GR [reg2] ← 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 operation result quotient is a negative value; otherwise, “0”.
Z	“1” when operation result quotient is a “0”; otherwise, “0”.
SAT	—

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

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

When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] (1) An overflow occurs when there is division by zero (the operation result is undefined).

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

When an exception occurs during execution of this instruction, the execution is aborted. After exception handling is completed, using the return address as this instruction’s start address, the execution resumes when returning from the exception. General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

(2) The smaller the difference in the number of valid bits between reg1 and reg2, the smaller the number of execution cycles. In most cases, the number of instruction cycles is smaller than that of the ordinary division instruction. If data of 16-bit integer type is divided by another 16-bit integer type data, the difference in the number of valid bits is 15 or less, and the operation is completed within 20 cycles.

Cautions

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

<Divide instruction>

DIVU

Divide word unsigned

Division of (unsigned) word data

[Instruction format] DIVU reg1, reg2, reg3

[Operation] GR [reg2] ← 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.
When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Supplement] When an exception occurs during the DIVU instruction execution, the execution is aborted to process the exception.
If reg2 and reg3 are the same register, the remainder is stored in that register.
The execution resumes at the original instruction address upon returning from the exception.
General-purpose register reg1 and general-purpose register reg2 retain their values prior to execution of this instruction.

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

<Special instruction>

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 maskable exceptions starting the next instruction.

[Supplement] If the MCTL.UIC bit has been cleared to 0, this instruction is a supervisor-level instruction.
If the MCTL.UIC bit has been set to 1, this instruction can always be executed.

<Special instruction>

EIRET

Return from trap or interrupt

Return from EL 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.

[Supplement] This instruction is a supervisor-level instruction.

<Special instruction>

FERET

Return from 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-level instruction.

Caution The FERET instruction can also be used as a hazard barrier instruction when the CPU’s operating status (PSW) is changed by a control program such as the OS. Use the FERET instruction to clarify the program blocks on which to effect the hardware function associated with the UM bit in the PSW when these bits are changed to accord with the mounted CPU. The hardware function that operates in accordance with the PSW value updated by the FERET instruction is guaranteed to be effected from the instruction indicated by the return address of the FERET instruction.

<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 2]

Notes 1. See **Table 4-1 Exception Cause List**.

2. See **4.5 Exception Handler Address**.

[Format] Format I

[Opcode]

```
15             0
0vvvv00001000000
```

Where `vvvv` is vector4.

Do not set 0H 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 **Table 4-1**. Execution then branches to the exception handler address and exception handling is started.

Table 7-6 shows the correspondence between vector4 and exception cause codes and exception handler address offset. Exception handler addresses are calculated based on the offset addresses listed in **Table 7-6**. For details, see **4.5 Exception Handler Address**.

Table 7-6 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset

vector4	Exception Cause Code	Offset Address
00H	Not specifiable	
01H	00000031H	30H
02H	00000032H	
...		
0FH	0000003FH	

<Special instruction>

HALT

Halt

Halt

[Instruction format] HALT

[Operation] Places the CPU core in the HALT state.

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000100100000

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Places the CPU core that executed the HALT instruction in the HALT state.

Occurrence of the HALT state release request will return the system to normal execution status.

If an exception is acknowledged while the system is in HALT state, the return PC of that exception is the PC of the instruction that follows the HALT instruction.

The HALT state is released under the following condition.

- A terminating exception occurs

Even if the conditions for acknowledging the above exceptions are not satisfied (due to the ID or NP value), as long as a HALT mode release request exists, HALT state is released (for example, even if PSW.ID = 1, HALT state is released when INT0 occurs).

Note, however, that the HALT mode will not be released if terminating exceptions are masked by the following mask settings, which are defined individually for each function:

- Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller[Note].
- 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 This does not include masking specified by the ISPR and PMR registers.

[Supplement] This instruction is a supervisor-level instruction.

<Data manipulation instructions>

HSH

Halfword swap halfword

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.

<Data manipulation instruction>

HSW

Halfword swap word

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.

<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.

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

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

<Branch instruction>

JMP

Jump register

Unconditional branch (register relative)

[Instruction format] (1) JMP [reg1]

(2) JMP disp32 [reg1]

[Operation] (1) PC ← GR [reg1]

(2) PC ← GR [reg1] + disp32

[Format] (1) Format I

(2) Format VI

[Opcode]

(1)

15 0
00000000011RRRRR

(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.

<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.

<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

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

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

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

GR [reg3] ← sign-extend (Load-memory (adr, Byte))

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

[Format] (1) Format VII

(2) Format XIV

[Opcode]

	15 0	31 16
(1)	rrrrr111000RRRRR	dddddddddddddddd

	15 0	31 16	47 32
(2)	00000111100RRRRR	wwwwwddddddd0101	DDDDDDDDDDDDDDDD

Where RRRRR = reg1, wwwww = reg3.

ddddddd 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] (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.

<Load instruction>

LD.BU

Load byte unsigned

Load of (unsigned) byte data

[Instruction format] (1) LD.BU disp16 [reg1] , reg2

(2) LD.BU disp23 [reg1] , reg3

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

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

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

GR [reg3] ← zero-extend (Load-memory (adr, Byte))

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

[Format] (1) Format VII

(2) Format XIV

[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.)

(2)

15 0	31 16	47 32
00000111101RRRRR	wwwwwddddddd0101	DDDDDDDDDDDDDDDD

Where RRRRR = reg1, wwwww = reg3.

ddddddd is the lower 7 bits of disp23.

DDDDDDDDDDDDDDDD is the higher 16 bits of disp23.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

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

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

Caution Do not specify r0 for reg2.

<Load instruction>

LD.DW

Load Double Word

Load of doubleword data

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

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

data ← Load-memory (adr, Double-word)

GR [reg3 + 1] || GR [reg3] ← data

Note 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 `RRRRRR` = 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. Doubleword 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.

Cautions 1. According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

2. However, a misaligned access exception will not occur if the result of address calculation has a word boundary.

<Load instruction>

LD.H

Load halfword

Load of (unsigned) halfword data

[Instruction format] (1) LD.H disp16 [reg1] , reg2

(2) LD.H disp23 [reg1] , reg3

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

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

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

GR [reg3] ← sign-extend (Load-memory (adr, Halfword))

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

[Format] (1) Format VII

(2) Format XIV

[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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Load instruction>

LD.HU	Load halfword unsigned Load of (signed) halfword data

[Instruction format] (1) LD.HU disp16 [reg1] , reg2

(2) LD.HU disp23 [reg1] , reg3

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

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

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

GR [reg3] ← zero-extend (Load-memory (adr, Halfword))

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

[Format] (1) Format VII

(2) Format XIV

[Opcode]

	15 0	31 16
(1)	`rrrrr111111RRRRR`	`ddddddddddddddd1`

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

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

	15 0	31 16	47 32
(2)	`00000111101RRRRR`	`wwwwwdddddd00111`	`DDDDDDDDDDDDDDDD`

Where `RRRRR` = reg1, `wwwww` = reg3.

`dddddd` is the lower side bits 6 to1 of disp23.

`DDDDDDDDDDDDDDDD` is the higher 16 bits of disp23.

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

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

Cautions 1. Do not specify r0 for reg2.

2. According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Load instruction>

LD.W

Load word

Load of word data

[Instruction format] (1) LD.W disp16 [reg1] , reg2

(2) LD.W disp23 [reg1] , reg3

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

GR [reg2] ← Load-memory (adr, Word)

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

GR [reg3] ← Load-memory (adr, Word)

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

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15 0	31 16
rrrrr111001RRRRR	ddddddddddddddd1

Where ddddddddddddddd is the higher 15 bits of disp16.

(2)

15 0	31 16	47 32
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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Special instruction>

LDL.W

Load Linked

Load to start atomic word data manipulation

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

[Operation]

adr ← GR[reg1]$^{\text{Note 1}}$

GR[reg3] ← Load-memory (adr, Word)

LLbit ← 1$^{\text{Note 2}}$

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

2. The result of an interrupt or exception, or the execution of a CLL, EIRET, or FERET instruction is LLbit ← 0.

[Format] Format VII

[Opcode]

15 0	31 16
00000111111RRRRR	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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Special instruction>

LDSR

Load to system register

Load to system register

[Instruction format] LDSR reg2, regID, selID

LDSR reg2, regID

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

Note An exception might occur depending on the access permission. For details, see **2.5.3 Register Updating**.

[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 group number (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 **2.5.3 Register Updating**.

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 group number is a unique number used to identify each system register. How to access undefined registers is described in **2.5.4 Accessing Undefined Registers**, but accessing undefined registers is not recommended.

<Loop instruction>

LOOP

Loop

[Instruction format] LOOP reg1,disp16

[Operation]

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

Note −1 (0xFFFFFFFF) is added. The carry flag is updated in the same way as when the ADD instruction is executed.

[Format] Format VII

[Opcode]

15 0	31 16
00000110111RRRRR	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 (0xFFFFFFFF) is added to general-purpose register reg1. The carry flag is updated in the same way as when the ADD instruction, not the SUB instruction, is executed.

[Supplement] “0” is implicitly used for bit 0 of the 16-bit displacement. Note that, because the current PC value used for calculation is the address of the first byte of this instruction, if the displacement value is 0, the branch destination is this instruction.

Caution Do not specify r0 for reg1.

<Multiply-accumulate instruction>

MAC

Multiply and add word

Multiply-accumulate for (signed) word data

[Instruction format] MAC reg1, reg2, reg3, reg4

[Operation] GR [reg4+1] || 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.

<Multiply-accumulate instruction>

MACU

Multiply and add word unsigned

Multiply-accumulate for (unsigned) word data

[Instruction format] MACU reg1, reg2, reg3, reg4

[Operation] GR [reg4+1] || 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 signed integers.

This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.

Caution General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered register (r0, r2, r4, …, r30). The result is undefined if an odd-numbered register (r1, r3, …, r31) is specified.

<Arithmetic instruction>

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).

<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.

<Arithmetic instruction>

MOVHI

Move high halfword

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.

<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]

	15 0	31 16
(1)	rrrrr111111RRRRR	wwwww01000100000
	15 0	31 16
(2)	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.

<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]

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

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

```
     15             0
(2) |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.

<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.

<Multiply instruction>

MULU

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

Multiplication of (unsigned) word data

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

(2) MULU imm9, reg2, reg3

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

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

[Format] (1) Format XI

(2) Format XII

[Opcode]

	15 0	31 16
(1)	rrrrr111111RRRRR	wwwww01000100010

	15 0	31 16
(2)	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.

<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.

<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.

<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]
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]
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← not-bit (token, reg2)
Store-memory (adr, token, Byte)

Note 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 This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.

<Logical instruction>

OR

OR

OR

[Instruction format] OR reg1, 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.

<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.

<Special instruction>

POPSP

Pop registers from Stack

POP from the stack

[Instruction format] POPSP rh-rt

[Operation]

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

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

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.)

Caution If a register that includes sp(r3) is specified as the restore register (rh = 3 to 31), the value read from the memory is not stored in sp(r3). This allows the POPSP instruction to be correctly re-executed after execution has been halted.

<Special instruction>

PREPARE

Function prepare

Create stack frame

[Instruction format] (1) PREPARE list12, imm5

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

Note 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[Notes 1, 2]
 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[Notes 1, 2]
 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
```

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

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 `LLLLLLLLLLL` 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": 08000001H
- When all of the register's non-corresponding bits are "1": 081FFFFFH

[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.

Caution If an exception occurs while this instruction is being executed, execution of the instruction might be stopped after the write cycle and the register value write operation are completed, but sp will retain its original value from before the start of execution. The instruction will be executed again later, after a return from the exception.

<Special instruction>

PUSHSP

Push registers to Stack

[Instruction format] PUSHSP rh-rt

[Operation]

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

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

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.)

<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 (00000060H)

PSW.UM ← 0

PSW.NP ← 1

PSW.EP ← 1

PSW.ID ← 1

PC ← exception handler address (offset address 60H)

[Format] (1) Format I

(2) Format X

[Opcode]

```
       15                0
(1)   |0000000001000000|

       15               0 31              16
(2)   |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 **Table 4-1**. Execution then branches to the exception handler address and exception handling is started.

Exception handler addresses are calculated based on the offset address 60H. For details, see **4.5 Exception Handler Address**.

<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

	15 0	31 16
(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.

<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]

(1)

15 0	31 16
rrrrr111111RRRRR	0000000010100000

(2)

15 0
rrrrr010101iiiii

(3)

15 0	31 16
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.

<Data manipulation instruction>

SASF

Shift and set flag condition

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 00000001H

else GR [reg2] ← (GR [reg2] Logically shift left by 1) OR 00000000H

[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	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 SETF instruction.

<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]

```
     15             0
(1)  |rrrrr000110RRRRR|
```

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

```
     15             0
(2)  |rrrrr010001iiiii|
```

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

```
     15             0 31            16
(3)  |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 7FFFFFFFH, 7FFFFFFFH is stored in reg2, and when it exceeds the maximum negative value 80000000H, 80000000H 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 7FFFFFFFH, 7FFFFFFFH is stored in reg2, and when it exceeds the maximum negative value 80000000H, 80000000H 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 7FFFFFFFH, 7FFFFFFFH is stored in reg3, and when it exceeds the maximum negative value 80000000H, 80000000H is stored in reg3; then the SAT flag is set (1). General-purpose registers reg1 and reg2 are not affected.

[Supplement] The SAT flag is a cumulative flag. The saturate result sets the flag to “1” and will not be cleared to “0” even if the result of the subsequent operation is not saturated. The saturated operation instruction is executed normally, even with the SAT flag set to “1”.

Cautions 1. Use LDSR instruction and load data to the PSW to clear the SAT flag to “0”.

2. Do not specify r0 as reg2 in instruction format (1) SATADD reg1, reg2 and in instruction format (2) SATADD imm5, reg2.

<Saturated operation instruction>

SATSUB

Saturated subtract

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]

```
      15              0
(1)  |rrrrr000101RRRRR|
```

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

```
      15              0 31              16
(2)  |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 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H 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 7FFFFFFFH, 7FFFFFFFH is stored in reg3, and when it exceeds the maximum negative value 80000000H, 80000000H 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.

<Saturated operation instruction>

SATSUBI

Saturated subtract immediate

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 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H 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.

<Saturated operation instruction>

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 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H 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.

<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	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	0110	LT	(S xor OV) = 1
0010	Z	Z = 1	1110	GE	(S xor OV) = 0
1010	NZ	Z = 0	0111	LE	((S xor OV) or Z) = 1
0011	NH	(CY or Z) = 1	1111	GT	((S xor OV) or Z) = 0
1011	H	(CY or Z) = 0	(1101)	1011	Setting prohibited

<Bit search instructions>

SCH0L

Search zero from left

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, 01H 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).

<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, 01H 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).

<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 (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 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, 01H 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).

<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
rrrrr111111100000	wwwww01101100010

[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 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, 01H 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).

<Bit manipulation instruction>

SET1

Set bit

Bit setting

[Instruction format] (1) SET1 bit#3, disp16 [reg1]

(2) SET1 reg2, [reg1]

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

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

token ← Load-memory (adr, Byte)

Z flag ← Not (extract-bit (token, reg2))

token ← set-bit (token, reg2)

Store-memory (adr, token, Byte)

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

[Format] (1) Format VIII

(2) Format IX

[Opcode]

	15 0	31 16
(1)	00bbb111110RRRRR	dddddddddddddddd

	15 0	31 16
(2)	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 the16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, the bits indicated by the 3-bit bit number are set (1) and the data is written back to the original address.

If the specified bit of the read byte data is “0”, the Z flag is set to “1”, and if the specified bit is “1”, the Z flag is cleared to “0”.

(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the lower 3 bits indicated of general-purpose register reg2 are set (1) and the data is written back to the original address.

If the specified bit of the read byte data is “0”, the Z flag is set to “1”, and if the specified bit is “1”, the Z flag is cleared to “0”.

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

Caution This instruction provides an atomic guarantee aimed at exclusive control, and during the period between read and write operations, the target address is not affected by access due to any other cause.

<Data manipulation instruction>

SETF

Set flag condition

Flag condition setting

[Instruction format] SETF cccc, reg2

[Operation]

```
if conditions are satisfied
  then GR [reg2] ← 00000001H
  else GR [reg2] ← 00000000H
```

[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 (a_1) then, if (a_2) then*. The object code executes "conditional branch" by checking the result of evaluation equivalent to a_n. Because a pipeline operation requires more time to execute "condition judgment" + "branch" than to execute an ordinary operation, the result of evaluating each condition clause *if (a_n)* is stored in register Ra. By performing a logical operation to Ra_n after all the condition clauses have been evaluated, the pipeline delay can be prevented.

(2) Double-length operation

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

<Data manipulation instruction>

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]

	15 0	31 16
(1)	rrrrr111111RRRRR	0000000011000000

	15 0
(2)	rrrrr010110iiiii

	15 0	31 16
(3)	rrrrr111111RRRRR	wwwww00011000010

[Flags]

CY	“1” if the last bit shifted out is “1”; otherwise, “0” including non-shift.
OV	0
S	“1” if the operation result is negative; otherwise, “0”.
Z	“1” if the operation result is “0”; otherwise, “0”.
SAT	—

[Description]

(1) Logically left-shifts the word data of general-purpose register reg2 by ‘n’ (0 to +31), 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.

<Data manipulation instruction>

SHR

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

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.

<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

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

Note 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.

<Load instruction>

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

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

[Operation] adr ← ep + zero-extend (disp4)[Note]

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

Note 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.

<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]

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

Note 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 According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<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]

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

Note 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. According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<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)$^{\text{Note}}$

GR [reg2] ← Load-memory (adr, Word)

Note 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 According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<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 halts operation of the CPU core for the period defined by the hardware specifications or when the CPU enters a specific state.
When the specified period has elapsed or the CPU exits the specified state, CPU operation automatically resumes and instruction execution begins from the next instruction.
The SNOOZE state is released under the following conditions.

- The predefined period of time passes
- A terminating exception occurs

Even if the conditions for acknowledging the above exceptions are not satisfied (due to the ID or NP value), as long as a SNOOZE mode release request exists, the SNOOZE state is released (for example, even if PSW.ID = 1, the SNOOZE state is released when INT0 occurs).
Note, however, that the SNOOZE mode will not be released if terminating exceptions are masked by the following mask settings, which are defined individually for each function.

- Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller[Note].
- 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 This does not include masking specified by the ISPR and PMR registers.

[Supplement] This instruction is used to prevent the CPU performance from dropping in a multi-core system due to bus band occupancy during a spinlock.

Caution 1. The period of the pause triggered by the SNOOZE instruction is defined according to the hardware specifications of the CPU core. For details, see the hardware manual of the product used.

<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]

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

Note 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.

<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]

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

Note 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 According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<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

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

Note 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 According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Store instruction>

ST.B

Store byte

Storage of byte data

[Instruction format] (1) ST.B reg2, disp16 [reg1]

(2) ST.B reg3, disp23 [reg1]

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

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

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

Store-memory (adr, GR [reg3], Byte)

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

[Format] (1) Format VII

(2) Format XIV

[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.

[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.

<Store instruction>

ST.DW

Store Double Word

Storage of doubleword data

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

[Operation]

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

data ← GR[reg3+1] || GR[reg3]

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

Note 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 RRRRRR = 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. Doubleword 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.

Cautions 1. According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

2. However, a misaligned access exception will not occur if the result of address calculation has a word boundary.

<Store instruction>

ST.H

Store halfword

Storage of halfword data

[Instruction format] (1) ST.H reg2, disp16 [reg1]

(2) ST.H reg3, disp23 [reg1]

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

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

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

Store-memory (adr, GR [reg3], Halfword)

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

[Format] (1) Format VII

(2) Format XIV

[Opcode]

	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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Store instruction>

ST.W

Store word

Storage of word data

[Instruction format] (1) ST.W reg2, disp16 [reg1]

(2) ST.W reg3, disp23 [reg1]

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

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

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

Store-memory (adr, GR [reg3], Word)

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

[Format] (1) Format VII

(2) Format XIV

[Opcode]

	15 0	31 16
(1)	rrrrr111011RRRRR	ddddddddddddddd1

Where ddddddddddddddd is the higher 15 bits of disp16.

	15 0	31 16	47 32
(2)	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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.
For details, see the hardware manual of the product used.

<Special instruction>

STC.W

Store Conditional

Conditional storage when atomic word data manipulation is complete

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

[Operation]

```
adr ← GR[reg1][Note 1]
data ← GR[reg3]
token ← LLbit[Note 2]

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

LLbit ← 0[Note 2]
```

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

2. For details about the link operation, see **5.3.2 Performing Mutual Exclusion by Using the LDL.W and STC.W Instructions**.

[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.

[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.

Caution According to the CPU core hardware specifications, a misaligned access exception (MAE) might occur as a result of address calculation.

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

<Special 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

Note An exception might occur depending on the access permission. For details, see **2.5.3 Register Updating**.

[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 group number (regID, selID) in general-purpose register reg2. The system register is not affected. If selID is omitted, it is assumed that selID is 0.

[Supplement] A PIE or UCPOP exception might occur as a result of executing this instruction, depending on the combination of CPU operating mode and system register to be accessed. For details, see **2.5.3 Register Updating**.

Caution The system register number or group number is a unique number used to identify each system register. How to access undefined registers is described in **2.5.4 Accessing Undefined Registers**, but accessing undefined registers is not recommended**.**

<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.

<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]

[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.

<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]

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

Note 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.

<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.

<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.

<Special instruction>

SYNCE	Synchronize exceptions Exception synchronization instruction

[Instruction format] SYNCE

[Operation] Synchronizes exceptions.

[Format] Format I

[Opcode]

```
15              0
0000000000011101
```

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Synchronizes all preceding imprecise exceptions (FPI exceptions) of this instruction. “Imprecise exception synchronization” means that all imprecise exceptions, that are generated by the preceding instructions, are notified to the CPU and are kept waiting until their priority is judged. If a condition of acknowledging exceptions is satisfied when this instruction is executed, all imprecise exceptions (FPI exceptions), that are generated by the preceding instructions, are always acknowledged by executing the SYNCE instruction.

This instruction can be used to guarantee completion of exception handling by the preceding task before a task is changed or terminated in a multi-processing environment.

[Supplement] For details about the synchronization function, see **5.4 Synchronization Function**.

<Special instruction>

SYNCI

Synchronize instruction pipeline

Instruction pipeline synchronization instruction

[Instruction format] SYNCI

[Operation] Synchronizes instruction fetches.

[Format] Format I

[Opcode]

```
15              0
0000000000011100
```

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Discards unexecuted instructions in the pipeline, and re-fetches the subsequent instructions. The SYNCI instruction waits for the completion of execution of the cache instruction and the instruction to update the cache operation function registers[Note]. The SYNCI instruction does not wait for the result of the preceding load and store instructions.

Note When completion of instruction cache clearance is confirmed, check the read value of the ICCTRL.ICHCLR bit.

[Supplement] For details about the synchronization function, see **5.4 Synchronization Function**.

If the CPU includes an instruction cache, the instruction cache must be disabled to realize self-programming code that alters instructions on the memory.

<Special instruction>

SYNCM

Synchronize memory

Memory synchronize instruction

[Instruction format] SYNCM

[Operation] Synchronizes memory accesses.

[Format] Format I

[Opcode]

```
15              0
0000000000011110
```

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Waits for the completion of execution of all preceding instructions and all preceding memory accesses (load and store). By executing the SYNCM instruction, the result of the preceding memory accesses can be referenced by any master device within the system.

[Supplement] For details about the synchronization function, see **5.4 Synchronization Function**. The completion of a store instruction may not be guaranteed by the SYNCM instruction depending on the destination of the store instruction. For details, see the hardware manual of the product used.

<Special instruction>

SYNCP

Synchronize pipeline

Pipeline synchronize instruction

[Instruction format] SYNCP

[Operation] Synchronizes pipeline.

[Format] Format I

[Opcode]

```
15              0
0000000000011111
```

[Flags]

CY	—
OV	—
S	—
Z	—
SAT	—

[Description] Waits for the completion of execution of preceding instructions to reflect the result of the preceding instructions to subsequent instructions. The SYNCP instruction waits for the completion of load instruction (until the loaded data is stored in a register), but does not wait for the completion of store instruction (until the destination memory or register is updated).

[Supplement] For details about the synchronization function, see **5.4 Synchronization Function**.

<Special instruction>

System call

SYSCALL

System call exception

[Instruction format] SYSCALL vector8

[Operation]

```
EIPC ← PC + 4 (return PC)
EIPSW ← PSW
EIIC ← exception cause code[Note 1]
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 2]
    else   adr ← SCBP[Note 2]
PC ← SCBP + Load-memory (adr, Word)
```

Notes

1. See Table 4-1 Exception Cause List.
2. 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]

<1> Saves the contents of the return PC (address of the instruction next to the SYSCALL instruction) and PSW to EIPC and EIPSW.

<2> Stores the exception cause code corresponding to vector8 in the EIIC register.

The exception cause code is the value of vector8 plus 8000H.

<3> Updates the PSW according to the exception causes listed in **Table 4-1**.

<4> Generates a 32-bit table entry address by adding the value of the SCBP register and vector8 that is logically shifted 2 bits to the left and zero-extended to a word length.

If vector8 is greater than the value specified by the SIZE bit of system register SCCFG; however, vector8 that is used for the generation of a 32-bit table entry address is handled as 0.

<5> Loads the word of the address generated in <4>.

<6> Generates a 32-bit target address by adding the value of the SCBP register to the data in <5>.

<7> Branches to the target address generated in <6>.

Caution In the SYSCALL instruction memory read operation executed in order to read the table, memory protection is performed with the supervisor privilege.

<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 2]

Notes

1. See Table 4-1 Exception Cause List.
2. See 4.5 Exception Handler Address.

[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 **Table 4-1**. Execution then branches to the exception handler address and exception handling is started.

The following table shows the correspondence between vector5 and exception cause codes and exception handler address offset. Exception handler addresses are calculated based on the offset addresses listed in the following table. For details, see **4.5 Exception Handler Address**.

vector5	Exception Cause Code	Offset Address
00H	00000040H	40H
01H	00000041H	
...		
0FH	0000004FH	
10H	00000050H	50H
11H	00000051H	
...		
1FH	0000005FH	

<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.

<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]

token ← Load-memory (adr, Byte)

Z flag ← Not (extract-bit (token, bit#3))

(2) adr ← GR [reg1][Note]

token ← Load-memory (adr, Byte)

Z flag ← Not (extract-bit (token, reg2))

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

[Format] (1) Format VIII

(2) Format IX

[Opcode]

	15 0	31 16
(1)	11bbb111110RRRRR	dddddddddddddddd

	15 0	31 16
(2)	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 bit number at the byte data location referenced by the generated address. If the specified bit is “0”, “1” is set to the Z flag of PSW and if the bit is “1”, the Z flag is cleared to “0”. The byte data, including the specified bit, is not affected.

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

<Logical instruction>

XOR

Exclusive OR

Exclusive OR

[Instruction format] XOR 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.

<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.

<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.

<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.	

7.3 Cache Instructions

7.3.1 Overview of Cache Instructions

This CPU provides the cache instructions to enable efficient manipulation of the cache by the CPU.
The following cache instructions (mnemonics) are available。

- CACHE: Cache
- PREF: Prefetch

7.3.2 Cache Instruction Set

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

- Instruction format: Indicates how the instruction is written and its operand(s).
- Operation: Indicates the function of the instruction.
- Format: Indicates the instruction format.
- Opcode: Indicates the bit field of the instruction opcode.
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.

<Cache instruction>

CACHE

Cache

Cache operation

[Instruction format] CACHE cacheop, [reg1]

[Operation] Manipulates the cache specified by cacheop.

[Format] Format X

[Opcode]

15 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 **5.2.5 Cache Index Specification Method**.

[Supplement] Each cache operation has its own instruction execution privilege. For details about the correspondence between cache operations and instruction execution privileges, see **5.2.6 Execution Privilege of the CACHE/PREF Instruction**.

When manipulating the cache by specifying the address, it might become the target of memory protection by the MPU. For details about the relationship between cache manipulation and memory protection, see **5.2.7 Memory Protection for Address Hit-Type Operations**.

Table 7-7 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. If the specified address does not hit an address in the instruction cache, no processing is performed.
0100000	Instruction	CIBII	Index	(Cache Indexed Block Invalidate, Instruction cache) Disables the instruction cache line of the specified index. This instruction can be used in cases such as when the entire memory cache is initialized by software.
1000000	Instruction	CFALI	Address	(Cache Fetch And Lock, Instruction cache) Loads the data from the specified address and stores it in the instruction cache. At this time, the corresponding cache line is locked. If the data at the specified address is already stored in the instruction cache, this instruction only locks the cache line. If the data at the specified address is already stored in the instruction cache and the corresponding cache line is locked, no processing is performed.
1100000	Instruction	CISTI	Index	(Cache Indexed Store, Instruction cache) Writes (stores) data from a system register to the instruction cache line of the specified index. The specifications of the data to be written and the system register depend on the specifications of the CPU core. For details, see the hardware manual of the product used.
1100001	Instruction	CILDI	Index	(Cache Indexed Load, Instruction cache) Reads (loads) data from the instruction cache line of the specified index to a system register. The specifications of the data to be read and the system register depend on the specifications of the CPU core. For details, see the hardware manual of the product used.
1111110	—	CLL	—	(Clear Load Link-bit) Functions as the CLL instruction.

<Cache instruction>

PREF

Prefetch

Prefetch

[Instruction format] PREF prefop, [reg1]

[Operation] Executes the prefetch operation specified by prefop.

[Format] Format X

[Opcode]

15 0	31 16
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] The prefetch instruction does not generate an execution privilege exception in any CPU mode. If the CPU being used does not contain a cache, this instruction will not generate a reserved instruction exception and no processing is performed, in the same way as a NOP instruction.

A mismatch exception or access privilege exception will not be generated by the MMU memory protection feature during a prefetch operation. If a mismatch or access privilege violation occurs, the prefetch operation is implicitly ignored and no processing is performed, in the same way as 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 7-8 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.

Caution 1. The size of the data prefetched in one prefetch operation depends on the specifications of the CPU core.

7.4 Floating-Point Instructions

7.4.1 Instruction Formats

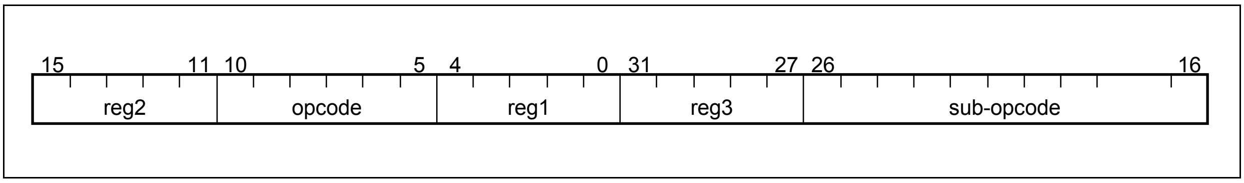
All floating-point instructions are in 32-bit format.

When an instruction is actually saved to memory, it is placed as shown below.

- Lower part of instruction format (including bit 0) → 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.



7.4.2 Overview of Floating-Point Instructions

Floating-point instructions are divided into single-precision instructions (single) and double-precision instructions (double), and include the following instructions (mnemonics).

(1) Basic operation instructions

- ABSF.D: Floating-point absolute value (Double)
- ABSF.S: Floating-point absolute value (Single)
- ADDF.D: Floating-point add (Double)
- ADDF.S: Floating-point add (Single)
- DIVF.D: Floating-point divide (Double)
- DIVF.S: Floating-point divide (Single)
- MAXF.D: Floating-point maximum (Double)
- MAXF.S: Floating-point maximum (Single)
- MINF.D: Floating-point minimum (Double)
- 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 long (Double)
- CVTF.LD: Floating-point convert long to double (Double)
- CVTF.LS: Floating-point convert long to single (Single)
- CVTF.SD: Floating-point convert single to double (Double)
- CVTF.SL: Floating-point convert single to long (Single)
- CVTF.SUL: Floating-point convert single to unsigned-long (Single)
- CVTF.SUW: Floating-point convert single to unsigned-word (Single)
- CVTF.SW: Floating-point convert single to long (Single)
- CVTF.ULD: Floating-point convert unsigned-long to double (Double)
- CVTF.ULS: Floating-point convert unsigned-long to single (Single)
- CVTF.UWD: Floating-point convert unsigned-word to double (Double)
- CVTF.UWS: Floating-point convert unsigned-word to single (Single)
- CVTF.WD: Floating-point convert word to double (Double)
- CVTF.WS: Floating-point convert word to single (Single)
- FLOORF.DL: Floating-point convert double to long, round toward negative (Double)
- FLOORF.DW: Floating-point convert double to long, round toward negative (Double)
- FLOORF.DUL: Floating-point convert double to unsigned-long, round toward negative (Double)
- FLOORF.DUW: Floating-point convert double to unsigned-word, round toward negative (Double)
- FLOORF.SL: Floating-point convert single to long, round toward negative (Single)
- FLOORF.SW: Floating-point convert single to long, round toward negative (Single)
- FLOORF.SUL: Floating-point convert single to unsigned-long, round toward negative (Single)
- FLOORF.SUW: Floating-point convert single to unsigned-word, round toward negative (Single)
- TRNCF.DL: Floating-point convert double to long, round toward zero (Double)
- TRNCF.DUL: Floating-point convert double to unsigned-long, round toward zero (Double)
- TRNCF.DUW: Floating-point convert double to unsigned-word, round toward zero (Double)
- TRNCF.DW: Floating-point convert double to long, round toward zero (Double)
- TRNCF.SL: Floating-point convert single to long, round toward zero (Single)

- TRNCF.SUL: Floating-point convert single to unsigned-long, round toward zero (Single)
- TRNCF.SUW: Floating-point convert single to unsigned-word, round toward zero (Single)
- TRNCF.SW: Floating-point convert single to long, round toward zero (Single)
- CVTF.HS: Floating-point convert half to single (Single)
- CVTF.SH: Floating-point convert single to half (Single)

(4) Comparison instructions

- CMPF.S: Compare floating-point values (Single)
- CMPF.D: Compare floating-point values (Double)

(5) Conditional move instructions

- CMOVF.S: Floating-point conditional move (Single)
- CMOVF.D: Floating-point conditional move (Double)

(6) Condition bit transfer instruction

- TRFSR: Transfers specified CC bit to zero flag in PSW (Single)

7.4.3 Conditions for Comparison Instructions

Floating-point comparison instructions (CMPF.D and CMPF.S) perform two floating-point data compare operations. The result is determined based on the comparison condition contained in the data and code. **Table 7-9** lists the mnemonics for conditions that can be specified by comparison instructions.

The comparison instruction result is transferred by the TRFSR instruction to the Z flag of PSW (program status word), and when performing a conditional branch, the condition logic is inverted and then can be used. **Table 7-10** shows logic inversion based on the true/false status of conditions. In a 4-bit condition code for a floating-point comparison instruction, the condition is specified in the “True” column of the table. The conditional branch instruction BT performs a branch when the comparison result is true, while BF performs a branch when the result is false.

Table 7-9 List of Conditions for Comparison Instructions

Mnemonic	Definition	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 7-10 Definitions of Condition Code Bits and Their Logical Inversions

Mnemonic	Condition Code fcond		Bit Definition of Condition Code fcond(3:0)				Inverted Logic
			Less than	Equal to	Unordered	Invalid Operation Exception Occurs when Unordered	
(True)	Decimal	Binary	fcond(2)	fcond(1)	fcond(0)	fcond(3)	(False)
F	0	0b0000	F	F	F	No	(T)
UN	1	0b0001	F	F	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)

7.4.4 Floating-Point Instruction Set

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

- Instruction format: Indicates how the instruction is written and its operand(s) (symbols are listed in **Table 7-11**).
- Operation: Indicates the function of the instruction. (symbols are listed in **Table 7-12**).
- Format: Indicates the instruction format (see **7.4.1 Instruction Formats**).
- Opcode: Indicates the instruction opcode in bit fields (symbols are listed in **Table 7-13**).
- Description: Describes the operation of the instruction.
- Supplement: Provides supplementary information on the instruction.

Table 7-11 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 **7.4.3 Conditions for Comparison Instructions**).

Table 7-12 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 7-13 Opcodes

Symbol	Explanation
R	Single bit data of code specifying reg1
r	Single bit data of code specifying reg2
w	Single bit data of code specifying reg3
W	Single bit data of code specifying reg4
I	Single bit data of immediate data (indicates higher bit of immediate data)
i	Single bit data of immediate data
fff	3-bit data that specifies the bit number (fcbit) of the condition bit that stores the result of a floating-point comparison instruction
FFFF	4-bit data corresponding to the mnemonic or condition code (fcond) of the comparison condition of a comparison instruction

<Floating-point instruction>

ABSF.D

Floating-point Absolute Value (Double)

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.

<Floating-point instruction>

ABSF.S

Floating-point Absolute Value (Single)

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.

<Floating-point instruction>

ADDF.D

Floating-point Add (Double)

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]

reg2 (B) reg1 (A)	+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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

ADDF.S

Floating-point Add (Single)

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]

reg2 (B) reg1 (A)	+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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.DL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.DUL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.DUW

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

Conversion to unsigned fixed-point format (double precision)

[Instruction format] CEILF.DUW reg2, reg3

[Operation] reg3 ← 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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.DW

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.SL

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

Conversion to fixed-point format (single precision)

[Instruction format] CEILF.SL reg2, reg3

[Operation] reg3 ← ceil 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 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.SUW

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

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 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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CEILF.SW

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CMOVF.D

Floating-point Conditional Move (Double)

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 Note		category	type	sub-op	

Note reg3: wwww! = 0

wwww ≠ 0000 (do not set reg3 to r0)

Remark fcbit: fff

[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.

<Floating-point condition instruction >

CMOVF.S

Floating-point Conditional Move (Single)

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	reg3Note		category	type	sub-op	

Note reg3: wwwww! = 0

wwwww ≠ 00000 (do not set reg3 to r0)

Remark fcbit: fff

[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.

<Floating-point instruction>

CMPF.D

Compare floating-point values (Double)

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 7-14 Comparison Conditions**.

If one of the values is not-a-number, and the MSB of the comparison condition “fcond” has been set, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are enabled, the comparison result is not set and processing is passed to the exception.
If the enable bits are not set, no exception occurs, and the preservation bit (bit 4) of the FPSR register is set, then the comparison result is set to the CC(7:0) bits of the FPSR register.
When SignalingNaN (S-NaN) is acknowledged as an operand value in a floating-point instruction (including a comparison), it is regarded as an invalid operation condition. When using only S-NaN but also QuietNaN (Q-NaN) for a comparison that is an invalid operation, it is simpler to use a program in which any NaN results in an error. In other words, there is no need to insert code that checks for Q-NaN that would result in an unordered result. Instead, the exception handling system should perform error processing when an exception occurs after detecting an invalid operation.
The following shows a comparison that checks for equivalence of two numerical values and triggers an error when an unordered result is detected.

Table 7-14 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 or equal to	No
OLE	6	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	No
ULE	7	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
SF	8	FALSE	Always false	Yes
NGLE	9	Unordered	One of reg1 and reg2 is not-a-number	Yes
SEQ	10	reg2 = reg1	Ordered (both reg1 and reg2 are not not-a-number) and equal	Yes
NGL	11	reg2 ? = reg1	Unordered (one of reg1 and reg2 is not-a-number) or equal	Yes
LT	12	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	Yes
NGE	13	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than	Yes
LE	14	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	Yes
NGT	15	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	Yes

Remark ?: 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]

<table>
<tr><th>reg1 (B)
reg2 (A)</th><th>+Normal</th><th>−Normal</th><th>+0</th><th>−0</th><th>+∞</th><th>−∞</th><th>Q-NaN</th><th>S-NaN</th></tr>
<tr><td>±Normal</td><td colspan="6" rowspan="3">Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)</td><td colspan="2" rowspan="3"></td></tr>
<tr><td>±0</td></tr>
<tr><td>±∞</td></tr>
<tr><td>Q-NaN</td><td colspan="8">Unordered</td></tr>
<tr><td>S-NaN</td><td colspan="8">Unordered [V]</td></tr>
</table>

[Condition code (fcond) = 8 to 15]

<table>
<tr><th>reg1 (B)
reg2 (A)</th><th>+Normal</th><th>−Normal</th><th>+0</th><th>−0</th><th>+∞</th><th>−∞</th><th>Q-NaN</th><th>S-NaN</th></tr>
<tr><td>±Normal</td><td colspan="6" rowspan="3">Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)</td><td colspan="2" rowspan="3"></td></tr>
<tr><td>±0</td></tr>
<tr><td>±∞</td></tr>
<tr><td>Q-NaN</td><td colspan="8" rowspan="2">Unordered [V]</td></tr>
<tr><td>S-NaN</td></tr>
</table>

Remark [ ] indicates an exception that must occur.

<Floating-point instruction>

CMPF.S

Compare floating-point values (Single)

Floating-point comparison (single precision)

[Instruction format] CMPF.S fcond, reg2, reg1, fcbit

CMPF.S fcond, reg2, reg1

[Operation]

```
if isNaN(reg1) or isNaN(reg2) then
  result.less ← 0
  result.equal ← 0
  result.unordered ← 1
  if fcond[3] == 1 then
    Invalid operation exception is detected.
  endif
else
  result.less ← 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 7-15 Comparison Conditions**.

If one of the values is not-a-number, and the MSB of the comparison condition “fcond” has been set, an IEEE754-defined invalid operation exception is detected. If invalid operation exceptions are enabled, the comparison result is not set and processing is passed to the exception.
If the enable bits are not set, no exception occurs, and the preservation bit (bit 4) of the FPSR register is set, then the comparison result is set to the CC(7:0) bits of the FPSR register.
When SignalingNaN (S-NaN) is acknowledged as an operand value in a floating-point instruction (including a comparison), it is regarded as an invalid operation condition. When using only S-NaN but also QuietNaN (Q-NaN) for a comparison that is an invalid operation, it is simpler to use a program in which any NaN results in an error. In other words, there is no need to insert code that explicitly checks for Q-NaN that would result in an unordered result. Instead, the exception handling system should perform error processing when an exception occurs after detecting an invalid operation. The following shows a comparison that checks for equivalence of two numerical values and triggers an error when an unordered result is detected.

Table7-15 Comparison Conditions

Comparison Conditions		Definition	Description	Detection of invalid operation exception by unordered?
	fcond			
F	0	FALSE	Always false	No
UN	1	Unordered	One of reg1 and reg2 is not-a-number	No
EQ	2	reg2 = reg1	Ordered (both reg1 and reg2 is not not-a-number) and equal	No
UEQ	3	reg2 ? = reg1	Unordered (at least, one of reg1 and reg2 is not-a-number) or equal	No
OLT	4	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	No
ULT	5	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
OLE	6	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	No
ULE	7	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	No
SF	8	FALSE	Always false	Yes
NGLE	9	Unordered	One of reg1 and reg2 is not-a-number	Yes
SEQ	10	reg2 = reg1	Ordered (both reg1 and reg2 are not not-a-number) and equal	Yes
NGL	11	reg2 ? = reg1	Unordered (one of reg1 and reg2 is not-a-number) or equal	Yes
LT	12	reg2 < reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than	Yes
NGE	13	reg2 ? < reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than	Yes
LE	14	reg2 ≤ reg1	Ordered (both reg1 and reg2 are not not-a-number) and less than or equal to	Yes
NGT	15	reg2 ? ≤ reg1	Unordered (one of reg1 and reg2 is not-a-number) or less than or equal to	Yes

Remark ?: Unordered (invalid comparison)

# When explicitly testing Q-NaN

```
      CMPF.S     OLT, r12, r13, 0  # Check if r12 < r14
      CMPF.S     UN, r12, r13, 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, r13, 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]

<table>
<tr><th>reg1 (B)
reg2 (A)</th><th>+Normal</th><th>−Normal</th><th>+0</th><th>−0</th><th>+∞</th><th>−∞</th><th>Q-NaN</th><th>S-NaN</th></tr>
<tr><td>±Normal</td><td rowspan="3" colspan="6">Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)</td><td rowspan="3" colspan="2"></td></tr>
<tr><td>±0</td></tr>
<tr><td>±∞</td></tr>
<tr><td>Q-NaN</td><td colspan="8">Unordered</td></tr>
<tr><td>S-NaN</td><td colspan="8">Unordered [V]</td></tr>
</table>

[Condition code (fcond) = 8 to 15]

<table>
<tr><th>reg1 (B)
reg2 (A)</th><th>+Normal</th><th>−Normal</th><th>+0</th><th>−0</th><th>+∞</th><th>−∞</th><th>Q-NaN</th><th>S-NaN</th></tr>
<tr><td>±Normal</td><td rowspan="3" colspan="6">Stores result of comparison (true or false) executed under the comparison condition (fcond) in the FPSR.CCn bit (n = fcbit)</td><td rowspan="3" colspan="2"></td></tr>
<tr><td>±0</td></tr>
<tr><td>±∞</td></tr>
<tr><td>Q-NaN</td><td rowspan="2" colspan="8">Unordered [V]</td></tr>
<tr><td>S-NaN</td></tr>
</table>

Remark [ ] indicates an exception that must occur.

<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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.DS

Floating-point Convert Double to Single (Double)

Conversion to floating-point format (double precision)

[Instruction format] CVTF.DS reg2, reg3

[Operation] reg3 ← 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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.DUL

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

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 +∞: $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.DUW

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

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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.DW

Floating-point Convert Double to Word (Double)

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 +∞: $2^{31} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.HS

Floating-point Convert Half to Single (Single)

Conversion to floating-point format (single precision)

[Instruction format] CVTF.HS reg2, reg3

[Operation] reg3 ← 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 (Half)		+0	−0	+∞	−∞	Q-NaN	Q-NaN [V]

Remark [ ] indicates an exception that must occur.

<Floating-point instruction>

CVTF.LD

Floating-point Convert Long to Double (Double)

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

<Floating-point instruction>

CVTF.LS

Floating-point Convert Long to Single (Single)

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

<Floating-point instruction>

CVTF.SD

Floating-point Convert Single to Double (Double)

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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.SL

Floating-point Convert Single to Long (Single)

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.SH

Floating-point Convert Single to Half (Single)

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

[Instruction format] CVTF.SH reg2, reg3

[Operation] reg3 ← 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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.SUL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.SUW

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

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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

CVTF.ULD

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

Conversion to floating-point format (double precision)

[Instruction format] CVTF.ULD reg2, reg3

[Operation] reg3 ← 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

<Floating-point instruction>

CVTF.ULS

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

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

<Floating-point instruction>

CVTF.UWD

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

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

<Floating-point instruction>

CVTF.UWS

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

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

<Floating-point instruction>

CVTF.WD

Floating-point Convert Word to Double (Double)

Conversion to floating-point format (double precision)

[Instruction format] CVTF.WD reg2, reg3

[Operation] reg3 ← 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

<Floating-point instruction>

CVTF.WS

Floating-point Convert Word to Single (single)

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

<Floating-point instruction>

DIVF.D

Floating-point Divide (Double)

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				+∞	–∞	Q-NaN	Q-NaN [V]
–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]							

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

DIVF.S

Floating-point Divide (Single)

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]

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

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.DL

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

Conversion to fixed-point format (double precision)

[Instruction format] FLOORF.DL reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.DUL

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

Conversion to unsigned fixed-point format (double precision)

[Instruction format] FLOORF.DUL reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.DW

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

Conversion to fixed-point format (double precision)

[Instruction format] FLOORF.DW reg2, reg3

[Operation] reg3 ← floor 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 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 –∞ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.SL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.SUL

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

Conversion to unsigned fixed-point format (single precision)

[Instruction format] FLOORF.SUL reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

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

FLOORF.SUW

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 $2^{64} - 1$ 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

FLOORF.SW

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

Conversion to fixed-point format (single precision)

[Instruction format] FLOORF.SW reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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)	reg2 (B) reg1 (A)	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
±Normal	+Normal	FMA (A, B, C)				+∞	–∞		
	–Normal					–∞	+∞		
	±0					Q-NaN [V]			
	+∞	+∞	–∞	Q-NaN [V]		+∞	–∞		
	–∞	–∞	+∞			–∞	+∞		
±0	+Normal	FMA (A, B, 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								

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

[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.

<Floating-point instruction>

FMSF.S

Floating-point Fused-Multiply-subtract (Single)

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 (A, B, C)				+∞	−∞		
	−Normal					−∞	+∞		
	±0					Q-NaN [V]			
	+∞	+∞	−∞	Q-NaN [V]		+∞	−∞		
	−∞	−∞	+∞			−∞	+∞		
±0	+Normal	FMS (A, B, 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								

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

[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.

<Floating-point instruction>

FNMAF.S

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

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) \ r eg2 (B)	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	FNMA (A, B, C)				−∞	+∞	Q-NaN	Q-NaN [V]
	−Normal					+∞	−∞		
	±0					Q-NaN [V]			
	+∞	−∞	+∞	Q-NaN [V]		−∞	+∞		
	−∞	+∞	−∞			+∞	−∞		
±0	+Normal	FNMA (A, B, 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								
Don’t care	S-NaN								
S-NaN	Don’t care								

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

[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.

<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	FNMS (A, B, C)				−∞	+∞	Q-NaN	Q-NaN [V]
	−Normal					+∞	−∞		
	±0					Q-NaN [V]			
	+∞	−∞	+∞	Q-NaN [V]		−∞	+∞		
	−∞	+∞	−∞			+∞	−∞		
±0	+Normal	FNMS (A, B, 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								
Don’t care	S-NaN								
S-NaN	Don’t care								

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

[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.

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

Remark [ ] indicates an exception that must occur.

<Floating-point instruction>

MAXF.S

Floating-point Maximum (Single)

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]

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

Remark [ ] indicates an exception that must occur.

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

Remark [ ] indicates an exception that must occur.

<Floating-point instruction>

MINF.S

Floating-point Minimum (Single)

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]

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

Remark [ ] indicates an exception that must occur.

<Floating-point instruction>

MULF.D

Floating-point Multiply (Double)

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 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.

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

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

MULF.S

Floating-point Multiply (Single)

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

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

NEGF.D

Floating-point Negate (Double)

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 result 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.

<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.

<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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

RECIPF.S

Reciprocal of a Floating-point Value (Single)

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]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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]	$1/\sqrt{A}$ [I]	Q-NaN [V]	+∞ [Z]	–∞ [Z]	+0	Q-NaN [V]	Q-NaN	Q-NaN [V]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

RSQRTF.S

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

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]	$1/\sqrt{A}$ [I]	Q-NaN [V]	+∞ [Z]	−∞ [Z]	+0	Q-NaN [V]	Q-NaN	Q-NaN [V]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

SQRTF.D

Floating-point Square Root (Double)

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]	$\sqrt{A}$	Q-NaN [V]	+0	–0	+∞	Q-NaN [V]	Q-NaN	Q-NaN [V]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

SQRTF.S

Floating-point Square Root (Single)

Square root (single precision)

[Instruction format] SQRTF.S reg2, reg3

[Operation] reg3 ← sqrt reg2

[Format] Format F:I

[Opcode]

15 11	10 5	4 0	31 27	26	25 23	22 21	20 17	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]	$\sqrt{A}$	Q-NaN [V]	+0	–0	+∞	Q-NaN [V]	Q-NaN	Q-NaN [V]

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

SUBF.D

Floating-point Subtract (Double)

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]

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

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

SUBF.S

Floating-point Subtract (Single)

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]

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

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRFSR

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

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

<Floating-point instruction>

TRNCF.DL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRNCF.DUL

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

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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRNCF.DUW

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

Conversion to unsigned fixed-point format (double precision)

[Instruction format] TRNCF.DUW reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRNCF.SL

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

Conversion to fixed-point format (single precision)

[Instruction format] TRNCF.SL reg2, reg3

[Operation] reg3 ← 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRNCF.SUW

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

Conversion to unsigned fixed-point format (single precision)

[Instruction format] TRNCF.SUW reg2, reg3

[Operation] reg3 ← 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 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

<Floating-point instruction>

TRNCF.SW

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

Conversion to fixed-point format (single precision)

[Instruction format] TRNCF.SW reg2, reg3

[Operation] reg3 ← 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 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]		

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

2. When the FS bit of the FPSR register is 1, subnormal numbers are flushed to the normalized numbers shown in **6.1.9 Flushing Subnormal Numbers**.

CHAPTER 8 RESET

8.1 Status of Registers After Reset

If a reset signal is input by a method defined by the hardware specifications, the program registers and system registers are placed in the status shown by the value after reset of each register in **CHAPTER 3 REGISTER SET**, and program execution is started. Set the contents of each register to an appropriate value in the program.

The CPU executes a reset to start execution of a program from the reset address specified by **4.5 Exception Handler Address**.

Note that because the PSW.ID bit is set (1) immediately after a reset, conditional EI level exceptions will not be acknowledged. To acknowledge conditional EI level exceptions, clear (0) the PSW.ID bit.

APPENDIX A HAZARD RESOLUTION PROCEDURE FOR SYSTEM REGISTERS

Certain system registers require the following procedures to resolve hazards when their values are updated by the LDSR instruction.

- Instruction fetching

When an instruction is to be fetched after updating a register covered by the description below, after executing the instruction to update the register, only allow the instruction fetch to start after execution of an EIRET, FERET, or SYNCI instruction.

- PSW.UM, MCFG0.SPID

When an instruction is to be fetched after updating a register covered by the description below, execute the instruction to update the register before allowing the instruction fetch to start.

- All registers related to ASID and MPU (register number: SR*,5-7)

- SYSCALL instruction

When a SYSCALL instruction is to be executed after updating the register below, execute a SYNCP instruction after the instruction to update the register and before the SYSCALL instruction.

-SCCFG

- Load/Store

When an instruction associated with Load/Store after updating the registers below, execute a SYNCP instruction after executing the instruction to update the registers before Load/Store instruction.

-ASID, MPU protection area setting register (Register number: SR*,6-7)

- Interrupt

Update the registers below when interrupt is inhibited. (PSW.ID=1).

-PSW.EBV, EBASE, INTBP, FPIPR, ISPR, PMR, ICSR, INTCFG

- Operation to clear instruction cache

When completion of instruction cache clearance is confirmed, check the read value of the ICCTRL.ICHCLR bit.

- FPU register update

After executing update instruction of the registers below, execute a SYNCP, EIRET or FERET instruction.

-All FPU-related registers (Register number: SR6-11,0)

- Change of FPP/FPI exception mode

When the FPP/FPI exception mode is changed, execute instructions of SYNCP and SYNCE first, and update the register below.

To update registers, proceed “FPU register update” above also.

-FPSR.PEM

Remark: Executing instructions other than the floating-point operation instruction that generates an FPP/FPI exception is possible among the SYNCP, SYNCE, and the instruction to update the register above.

- Coprocessor instruction

When a coprocessor instruction (floating-point operation instruction) is to be executed after updating the register below, execute instructions of EIRET, FERET, SYNCI or SYNCP after executing the instruction to update the registers and before executing a coprocessor instruction.

-PSW.CU0

APPENDIX B NUMBER OF INSTRUCTION EXECTUION CLOCKS

B.1 Number of G3M Instruction Execution Clocks

1) Basic instruction

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]
		disp23 [reg1] , reg3	6	1	1	3[*1]
	LD.BU	disp16 [reg1] , reg2	4	1	1	3[*1]
		disp23 [reg1] , reg3	6	1	1	3[*1]
	LD.H	disp16 [reg1] , reg2	4	1	1	3[*1]
		disp23 [reg1] , reg3	6	1	1	3[*1]
	LD.HU	disp16 [reg1] , reg2	4	1	1	3[*1]
		disp23 [reg1] , reg3	6	1	1	3[*1]
	LD.W	disp16 [reg1] , reg2	4	1	1	3[*1]
		disp23 [reg1] , reg3	6	1	1	3[*1]
	LD.DW	disp23 [reg1] , reg3	6	1	1	3[*1]
ep relative	SLD.B	disp7 [ep] , reg2	2	1	1	3[*1]
	SLD.BU	disp4 [ep] , reg2	2	1	1	3[*1]
	SLD.H	disp8 [ep] , reg2	2	1	1	3[*1]
	SLD.HU	disp5 [ep] , reg2	2	1	1	3[*1]
	SLD.W	disp8 [ep] , reg2	2	1	1	3[*1]
Store instrucrion	ST.B	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.H	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.W	reg2, disp16 [reg1]	4	1	1	1
		reg3, disp23 [reg1]	6	1	1	1
	ST.DW	reg3, disp23 [reg1]	6	1	1	1
ep relative	SST.B	reg2, disp7 [ep]	2	1	1	1
	SST.H	reg2, disp8 [ep]	2	1	1	1
	SST.W	reg2, disp8 [ep]	2	1	1	1
Multiplication instruction	MUL	reg1, reg2, reg3	4	1	1	3
		imm9, reg2, reg3	4	1	1	3
	MULH	reg1, reg2	2	1	1	3
		imm5, reg2	2	1	1	3
	MULHI	imm16, reg1, reg2	4	1	1	3
	MULU	reg1, reg2, reg3	4	1	1	3
		imm9, reg2, reg3	4	1	1	3
Multiply-accumulate operation	MAC	reg1, reg2, reg3, reg4	4	1	1	3
	MACU	reg1, reg2, reg3, reg4	4	1	1	3
Arithmetic instruction	ADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
	MOV	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		imm32, reg1	6	1	1	1
	MOVEA	imm16, reg1, reg2	4	1	1	1
	MOVHI	imm16, reg1, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
Operation with condition	ADF	cccc, reg1, reg2, reg3	4	1	1	1
	SBF	cccc, reg1, reg2, reg3	4	1	1	1
Saturated operation	SATADD	reg1, reg2	2	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
	SATSUBR	reg1, reg2	2	1	1	1
Logical instruction	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
	XORI	imm16, reg1, reg2	4	1	1	1
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
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
		cccc, imm5, reg2, reg3	4	1	1	1
	HSH	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	ROTL	imm5, reg2, reg3	4	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SASF	cccc, reg2	4	1	1	1

1) Basic instruction

Type of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Excution Clocks		
				issue	repeat	latency
	SETF	cccc, reg2	4	1	1	1
	SHL	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SHR	reg1, reg2	4	1	1	1
		imm5, reg2	2	1	1	1
		reg1, reg2, reg3	4	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1
Bit search instruction	SCH0L	reg2, reg3	4	1	1	1
	SCH0R	reg2, reg3	4	1	1	1
	SCH1L	reg2, reg3	4	1	1	1
	SCH1R	reg2, reg3	4	1	1	1
Division instruction	DIV	reg1, reg2, reg3	4	19	19	19
	DIVH	reg1, reg2	2	19	19	19
		reg1, reg2, reg3	4	19	19	19
	DIVHU	reg1, reg2, reg3	4	19	19	19
	DIVU	reg1, reg2, reg3	4	19	19	19
High-speed divide operation	DIVQ	reg1, reg2, reg3	4	N+3[*2]	N+3[*2]	N+3[*2]
	DIVQU	reg1, reg2, reg3	4	N+3[*2]	N+3[*2]	N+3[*2]
Branch instructions	Bcond	disp9 (When Branch prediction is matched)	2	1[*3]	1[*3]	1[*3]
		disp9 (When Branch prediction is not matched)	2	4[*3]	4[*3]	4[*3]
	Bcond	disp17 (When Branch prediction is matched)	4	1[*3]	1[*3]	1[*3]
		disp17 (When Branch prediction is not matched)	4	4[*3]	4[*3]	4[*3]
	JARL	disp22, reg2	4	4	4	4
		disp32, reg1	6	4	4	4
		[reg1], reg3	4	4	4	4
	JMP	[reg1]	2	4	4	4
		disp32 [reg1]	6	5	5	5
	JR	disp22 (When Branch prediction is matched)	4	1[*3]	1[*3]	1[*3]
		disp22 (When Branch prediction is not matched)	4	4[*3]	4[*3]	4[*3]
	JR	disp32 (When Branch prediction is matched)	6	1[*3]	1[*3]	1[*3]
		disp32 (When Branch prediction is not matched)				
Loop instruction	LOOP	reg1, disp16 (When Branch prediction is matched)	4	1[*3]	1[*3]	1[*3]
		reg1, disp16 (When Branch prediction is not matched)	4	4[*3]	4[*3]	4[*3]
Bit manipulation instruction	CLR1	bit#3, disp16 [reg1]	4	4[*4]	4[*4]	4[*4]
		reg2, [reg1]	4	4[*4]	4[*4]	4[*4]
	NOT1	bit#3, disp16 [reg1]	4	4[*4]	4[*4]	4[*4]
		reg2, [reg1]	4	4[*4]	4[*4]	4[*4]
	SET1	bit#3, disp16 [reg1]	4	4[*4]	4[*4]	4[*4]
		reg2, [reg1]	4	4[*4]	4[*4]	4[*4]
	TST1	bit#3, disp16 [reg1]	4	4[*4]	4[*4]	4[*4]
		reg2, [reg1]	4	4[*4]	4[*4]	4[*4]
Special instruction						
Table reference branch	SWITCH	reg1	2	8	8	8
Sub routine call	CALLT	imm6	2	10	10	10
	CTRET	—	4	7	7	7
System call exception	SYSCALL	vector8	4	10	10	10
Software exception	FETRAP	vector4	2	7	7	7
	TRAP	vector5	4	7	7	7
Return from exception processing	EIRET	—	4	7	7	7
	FERET	—	4	7	7	7
EI level interrupt	DI	—	4	3	3	3
	EI	—	4	3	3	3
Restoration from & storage on stack	DISPOSE	imm5, list12	4	n+2[*5]	n+2[*5]	n+2[*5]
		imm5, list12, [reg1]	4	n+4[*5]	n+4[*5]	n+4[*5]
	PREPARE	list12, imm5	4	n+2[*5]	n+2[*5]	n+2[*5]
		list12, imm5, sp	4	n+2[*5]	n+2[*5]	n+2[*5]
		list12, imm5, imm16	6	n+2[*5]	n+2[*5]	n+2[*5]
		list12, imm5, imm16<<16	6	n+2[*5]	n+2[*5]	n+2[*5]
		list12, imm5, imm32	8	n+2[*5]	n+2[*5]	n+2[*5]
	POPSP	rh-rt	4	n+2[*7]	n+2[*7]	n+2[*7]
	PUSHSP	rh-rt	4	n+2[*7]	n+2[*7]	n+2[*7]
System register operation	LDSR	reg2, regID, selID	4	3[*6]	3	3
	STSR	regID, reg2, selID	4	1	1	1
Exclusive control	CAXI	[reg1], reg2, reg3	4	4[*4]	4[*4]	4[*4]
	LDL.W	[reg1], reg3	4	1	1	3[*2]
	STC.W	reg3, [reg1]	4	1	1	1
Stop	HALT	—	4	1	1	1
	SNOOZE	—	4	Undefined	Undefined	Undefined
Synchronization	SYNCE	—	2	Undefined	Undefined	Undefined
	SYNCI	—	2	Undefined	Undefined	Undefined
	SYNCM	—	2	Undefined	Undefined	Undefined
	SYNCP	—	2	Undefined	Undefined	Undefined

1) Basic instruction

Type of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Others	NOP	—	2	1	1	1
	RIE	—	4	7	7	7

2) Cache instruction

Type of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks		
				issue	repeat	latency
Cache operation instruction	CACHE	cacheop, [reg1]	4	1	1	Undefined
Pre-fetch instruction	PREF	prefop, [reg1]	4	1	1	1

3) Floating-point operation instruction – single precision –

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks (Imprecise)			Number of Execution Clocks (Precise)		
				issue	repeat	latency *8	issue	repeat	latency
Floating-point arithmetic operation	ABSF.S	reg2, reg3	4	1	1	4	7	7	7
	ADDF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
	NEGF.S	reg2, reg3	4	1	1	4	7	7	7
	SUBF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
Floating-point multiplication	MULF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
Multiply-accumulate/ subtract operation	FMAF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
	FMSF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
	FNMAF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
	FNMSF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
Floating-point subtraction	DIVF.S	reg1, reg2, reg3	4	14 *9	14	17	20	20	20
Square root of a Floatingpoint value /Reciprocal	RECIPF.S	reg2, reg3	4	10 *9	10	13	16	16	16
	RSQRTF.S	reg2, reg3	4	14 *9	14	17	20	20	20
	SQRTF.S	reg2, reg3	4	14 *9	14	17	20	20	20
Conversion between floating-point formats/ Conversion between fixed-point and flating point formats	CVTF.HS	reg2, reg3	4	1	1	4	7	7	7
	CVTF.LS	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SH	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SL	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SUL	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SUW	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SW	reg2, reg3	4	1	1	4	7	7	7
	CVTF.ULS	reg2, reg3	4	1	1	4	7	7	7
	CVTF.UWS	reg2, reg3	4	1	1	4	7	7	7
	CVTF.WS	reg2, reg3	4	1	1	4	7	7	7
	CEILF.SL	reg2, reg3	4	1	1	4	7	7	7
	CEILF.SUL	reg2, reg3	4	1	1	4	7	7	7
	CEILF.SUW	reg2, reg3	4	1	1	4	7	7	7
	CEILF.SW	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.SL	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.SUL	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.SUW	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.SW	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.SL	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.SUL	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.SUW	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.SW	reg2, reg3	4	1	1	4	7	7	7
Floating-point comparison	CMPF.S	cond, reg1, reg2, cc	4	1	1	4	7	7	7
Transfer with conditions	CMOVF.S	cc, reg1, reg2, reg3	4	1	1	4	7	7	7
Bit transfer with conditions	TRFSR	cc	4	1	1	1	1	1	1
Floating-point maximum/ minimum values	MAXF.S	reg1, reg2, reg3	4	1	1	4	7	7	7
	MINF.S	reg1, reg2, reg3	4	1	1	4	7	7	7

3) Floating-point operation instruction – single precision –

Types of Instructions	Mnemonics	Operand	Instruction Length (Number of Bytes)	Number of Execution Clocks (Imprecise)			Number of Execution Clock (Precise)		
				issue	repeat	latency *8	issue	repeat	latency
Floating-point arithmetic operation	ABSF.D	reg2, reg3	4	1	1	4	7	7	7
	ADDF.D	reg1, reg2, reg3	4	1	1	4	7	7	7
	NEGF.D	reg2, reg3	4	1	1	4	7	7	7
	SUBF.D	reg1, reg2, reg3	4	1	1	4	7	7	7
Floating-point multiplication	MULF.D	reg1, reg2, reg3	4	2 *9	2	5	8	8	8
Floating-point division	DIVF.D	reg1, reg2, reg3	4	30 *9	30	33	36	36	36
Square root of a Floatingpoint value /Reciprocal	RECIPF.D	reg2, reg3	4	26 *9	26	29	32	32	32
	RSQRTF.D	reg2, reg3	4	36 *9	36	39	42	42	42
	SQRTF.D	reg2, reg3	4	30 *9	30	33	36	36	36
Conversion between floating-point formats/ Conversion between fixed-point and floating point formats	CVTF.DL	reg2, reg3	4	1	1	4	7	7	7
	CVTF.DS	reg2, reg3	4	1	1	4	7	7	7
	CVTF.DUL	reg2, reg3	4	1	1	4	7	7	7
	CVTF.DUW	reg2, reg3	4	1	1	4	7	7	7
	CVTF.DW	reg2, reg3	4	1	1	4	7	7	7
	CVTF.LD	reg2, reg3	4	1	1	4	7	7	7
	CVTF.SD	reg2, reg3	4	1	1	4	7	7	7
	CVTF.ULD	reg2, reg3	4	1	1	4	7	7	7
	CVTF.UWD	reg2, reg3	4	1	1	4	7	7	7
	CVTF.WD	reg2, reg3	4	1	1	4	7	7	7
	CEILF.DL	reg2, reg3	4	1	1	4	7	7	7
	CEILF.DUL	reg2, reg3	4	1	1	4	7	7	7
	CEILF.DUW	reg2, reg3	4	1	1	4	7	7	7
	CEILF.DW	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.DL	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.DUL	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.DUW	reg2, reg3	4	1	1	4	7	7	7
	FLOORF.DW	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.DL	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.DUL	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.DUW	reg2, reg3	4	1	1	4	7	7	7
	TRNCF.DW	reg2, reg3	4	1	1	4	7	7	7
Floating-point comparison	CMPF.D	cond, reg1, reg2, cc	4	1	1	4	7	7	7
Transfer with conditions	CMOVF.D	cc, reg1, reg2, reg3	4	1	1	4	7	7	7
Floating-point maximum/ minimum values	MAXF.D	reg1, reg2, reg3	4	1	1	4	7	7	7
	MINF.D	reg1, reg2, reg3	4	1	1	4	7	7	7

Note 1. When no waiting is required (3 + number of wait states for read access).

Note 2. N = int (((number of valid bits in absolute value of dividend) – (number of valid bits in absolute value of divisor)) ÷ 2) + 1
If the result for N < 1, N becomes 1. Division by 0 leads to N being 0. The range of N is from 0 to 16.

Note 3. Executing an instruction to rewrite the contents of the PSW register immediately beforehand does not affect the number of clock cycles for execution. Even if an immediately preceding instruction has rewritten the contents of a register, parallel execution is possible.

Note 4. When no waiting is required (4 + number of wait states for read access)

Note 5. “n” depends on the total number of registers specified in the list and on the register numbers. When a pair of consecutively numbered registers, such as r21 and r22, is specified, the operation will be completed in a single round of access. When a pair of non-consecutive registers, such as r29 and r31, is specified, multiple rounds of access may be required to complete the operation.
When no waiting is required, the values are as shown below.
PREPARE: Minimum value is 1, maximum value is 3
DISPOSE: Minimum value is 1, maximum value is 6 (if accompanied by JMP, add 2 clock cycles).
When n is 0, 1 clock cycle is required. (DISPOSE instruction with JMP: 3 clock cycles)

Note 6. In case of access to system registers which control operation (e.g. the PSW), stop the subsequent instructions. If they do not stop, issue becomes set to 1.

Note 7. “n” depends on the total number of registers specified in the sequence of register numbers (rh-rt) and on the register numbers.
When any pair of consecutive registers, such as r21 and r22, is specified, the operation is completed in a single round of access.
When a pair of non-consecutive registers, such as r29 and r31, is specified, multiple rounds of access may be required to complete the operation.
When no waiting is required, the values will be as follows:
PUSHSH: Minimum 1, maximum 8
POPSP: Minimum 1, maximum 16

Note 8. latency for the floating-point operation instruction, whereas latency + 1 for the instructions other than the floating-point operation instruction.

Note 9. In precise mode, issue for the subsequent instructions other than the floating-point operation instruction is 1.

Remark 1. Example of execution clocks

Symbol	Description
issue	When the other instruction is executed immediately after the execution of the current instruction
repeat	When the same instruction is repeated immediately after the execution of the current instruction
latency	When the following instruction uses the result of the current instruction

APPENDIX C REGISTER INDEX

APPENDIX D INSTRUCTION INDEX

APPENDIX E REVISION HISTORY

E.1 Major Revisions in This Edition

Page	Description	Classification
CHAPTER 3 REGISTER SET		
62	Table 3-21 Instructions Causing Exceptions and Values of MEI Register: RW of CACHE, modified	(c)
108	3.6.1, (7) ICERR - Instruction cache error: Description added	(c)
CHAPTER 5 MEMORY MANAGEMENT		
149	5.1.7, (2) Sample code: Modified (be → bnz)	(a)
155	5.2.7 Memory Protection for CACHE and PREF Instructions: Description modified	(c)
CHAPTER 6 COPROCESSOR		
168	6.1.2, (3) Expanded floating-point format: Description modified (single-precision → half-precision)	(a)
184	6.1.11 Flush to Nearest: Description modified	(b)

Remark: The classification in the table above means as follows.

(a): Error correction (b): Specifications added or changed (c): descriptions or notes added or changed

RH850G3M　User's Manual: Software

Publication Date:	Rev.0.51	Aug 07, 2014
	Rev.1.00	Aug 28, 2014
	Rev.1.10	Sep 15, 2015
	Rev.1.20	Nov 30, 2015
	Rev.1.30	Jun 30, 2016
	Rev.1.40	Dec 22, 2016

Published by:　Renesas Electronics Corporation

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

RH850G3M

Renesas Electronics Corporation