User's Manual

32

V850E2M

User's Manual: Architecture

RENESAS MCU
V850E2M Microprocessor Core

Renesas Electronics
www.renesas.com

Rev.1.00 Oct, 2012

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.

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 V_{IL} (MAX) and V_{IH} (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 V_{IL} (MAX) and V_{IH} (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 V_{DD} 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 Readers

This manual is intended for users who wish to understand the functions of the V850E2M CPU core for designing application systems using the V850E2M CPU core.

Purpose

This manual is intended for users to understand the architecture of the V850E2M CPU core described in the Organization below.

Organization

This manual contains the following information:

- Basic function
- Processor protection function
- Floating-point operation function

How to Use this Manual

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

To learn about the hardware functions,

→ Read Hardware User's Manual of each product.

To learn about the functions of a specific instruction in detail,

→ Read **PART 2 CHAPTER 5 INSTRUCTIONS,**
PART 4 CHAPTER 4 INSTRUCTIONS.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	×××B (B is appended to pin or signal name)
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numerical representation:	Binary ... ×××× or ××××B
	Decimal ... ××××
	Hexadecimal ... ××××H

Prefix indicating the 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

PART 1 OVERVIEW

CHAPTER 1 FEATURES

The V850E2M CPU conforms to the V850E2v3 architecture and is designed for microcontrollers that control embedded systems based on the concept of high performance, high functionality, and high reliability.

The V850E2M CPU supplies the following functions.

(1) Basic function
(2) Processor protection function
(3) Floating-point operation function

The V850E2M CPU executes almost all instructions, such as for address calculation, arithmetic and logic operations, and data transfer, with one clock under control of a 7-stage pipeline.

The V850E2M CPU can use the software resources of the conventional system as is because it is upwardly compatible with the V850 CPU, V850E1 CPU, and V850E2 CPU at object code level.

1.1 Basic function

Basic integer operation instructions allowing general data processing and control programming, and special instructions for application program optimization are provided. In addition, flexible exception processing functions that allow high-reliability programming, exclusive control instructions that enable data to be shared in a multi-core environment, and load/store instructions with an extended displacement range are also provided.

The basic funciton mainly consists of an instruction queue, program counter, execution unit, general-purpose registers, system registers, and control block. The execution unit contains dedicated hardware such as an ALU, LD/ST unit, multiplier (32 bits x 32 bits), barrel shifter (32 bits/clock), and divider, to execute complicated processing.

For details of the basic function, see **PART 2 BASIC FUNCTION**.

1.2 Processor protection function

The processor protection function that protects resources, such as the memory, peripheral devices, system registers, and CPU time of each program, thereby protecting the system from illegal use is provided.

The processor protection function is a function to guarantee high-reliability operations of a CPU which consists of system register protection, memory protection, peripheral unit protection, and timing supervision function.

For details of the processor protection function, see **PART 3 PROCESSOR PROTECTION FUNCTION**.

1.3 Floating-point operation function

The V850E2M CPU can implement a floating-point operation function (FPU) as a coprocessor.

The floating-point unit (FPU) of the V850E2M CPU conforms to the ANSI/IEEE standard 754-1985 (IEEE binary floating-point operation standard) and is provided with double-precision and single-precision floating-point operation instructions.

In addition, high-performance operations that can be executed with a high throughput, and high-reliability operations that can perform accurate exception processing can be selected depending on the situation.

The floating-point operation function depends on the products. Refer to Hardware User's Manual of each product.

For details of the floating-point operation function, see **PART 4 FLOATING-POINT OPERATION FUNCTION**.

PART 2 BASIC FUNCTION

CHAPTER 1 OVERVIEW

The V850E2M CPU conforms to the V850E2v3 Architecture, and it supplies basic operations to establish OS and application programs, and basic functions to manage exceptions.

(1) Integer operation instructions

Basic integer operation instructions that allow general data processing and control programming are provided. In addition, the conventional load/store instructions are extended with a 23-bit displacement format added.

(2) Special instructions

Instructions useful for optimizing an application program, such as stack frame manipulation instructions and common function call instructions, are provided.

(3) Multi-core support

Exclusive control functions and memory synchronization functions necessary for exclusive control between two or more CPUs are provided.

(4) High-function OS support

Instructions dedicated to supporting development of high-function OS are provided.

(5) Flexible and high-performance exception processing

Various exception processing functions that enable high-reliability programming are provided.

1.1 Features

(1) Advanced 32-bit architecture for embedded control

- Number of instructions: 98
- 32-bit general-purpose registers: 32 registers
- Load/store instructions with multiple displacement formats
  - Long (32 bits)
  - Middle (16 bits)
  - Short (8 bits)
- 3 operand instructions
- Address space: Program area … 4 GB linear
  Data area … 4 GB linear

(2) Instructions used for various application fields

- Saturated operation instructions
- Bit manipulation instructions

- Multiply instructions (on-chip hardware multiplier enable single-clock multiplication processing)
  16 bits × 16 bits → 32 bits
  32 bits × 32 bits → 32 bits or 64 bits
- MAC operation instructions
  32 bits × 32 bits + 64 bits → 64 bits
- High-speed division instruction
  This division instruction detects a valid bit length and changes it to the minimum number of execution cycles.
  32 bits ÷ 32 bits → 32 bits (quotient), 32 bits (remainder)

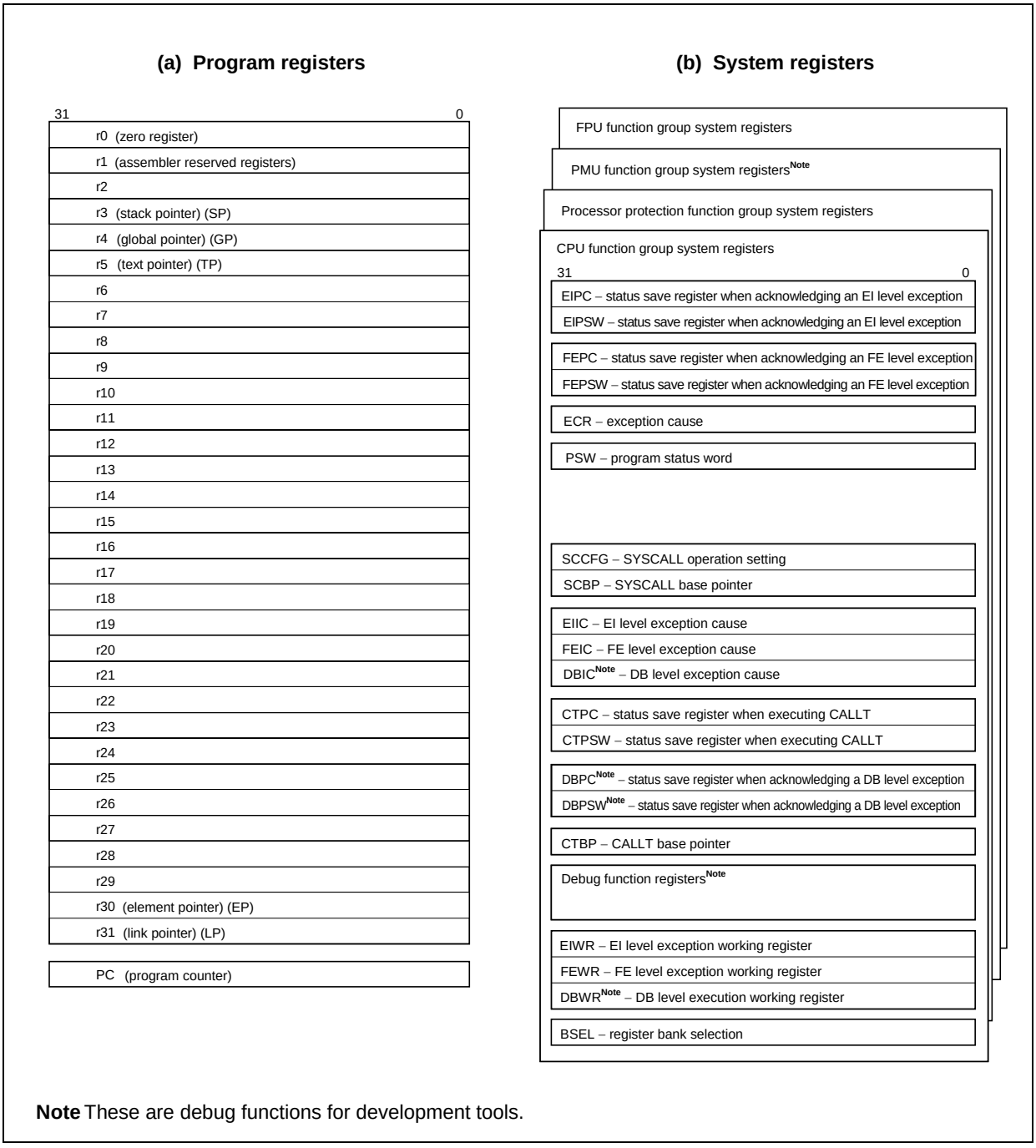
(3) Instructions suitable for high-function/high-performance programming

- Stack frame manipulation instruction
- Exclusive control instruction
- System call instruction (OS service calling instruction)
- Synchronization instruction (event control)

CHAPTER 2 REGISTER SET

There are two types of registers related to the basic functions: program registers that are used for ordinary programs and system registers that are used to control the execution environment. All are 32-bit registers.

Figure 2-1. Register List



Note These are debug functions for development tools.

2.1 Program Registers

Program registers includes general-purpose registers (r0 to r31) and the program counter (PC).

Table 2-1. Program Register List

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 (when the real-time OS being used does not use this register)	
	r3	Stack pointer (SP)	Used for stack frame generation when functions are called
	r4	Global pointer (GP)	When to access global variable in 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 base pointer for generating addresses when accessing memory
	r31	Link pointer (LP)	Used when 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, refer to the document of each software development environment.

(1) General-purpose registers (r0 to r31)

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.

The following points must be noted when using r0 to r5, r30, and r31 because these registers are assumed to be used for special purposes in a software development environment.

(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 instruction and DISPOSE instruction.

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.

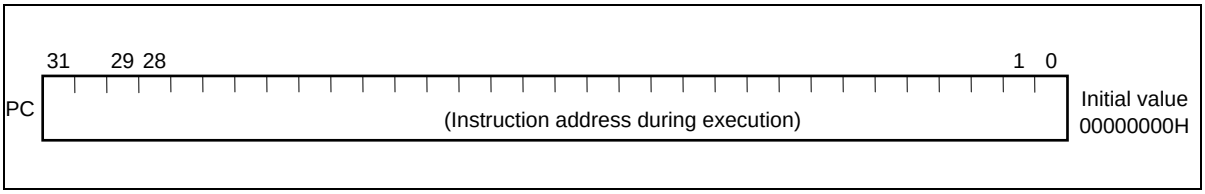
When using these registers, register contents must first be saved so they are not lost and can be restored after the registers are used.

(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.

(2) Program counter (PC)

The PC retains instruction addresses during program execution. Bit 0 is fixed to 0, and branching to an odd number address is disabled.



Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 is automatically set to bits 31 to 29.

2.2 System Register Bank

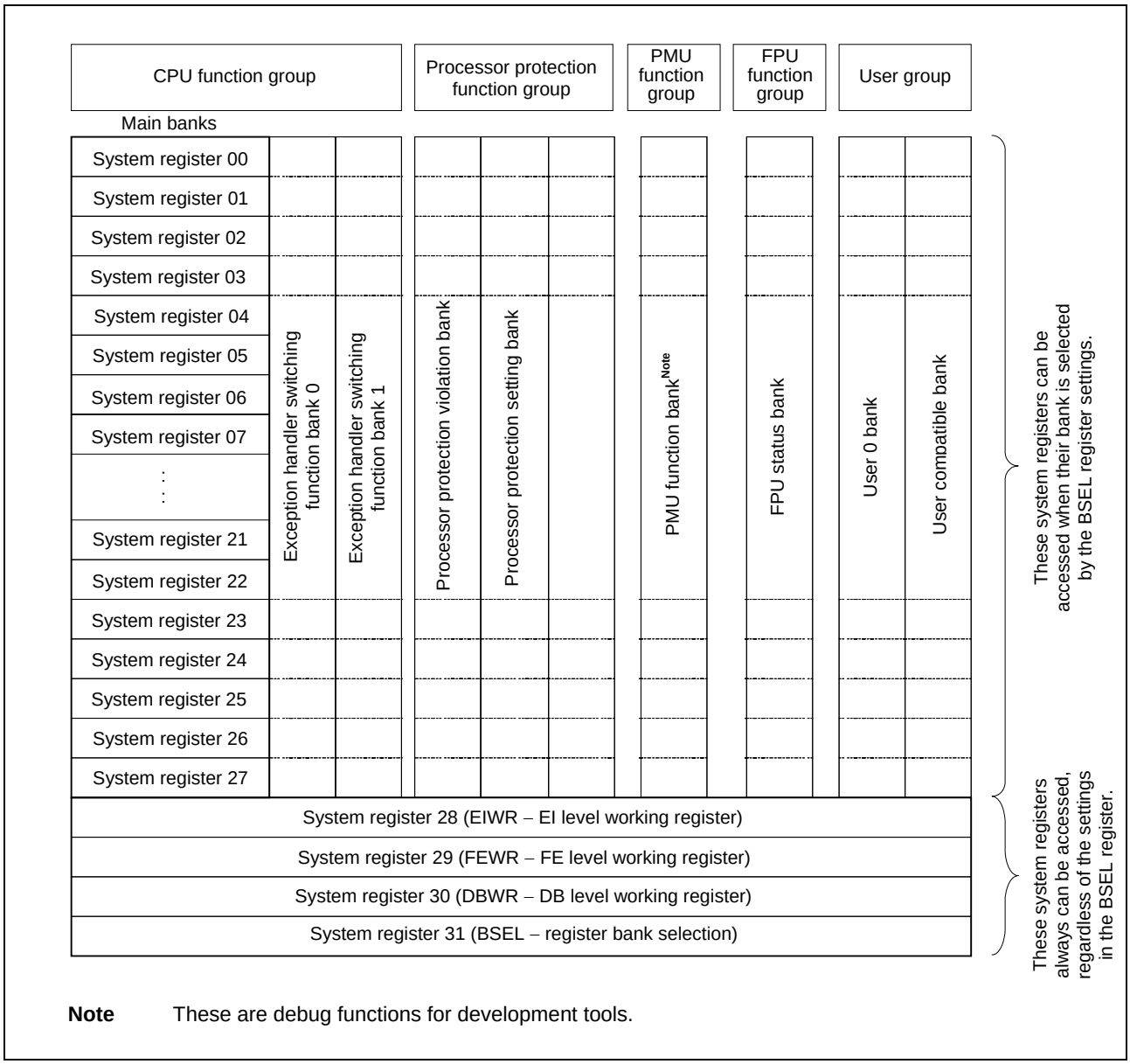
The V850E2M CPU system registers are provided in the system register bank. These system registers are defined in groups based on functions, and within these groups “banks” are defined for more specific applications. Up to 28 system registers can be defined (as registers 0 to 27) within each bank.

The V850E2M CPU includes the following groups and banks.

- CPU function group
  - Main bank: Conventional system registers
  - Exception handler switching function bank 0: System registers that switches exception handler addresses
  - Exception handler switching function bank 1: System registers that switches exception handler addresses
- Processor protection function group
  - Processor protection violation bank: System registers related to processor protection violations
  - Processor protection setting bank: System registers related to processor protection functions
  - Software paging bank: System registers that are used when the memory protection function is used for software paging operation
- PMU function group
  - PMU function bank: System registers that set performance measurement function[Note]
- FPU function group
  - FPU status bank: System registers related to floating-point operations
- User group
  - User 0 bank: This bank is able to access only system registers used by user applications.
  - User compatible bank: For the sake of compatibility, this bank is able to access exception-related system registers as well as system registers used by user applications

Note The PMU function bank is a debug function for development tools.

Figure 2-3. System Register Bank

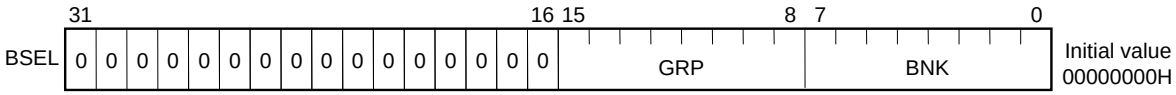


Note These are debug functions for development tools.

2.2.1 BSEL – Register bank selection

The BSEL register is used to select the system registers to be accessed by the LDSR instruction and STSR instruction. The BSEL register can also be referenced, regardless of which bank has been selected.

Always set “0” to bits 31 to 16.



Bit position	Bit name	Description
15 to 8	GRP	Specifies the system register bank group number (initial value: 0). 00H: CPU function group 10H: Processor protection function group 11H: PMU function group[Note] 20H: FPU function group FFH: User group Settings other than the above are prohibited.
7 to 0	BNK	Specifies bank number of system register bank (initial value: 0).

GRP	BNK	Corresponding execution level
00H	00H	Main bank
	10H	Exception handler switching function bank 0
	11H	Exception handler switching function bank 1
10H	00H	Processor protection violation bank
	01H	Processor protection setting bank
	10H	Software paging bank
11H	00H	PMU function bank[Note]
20H	00H	FPU status bank
FFH	00H	User 0 bank
	FFH	User compatible bank
Other than above		Setting prohibited

Note The PMU function bank is a debug function for development tools.

Part 2 described the CPU function group system register and user banks. For description of the system registers of the processor protection function group, see **PART 3 PROCESSOR PROTECTION FUNCTION.** For description of the system registers of the FPU function group, see **PART 4 FLOATING-POINT OPERATION FUNCTION**.

2.3 CPU Function Group/Main Bank

The system registers in the main bank are used to control CPU status and to retain exception information.

System register read and write operations are performed using the LDSR instruction and STSR instruction, as specified via the following system register numbers.

Table 2-2. System Register List (Main Bank)

System Register No.	Symbol	System Register Name	Able to Specify Operands?		System Register Protection
			LDSR Instruction	STSR Instruction	
0	EIPC	EI level exception status save register	√	√	√
1	EIPSW	EI level exception status save register	√	√	√
2	FEPC	FE level exception status save register	√	√	√
3	FEPSW	FE level exception status save register	√	√	√
4	ECR	Exception cause	×	√	√
5	PSW	Program status word	√	√	√ Note1
6 to 10		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
11	SCCFG	SYSCAL operation setting	√	√	√
12	SCBP	SYSCALL base pointer	√	√	√
13	EIIC	EI level exception cause	√	√	√
14	FEIC	FE level exception cause	√	√	√
15	DBIC Note2	DB level exception cause	–	–	–
16	CTPC	CALLT execution status save register	√	√	√
17	CTPSW	CALLT execution status save register	√	√	√
18	DBPC Note2	DB level exception status save register	–	–	–
19	DBPSW Note2	DB level exception status save register	–	–	–
20	CTBP	CALLT base pointer	√	√	×
21	DIR	Debug interface register	–	–	–
22 to 27		Debug function register	–	–	–
28	EIWR	EI level exception working register	√	√	√
29	FEWR	FE level exception working register	√	√	√
30	DBWR Note2	DB level exception working register	√	√	√
31	BSEL	Register bank selection	√	√	√

Notes 1. Only bits 31 to 6 are protected. Even if a write access is made while these bits are protected, a system register protection violation is not detected. For details, refer to **CHAPTER 5 SYSTEM REGISTER PROTECTION** in **PART 3**.

2. These are debug functions for development tools.

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

2.3.1 EIPC and EIPSW – Status save registers when acknowledging EI level exception

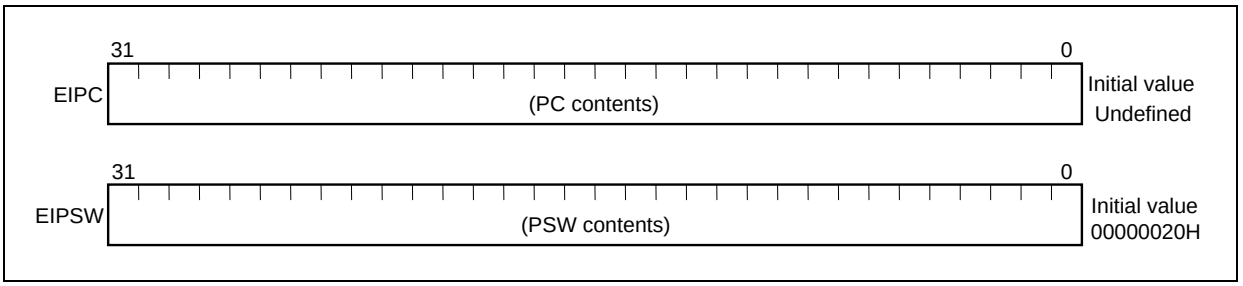
The EI level exception status save registers include EIPC and EIPSW.

When an EI level exception (EI level software exception, EI level interrupt (INT), etc.) has occurred, 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 **Table 6-1 Exception Cause List**). The current PSW information is saved to the EIPSW register.

Since 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.

If PSW bits are specified to be set to 0, the same bits in the EIPSW register must also be set to 0.



Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of EIPC is automatically set to bits 31 to 29.

2.3.2 FEPC and FEPSW – Status save registers when acknowledging FE level exception

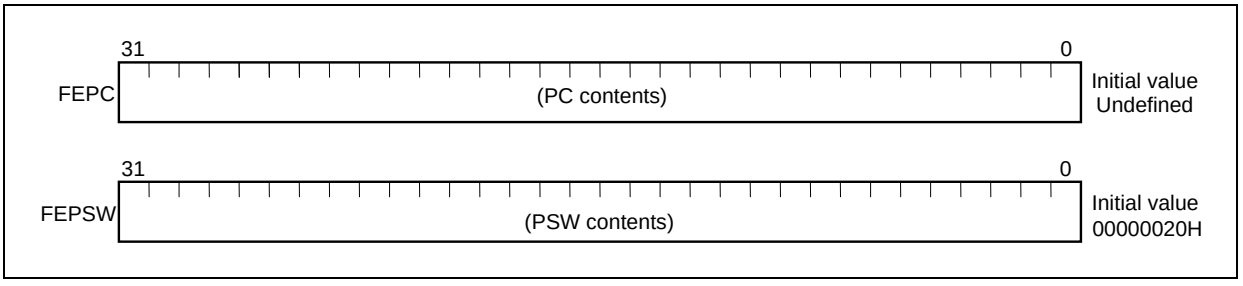
The FE level exception status save register include FEPC and FEPSW.

When an FE level exception (FE level software exception, FE level interrupt (FEINT or FENMI), etc.) has occurred, 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 **Table 6-1 Exception Cause List**). The current PSW information is saved to the FEPSW register.

Since 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.

If PSW bits are specified to be set to 0, the same bits in the FEPSW register must also be set to 0.

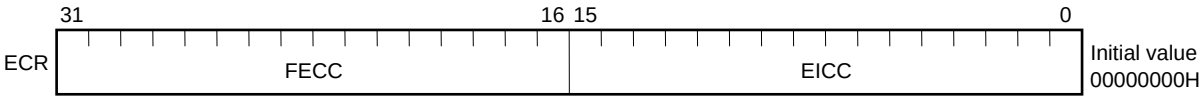


Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of FEPC is automatically set to bits 31 to 29.

2.3.3 ECR – Exception cause

When an exception has occurred, the ECR register retains the cause of the exception. These values retained in the ECR are exception codes corresponding to individual exception causes (see **Table 6-1 Exception Cause List**). Since this is a read-only register, the LDSR instruction cannot be used to write data to this register.

Caution The ECR register is for upward compatibility and is prohibited from being used in principle. For programs other than existing programs that do not enable modification, use a program that uses either the EIIC register or the FEIC register to overwrite all parts that were using the ECR register.



Bit position	Bit name	Description
31 to 16	FECC	Exception code for FE level exception (initial value: 0)
15 to 0	EICC	Exception code for EI level exception (initial value: 0)

2.3.4 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.)).

When the LDSR instruction is used to change the contents of various bits in a register, the changed contents become valid once execution of the LDSR instruction is completed.

Bits 31 to 6 are subject to system register protection. When system register protection is enabled, the contents of bits 31 to 6 cannot be changed by using the LDSR instruction (see **CHAPTER 5 SYSTEM REGISTER PROTECTION** in **PART 3**).

Bits 31 to 20, 15 to 12, and 8 are reserved for future function expansion, and be sure to set them to 0. Values are undefined when read[Note].

Note Use of bits 19 to 16 is described in **PART 3 PROCESSOR PROTECTION FUNCTIONS** (see **PART 3 PROCESSOR PROTECTION FUNCTION**).

Bit 11 to 9 are reserved for debug function for development tools. The LDSR instruction cannot be used to change these bits in user program.

(1/3)

	31										20	19	18	17	16	15			12	11	10	9	8	7	6	5	4	3	2	1	0	
PSW	0	0	0	0	0	0	0	0	0	0	0	PP	NPV	DMP	IMP	0	0	0	0	SS	SB	SE	0	NP	EP	ID	SAT	CY	OV	S	Z	Initial value 00000020H

Bit position	Bit or flag name	Description
19	PP	This bit indicates a state of peripheral device protection. It indicates whether the CPU trusts an access to a peripheral device by the program currently being executed. 0: T state (CPU trusts an access to the peripheral device.) (initial value) 1: NT state (CPU does not trust an access to the peripheral device.) The peripheral device protection function limits accesses when the PP bit indicates the T state. When the PP bit indicates the NT state, it strictly limits accesses.
18	NPV	This bit indicates a state of system register protection. It indicates whether the CPU trusts an access to a system register by the program currently being executed. 0: T state (CPU trusts an access to the system register.) (initial value) 1: NT state (CPU does not trust an access to the system register.) The system register protection function does not limit accesses when the NPV bit indicates the T state. When the NPV bit indicates the NT state, it limits accesses.

(2/3)

Bit position	Flag name	Description
17	DMP	This bit indicates a state of memory protection against a data access (to a data area). It indicates whether the CPU trusts a data access by the program currently being executed. 0: T state (CPU trusts the data access.) (initial value) 1: NT state (CPU does not trust the data access.) The memory protection function does not limit data accesses when the DMP bit indicates the T state. When the DMP bit indicates the NT state, it limits data accesses.
16	IMP	This bit indicates a state of memory protection in a program area. It indicates whether the CPU trusts an access to the program area by the program currently being executed. 0: T state (CPU trusts the access to the program area.) (initial value) 1: NT status (CPU does not trust the access to the program area.) The memory protection function does not limit accesses to the program area when the IMP bit indicates the T state. When the IMP bit indicates the NT state, it limits accesses to the program area
11	SS	Debug function for development tools.
10	SB	Debug function for development tools.
9	SE	Debug function for development tools.
7	NP	This bit indicates when FE level exception processing is in progress. When an FE level exception is acknowledged, this bit is set (1), which prohibits occurrence of multiple exceptions . 0: FE level exception processing is not in progress. (initial value) 1: FE level exception processing is in progress.
6	EP	This bit indicates that an exception other than an interrupt[Note] is being processed. It is set (1) when the corresponding exception occurs. This bit does not affect acknowledging an exception request even when it is set (1). 0: An interrupt is being processed (initial value). 1: An exception other than an interrupt is being processed.
5	ID	This bit indicates that an EI-level exception is being processed. It is set (1) when an EI level exception is acknowledged, disabling generation of multiple exceptions. 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 processed. It is set (1) when the DI instruction is executed, and cleared (0) when the EI instruction is executed. 0: EI level exception is being processed or the section is not a critical section (after execution of EI instruction). 1: EI level exception is being processed or the section is a critical section (after execution of DI instruction). (initial value)

Note For details of interrupts, see **6.1.2 Types of exceptions**.

(3/3)

Bit position	Flag name	Description
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 (1), but it is not later cleared to 0 when the operation result for a subsequent instruction is not saturated. This bit is cleared (0) by the LDSR instruction. This bit is neither set (1) nor cleared (0) when an arithmetic operation instruction is executed. 0: Not saturated (initial value) 1: Saturated
3	CY	This bit indicates whether a carry or borrow has occurred in the operation result. 0: Carry and borrow have not occurred (initial value). 1: Carry or borrow has occurred.
2	OV[Note]	This bit indicates whether or not an overflow has occurred during an operation. 0: Overflow has not occurred (initial value). 1: Overflow has occurred.
1	S[Note]	This bit indicates whether or not the result of an operation is negative. 0: Result of operation is positive or 0 (initial value). 1: Result of operation is negative.
0	Z	This bit indicates whether or not the result of an operation is 0. 0: Result of operation is not 0 (initial value). 1: Result of operation is 0.

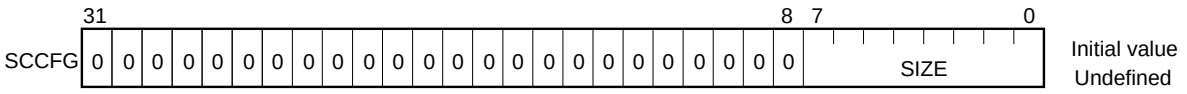
Note The operation result of the saturation processing is determined in accordance with the contents of the OV flag and S flag during a saturated operation. When only the OV flag is set (1) during a saturated operation, the SAT flag is set (1).

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	

2.3.5 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. Be sure to set 0 to bits 31 to 8.

Caution Do not place the SYSCALL instruction immediately after the LDSR instruction that changes the contents of the SCCFG register.



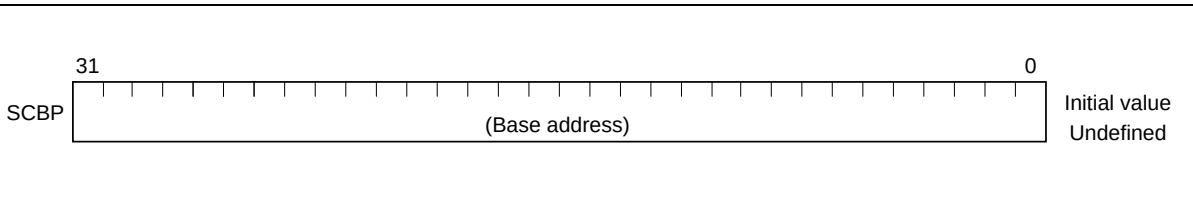
Bit position	Bit name	Description
7 to 0	SIZE	These bits specify the maximum number of entries of a table that the SYSCALL instruction references. The maximum number of entries the SYSCALL instruction references is 1 if SIZE is 0, and 256 if SIZE is 255. By setting the maximum number of entries appropriately 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.

2.3.6 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.

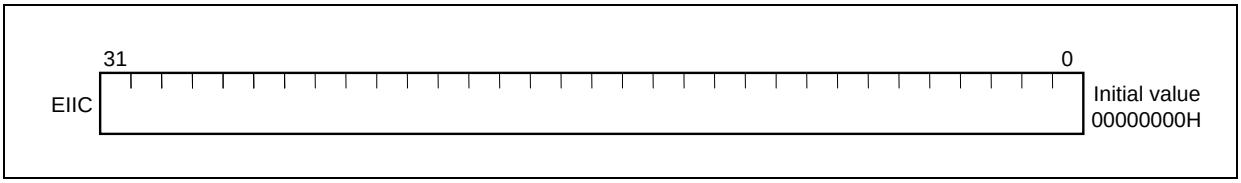
Note that bits 1 and 0 are fixed to 0.



Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of SCBP is automatically set to bits 31 to 29.

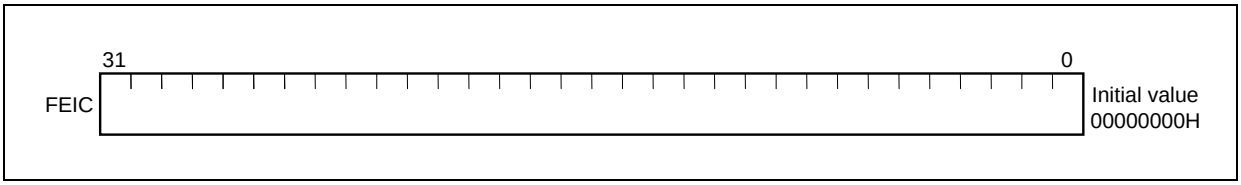
2.3.7 EIIC – EI level exception cause

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 6-1 Exception Cause List**).



2.3.8 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 6-1 Exception Cause List**).



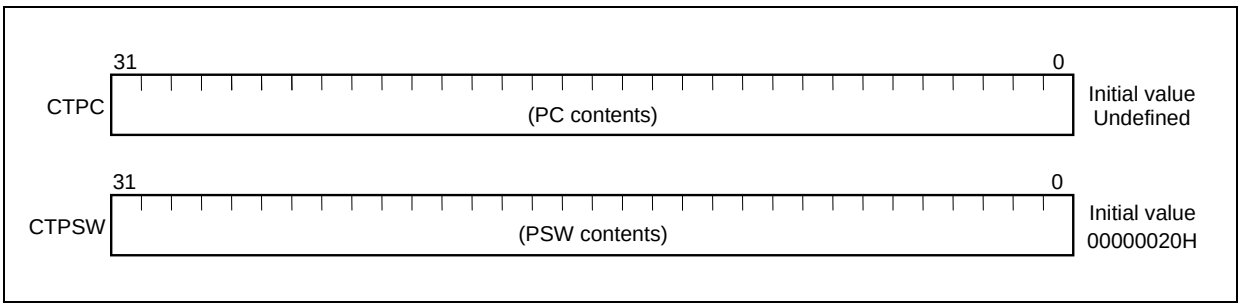
2.3.9 CTPC and CTPSW – Status save registers when executing CALLT

These are the status save registers when executing CALLT are CTPC and CTPSW.

When a CALLT instruction is executed, the address of the next instruction after the CALLT instruction is saved to CTPC and the contents of the PSW (program status word) is saved to CTPSW.

Be sure to set bit 0 of the CTPC register to 0.

Whenever a PSW bit is specified to be set to 0, the same bit in the CTPSW register must also be set to 0.



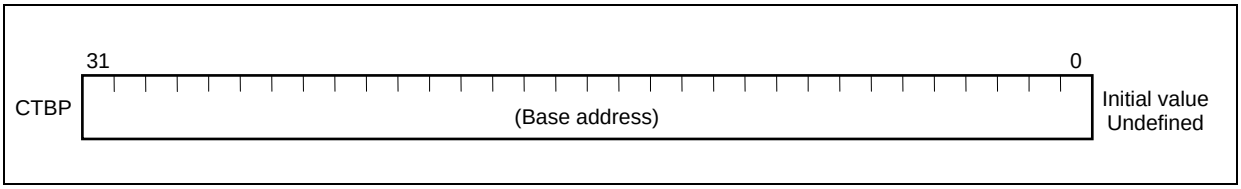
Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of CTPC is automatically set to bits 31 to 29.

2.3.10 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.

Note that bit 0 is fixed to 0.

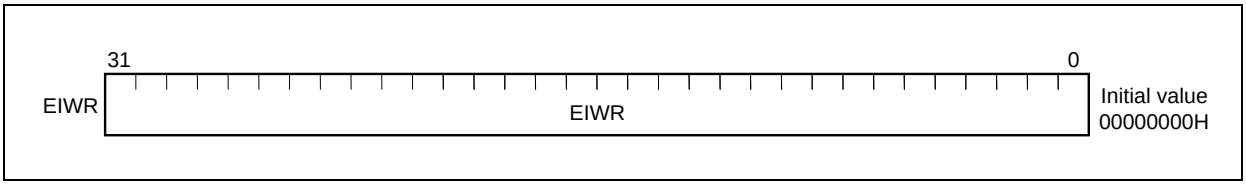


Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of CTBP is automatically set to bits 31 to 29.

2.3.11 EIWR – EI level exception working register

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

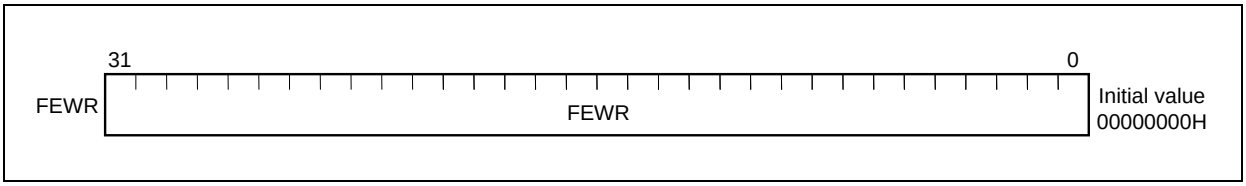
The EIWR register can always be referenced, regardless of which bank is selected.



2.3.12 FEWR – FE level exception working register

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

The FEWR register can always be referenced, regardless of which bank is selected.



2.3.13 DBIC – DB level exception cause

The DBIC register is related to debug function.

The DBIC register is a debug function for development tools.

2.3.14 DBPC and DBPSW – Status save registers when acknowledging DB level exception

The status save registers when acknowledging the DB level exception are DBPC and DBPSW.

The DBPC and DBPSW register are debug functions for development tools.

2.3.15 DBWR – DB level exception working register

The DBWR register is related to debug function.

The DBWR register is a debug a function for development tools.

2.3.16 DIR – Debug interface register

The DIR register controls and indicates the status of debug function.

The DIR register and other debug function-related registers (system registers 22 to 27) are debug functions for development tools.

2.4 CPU Function Group/Exception Handler Address Switching Function Banks

Exception handler switching function banks 0 and 1 are selected when 00000010H and 00000011H are set to the BSEL register by LDSR instructions (see **2.2.1 BSEL – Register bank selection**).

System registers 28 to 31 are system registers for all banks, and EIWR, FEWR, DBWR, and BSEL registers in the CPU function bank are referenced regardless of the settings in the BSEL register.

- Exception handler switching function bank 0
  (Group number 00H, bank number 10H, abbreviated as EHSW0 bank)
- Exception handler switching function bank 1
  (Group number 00H, bank number 11H, abbreviated as EHSW1 bank)

Table 2-3. System Register Bank

Group	CPU Function (00H)									
Bank	Exception Handler Switching Function Bank 0 (10H)					Exception Handler Switching Function Bank 1 (11H)				
Bank label	EHSW0					EHSW1				
Register No.	Name	Function	Able to Specify Operands?		System Register	Name	Function	Able to Specify Operands?		System Register
			LDSR Instruction	STSR Instruction				LDSR Instruction	STSR Instruction	
0	SW_CTL	Exception handler address switching control	√	√	√	Reserved for future function expansion		×	×	√
1	SW_CFG	Exception handler address switching configuration	√	√	√	EH_CFG	Exception handler configuration	×	√	√
2	Reserved for future function expansion		×	×	√	EH_RESE	Reset address register	×	√	√
3	SW_BASE	Exception handler address switching base address	√	√	√	EH_BASE	Exception handler base address	×	√	√
4 to 27	Reserved for future function expansion		×	×	√	Reserved for future function expansion		×	×	√
28	EIWR	EI level exception working register						√	√	√
29	FEWR	FE level exception working register						√	√	√
30	DBWR[Note]	DB level exception working register						√	√	√
31	BSEL	Register bank selection						√	√	√

Note The DBWR register is a debug function for development tools.

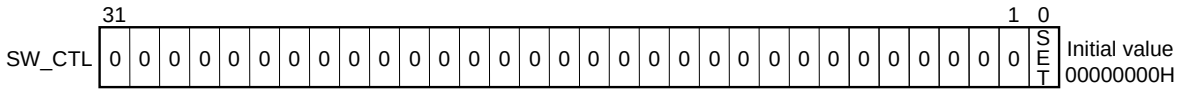
Remark √: Indicates in the column of “Able to specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

2.4.1 SW_CTL – Exception handler address switching control

This register controls the exception handler address switching functions.

Be sure to set bits 31 to 1 to “0”.

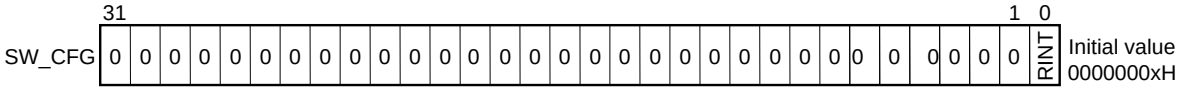


Bit position	Bit name	Description
0	SET	When the SET bit is set (1), values in the SW_CFG and SW_BASE registers are transferred to EH_CFG and EH_BASE. After these transfers are completed, the SET bit is cleared (0).

2.4.2 SW_CFG – Exception handler address switching configuration

This register specifies settings for the exception handler address switching function.

Be sure to set bits 31 to 1 to “0”.

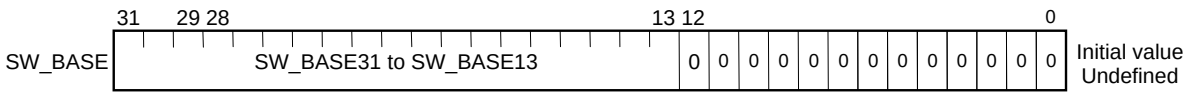


Bit position	Bit name	Description
0	RINT	When the SW_CTL.SET bit is set (1), values in the SW_CFG register are transferred to the EH_CFG register.

2.4.3 SW_BASE – Exception handler address switching base address

This register specifies the base address for the exception handler addresses used by the exception handler address switching function.

Be sure to set bits 12 to 0 to “0”.



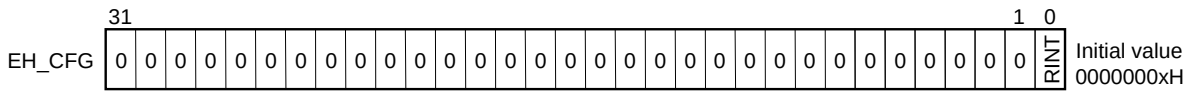
Bit position	Bit name	Description
31 to 13	SW_BASE31 to SW_BASE13	When the SW_CTL.SET bit is set (1), the contents of the SW_BASE register are transferred to the EH_BASE register.

Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of SW_BASE is automatically set to bits 31 to 29.

2.4.4 EH_CFG – Exception handler configuration

This register indicates the current settings of the exception handler address switching function.

Bits 31 to 1 are fixed to 0.



Bit position	Bit name	Description
0	RINT	When the RINT bit is set (1), exception handler addresses INT0 to INT255 are reduced to a single handler address (INT0). When this bit is cleared (0), INT0 to INT255 are held as independent exception handler addresses. The EH_CFG register is set to the initial value defined for each product at reset. This register cannot be overwritten directly by an LDSR instruction. When the SW_CTL.SET bit is set (1), the contents of SW_CFG are transferred.

2.4.5 EH_BASE – Exception handler base address

This register indicates the base addresses of the current exception handler address for the exception handler address switching function.

Bits 12 to 0 are fixed to 0.

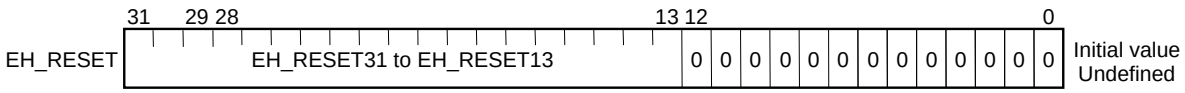
Bit position	Bit name	Description
31 to 13	EH_BASE31 to EH_BASE13	Addresses are changed by adding an offset address for each exception to the base addresses of exception handler routines specified in this register. The EH_BASE register is set to the initial value defined for each product at reset. This register cannot be overwritten directly by an LDSR instruction. When the SW_CTL.SET bit is set (1), the contents of SW_BASE are transferred.

Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of EH_BASE is automatically set to bits 31 to 29.

2.4.6 EH_RESET – Reset address

This register indicates the reset address when the current reset is input.

Bits 12 to 0 are fixed to 0.



Bit position	Bit name	Description
31 to 13	EH_RESET31 to EH_RESET13	The values in the EH_RESET register do not change except when initialized by a reset. For details, see the User's Manual of each product or CPU.

Caution For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 of EH_RESET is automatically set to bits 31 to 29.

2.5 User Group

The user group is selected when 0000FF00H is set by an LDSR instruction to the BSEL register (see **2.2.1 BSEL – Register bank selection**). System registers in the user group are maps of the registers in the main bank and FPU status bank. The user group includes the following two banks.

- User 0 bank (see **Table 2-4**)
- User compatible bank (see **Table 2-5**)

Caution The user compatible bank is defined for backward compatibility with the V850E1 and V850E2 architectures, and its use is generally prohibited. Use the user 0 bank unless manipulating a system register for which no uncorrectable program exists in the bank.

Table 2-4. System Register List (User 0 Bank)

System Register No.	Symbol	Function	Able to Specify Operands?		System Register Protection
			LDSR	STSR	
0 to 4		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
5	PSW	Program status word	√	√	√Note1
6, 7		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
8	FPST	Floating-point operation status	√	√	×
9	FPCC	Floating-point operation comparison result	√	√	×
10	FPCFG	Floating-point function configuration	√	√	×
11 to 15		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
16	CTPC	Status save register when executing CALLT	√	√	√
17	CTPSW	Status save register when executing CALLT	√	√	√
18, 19		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
20	CTBP	CALLT base pointer	√	√	×
21 to 27		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
28	EIWR	EI level exception working register	√	√	√
29	FEWR	FE level exception working register	√	√	√
30	DBWR Note2	DB level exception working register	√	√	√
31	BSEL	Register bank selection	√	√	√

Notes1. Only bits 31 to 6 are protected.

2. The DBWR register is a debug function for development tools.

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

Table 2-5. System Register List (User Compatible Bank)

System Register No.	Symbol	Function	Able to Specify Operands?		System Register Protection
			LDSR	STSR	
0	EIPC	Status save register when acknowledging EI level exception	√	√	√
1	EIPSW	Status save register when acknowledging EI level exception	√	√	√
2	FEPC	Status save register when acknowledging FE level exception	√	√	√
3	FEPSW	Status save register when acknowledging FE level exception	√	√	√
4	ECR	Exception cause	×	√	√
5	PSW	Program status word	√	√	√Note1
6, 7		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
8	FPST	Floating-point operation status	√	√	×
9	FPCC	Floating-point operation comparison result	√	√	×
10	FPCFG	Floating-point function configuration	√	√	×
11, 12		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
13	EIIC	EI level exception cause	√	√	√
14	FEIC	FE level exception cause	√	√	√
15		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
16	CTPC	Status save register when executing CALLT	√	√	√
17	CTPSW	Status save register when executing CALLT	√	√	√
18, 19		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
20	CTBP	CALLT base pointer	√	√	×
21 to 27		(Reserved for future function expansion (operation is not guaranteed when accessed))	×	×	√
28	EIWR	EI level exception working register	√	√	√
29	FEWR	FE level exception working register	√	√	√
30	DBWR Note2	DB level exception working register	√	√	√
31	BSEL	Register bank selection	√	√	√

Notes1. Only bits 31 to 6 are protected.

2. The DBWR register is a debug function for development tools.

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

CHAPTER 3 DATA TYPES

3.1 Data Formats

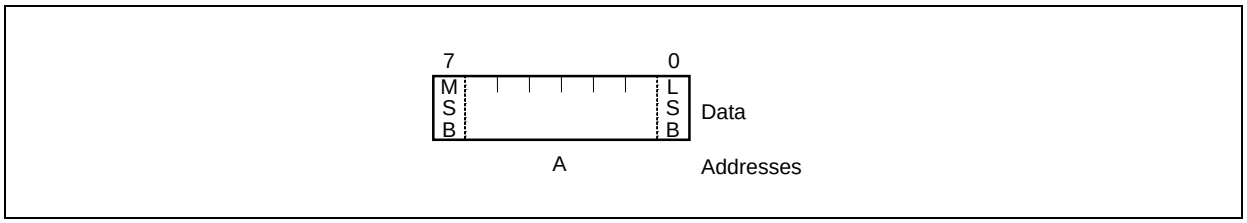
The V850E2M CPU handles data in little endian format. This means that byte 0 of a halfword or a word is always the least significant (rightmost) byte.

The supported data format is as follows.

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

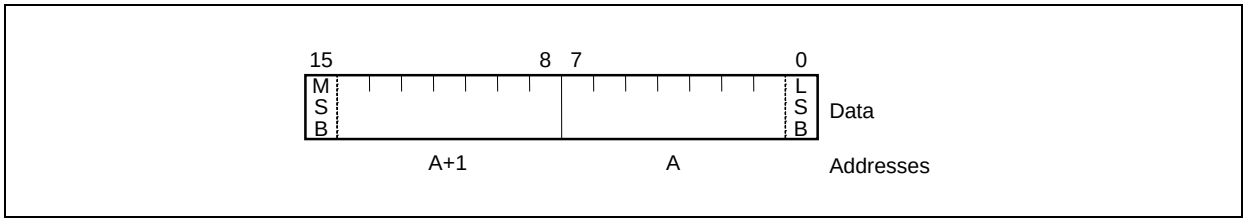
3.1.1 Byte

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”.



3.1.2 Halfword

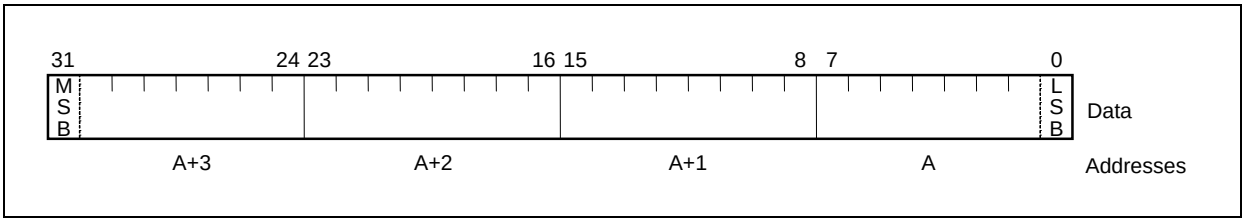
A halfword is two consecutive bytes (16 bits) of data that starts from any byte boundary[Note]. Numbers from 0 to 15 are assigned to these bits, with bit 0 as the LSB and bit 15 as the MSB. The bytes in a halfword are specified using address “A”, so that the two addresses comprise byte data of “A” and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



Note During word access, the V850E2M CPU can be accessed at all byte boundaries. See **3.3 Data Alignment**.

3.1.3 Word

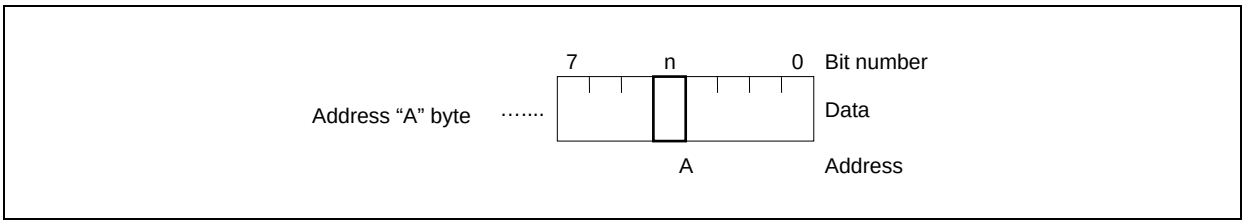
A word is four consecutive bytes (32 bits) of data that starts from any byte boundary[Note]. Numbers from 0 to 31 are assigned to these bits, with bit 0 as the LSB (least significant bit) and bit 31 as the MSB (most significant bit). A word is specified by address “A” and consists of byte data of four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



Note During word access, the V850E2M CPU can be accessed at all byte boundaries.
See **3.3 Data Alignment**.

3.1.4 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).



3.2 Data Representation

3.2.1 Integers

Integers are represented as binary values using 2's complement, and are used in one of three lengths: 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. Since 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.

- Word (32 bits): −2147483648 to +2147483647
- Halfword (16 bits): −32768 to +32767
- Byte (8 bits): −128 to +127

3.2.2 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 three lengths: 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.

- Word (32 bits): 0 to 4294967295
- Halfword (16 bits): 0 to 65535
- Byte (8 bits): 0 to 255

3.2.3 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

3.3 Data Alignment

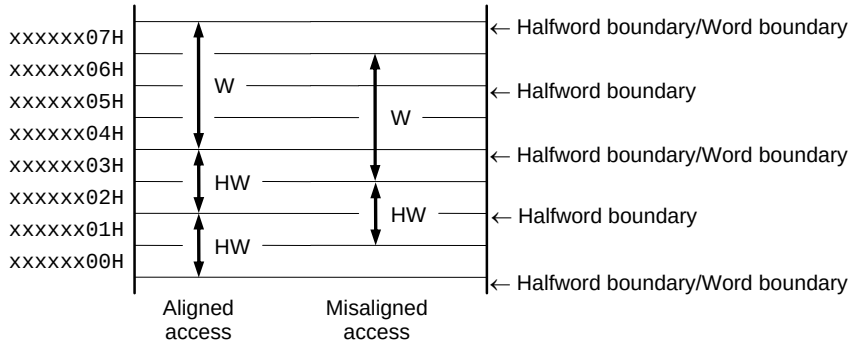
The V850E2M CPU allows misaligned placement of data.

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).

Regardless of the data format (byte, halfword, or word), data can be allocated at all addresses.

However, in the case of halfword data or word data, if the data is not aligned, at least one extra bus cycle will occur, which increases the execution time for the instruction.

Figure 3-1. Example of Data Placement for Misaligned Access



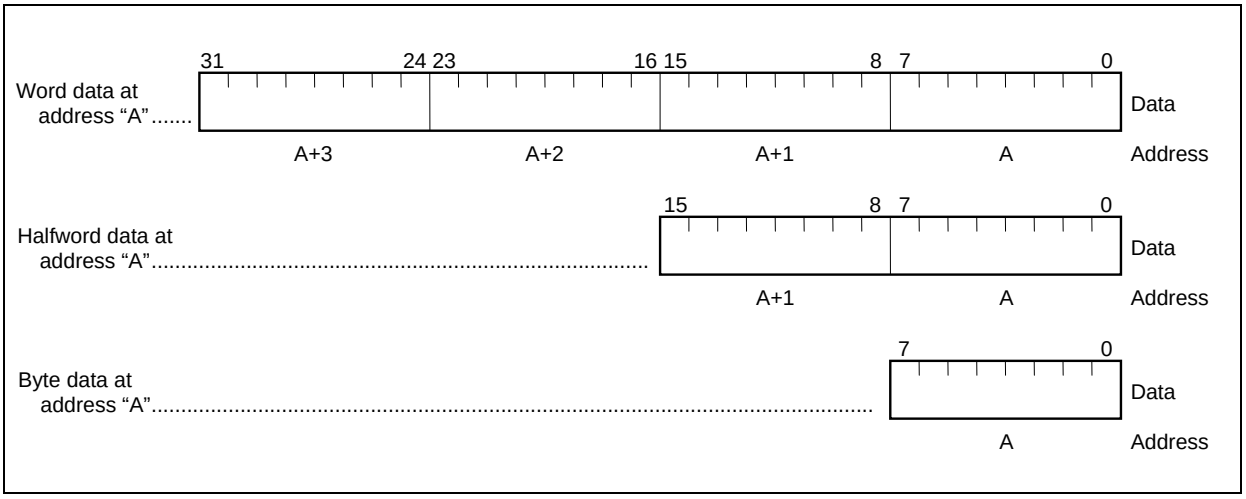
Remark W: word data
HW: halfword data

CHAPTER 4 ADDRESS SPACE

The V850E2M CPU supports a linear address space of up to 4 GB. Both memory and I/O are 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.



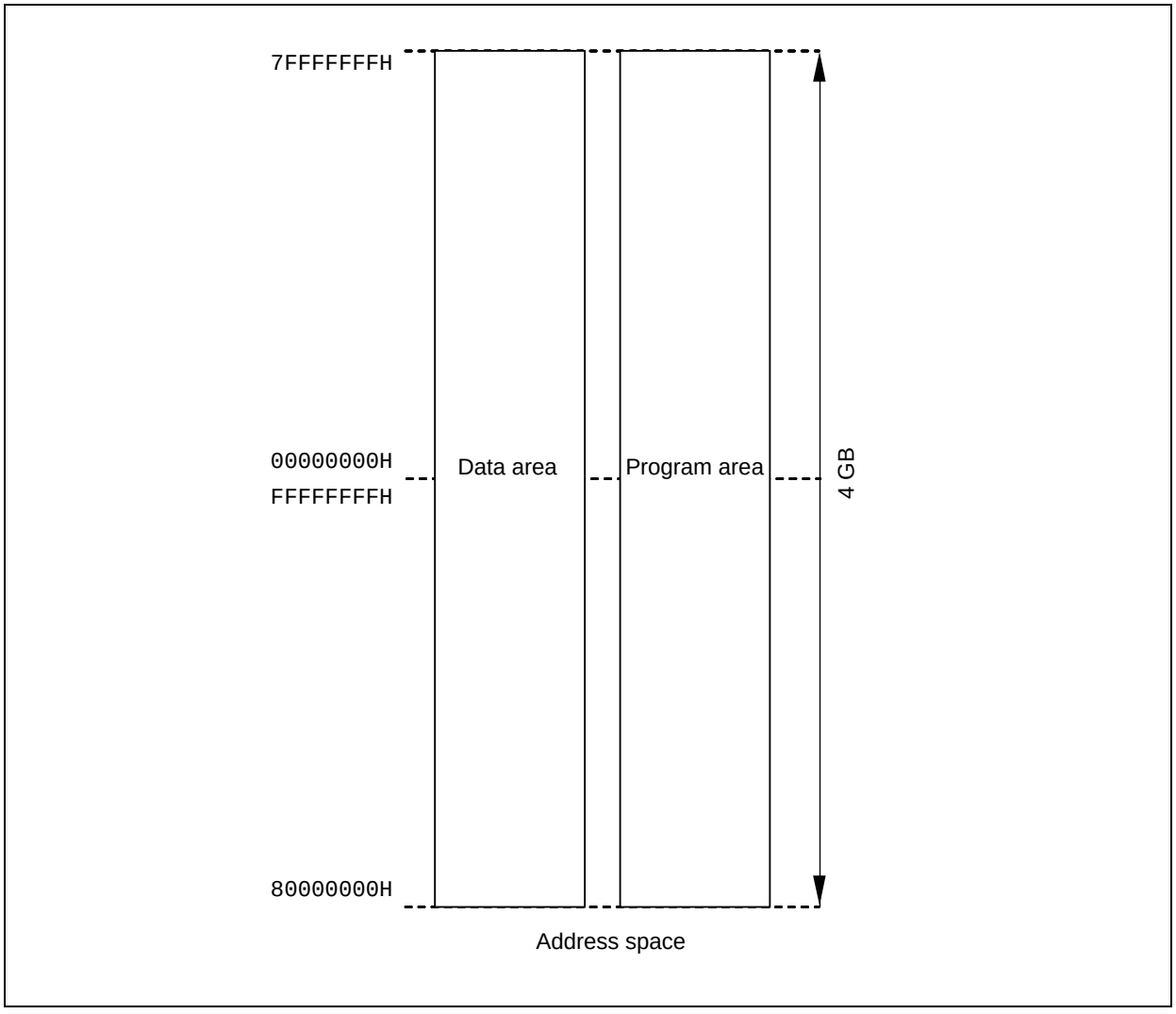
4.1 Memory Map

The V850E2M CPU is 32-bit architecture and supports a linear address space of up to 4 GB. The whole range of this 4 GB address space can be addressed by instruction addressing (instruction access) and operand addressing (data access).

Caution Instruction addressing is possible in a range of 64 MB with the V850E1 CPU and of 512 MB with the V850E2 CPU.

A memory map is shown in Figure 4-1.

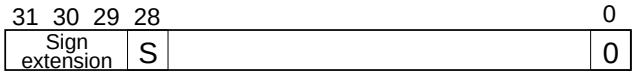
Figure 4-1. Memory Map (Address Space)



Caution For the V850E2M CPU, the range of the 4 GB program area that can be actually addressed is limited because of physical restrictions of registers that hold an instruction address (such as the program counter). The following registers hold an instruction address.

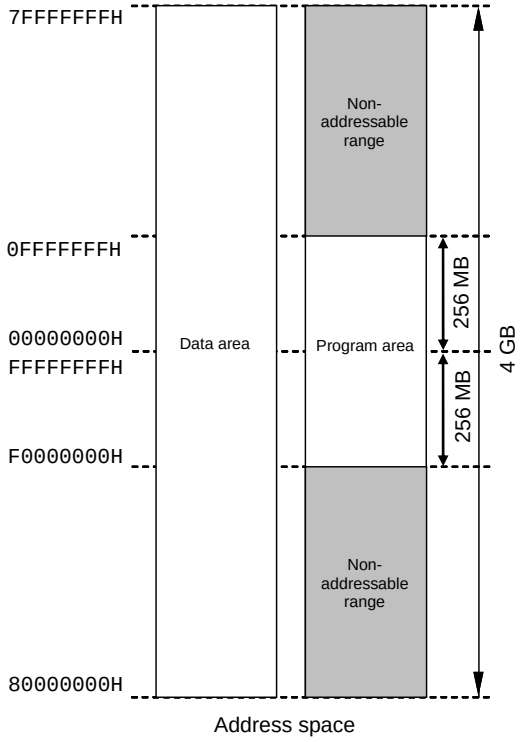
- **PC (program counter)**
- **EIPC and FEPC (exception context)**
- **SCBP, CTBP, and CTPC (table branch/exception instruction)**
- **SW_BASE, EH_BASE, and EH_RESET (exception handler selection function)**
- **FPEPC (floating-point operation function)**
- **VSADR (processor protection function)**

With a CPU whose addressable range of the program area is limited by the product specification to 512 MB, the higher 3 bits of these registers are automatically set to values resulting from a sign-extension of bit 28. Therefore, the addressable ranges are 00000000H to 0FFFFFFEH and F0000000H to FFFFFFFEH (the least significant bit is always 0).



(Instruction address register with limitation of 512 MB)

The memory map in this case is shown below.



(Memory map with 512 MB limitation)

When the memory map with a 512 MB limitation is used, be sure to place a table that is referenced by instructions and the SWITCH, CALLT, and SYSCALL instructions in a range that can be addressed by instruction addressing. The other data may be placed anywhere in the 4 GB space.

4.2 Addressing Modes

Two types of addresses are generated: instruction addresses that are used for instructions involved in branch operations, and operand addresses that are used for instructions that access data.

4.2.1 Instruction address

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.

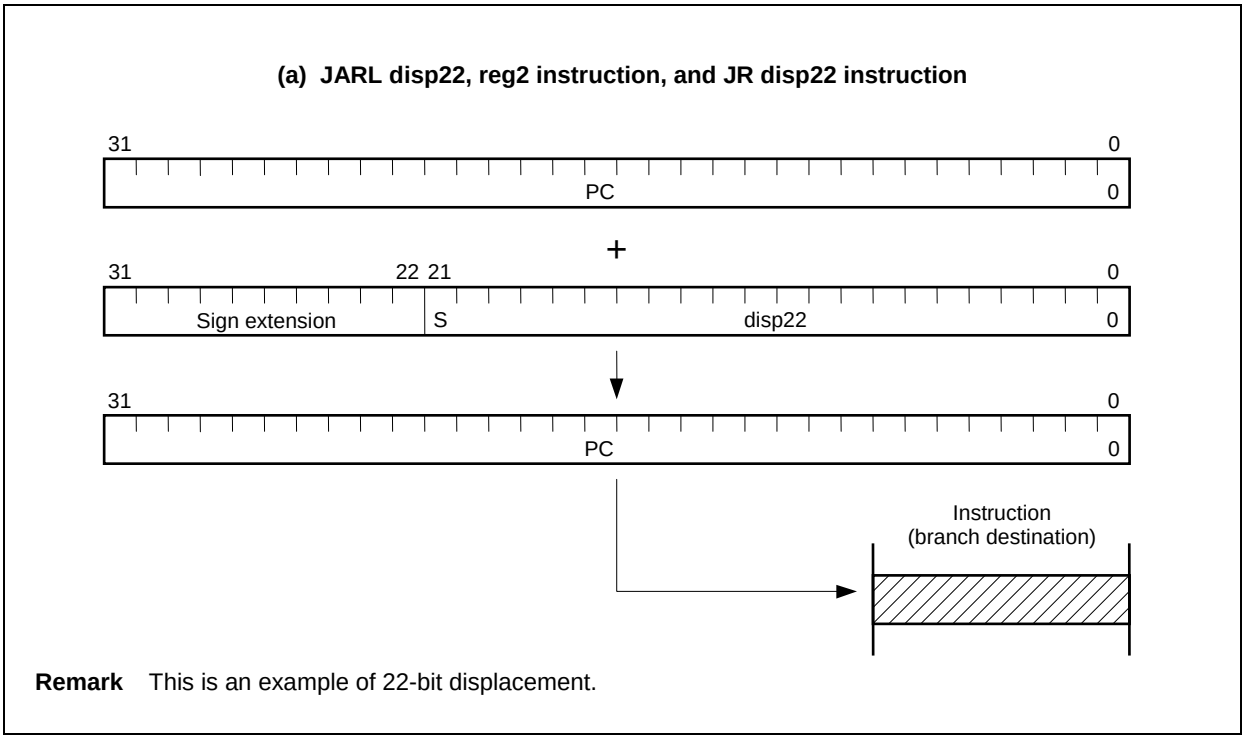
(1) Relative addressing (PC relative)

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 4-2. Relative Addressing



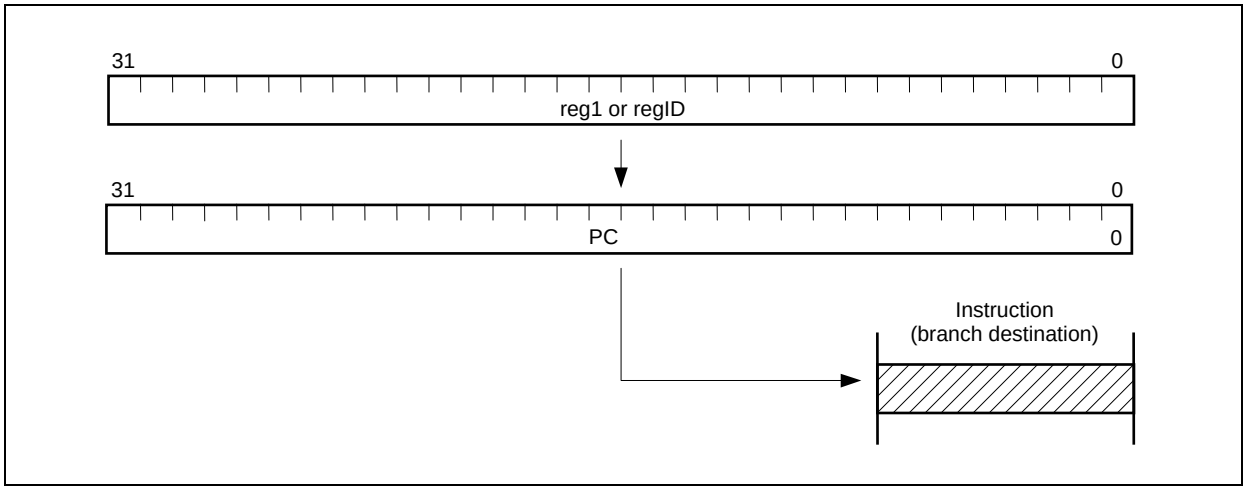
Remark This is an example of 22-bit displacement.

(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, RETI, and DISPOSE instructions are used with this type of addressing.

Figure 4-3. Register Addressing

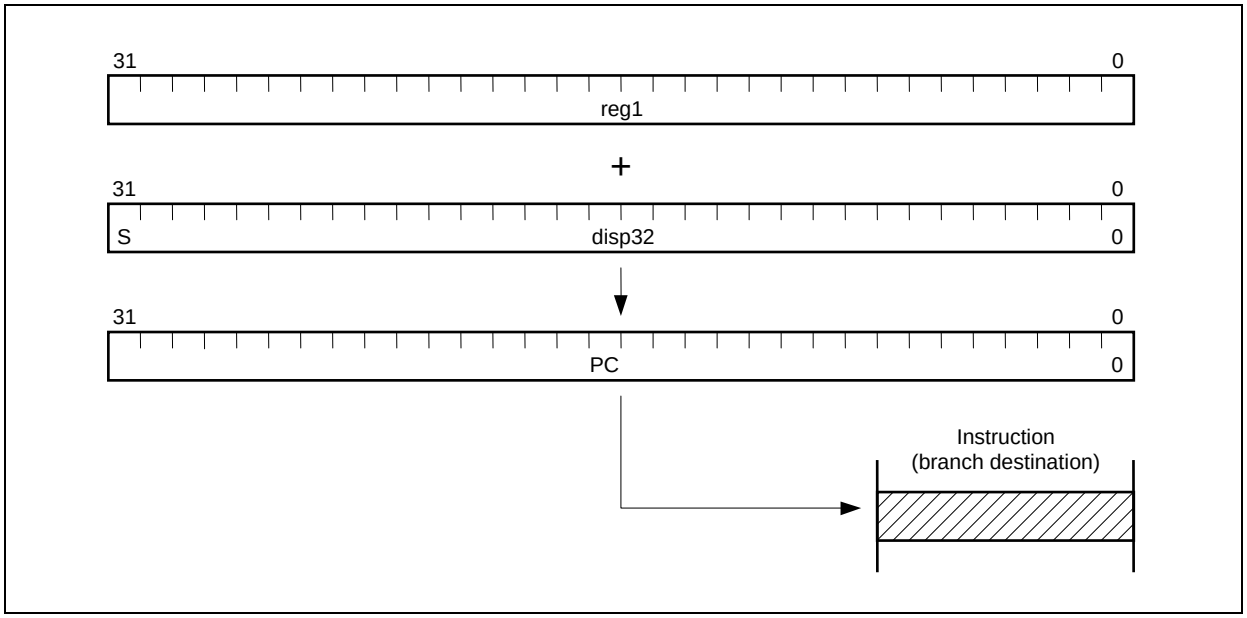


(3) Based addressing

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 4-4. 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.

4.2.2 Operand address

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 are 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 are 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

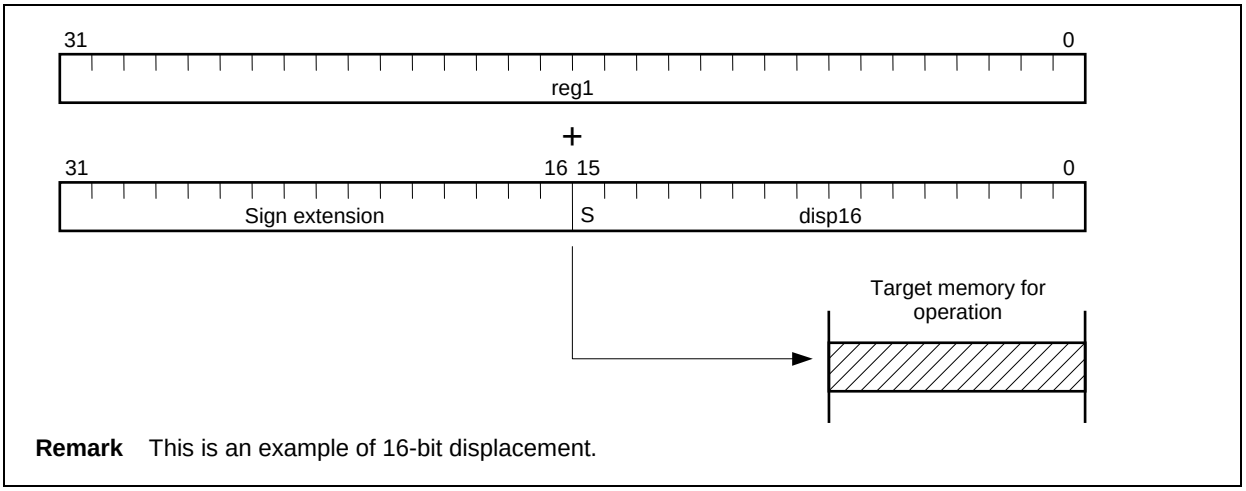
There are two types of based addressing, as described below.

(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 4-5. Based Addressing (Type 1)



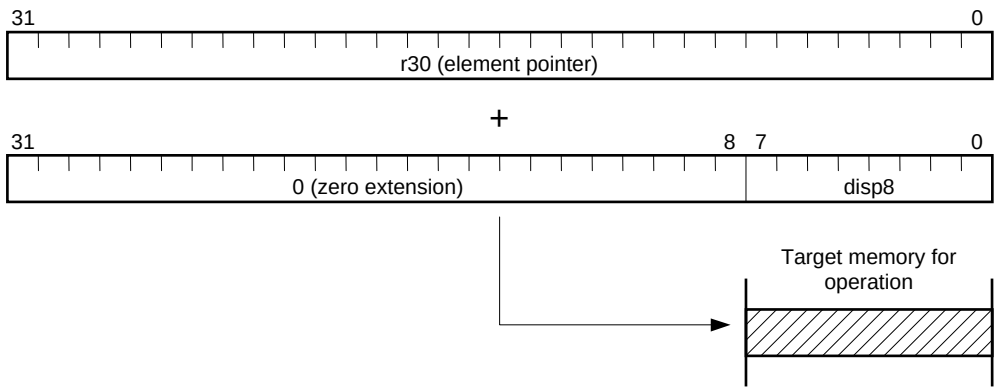
Remark This is an example of 16-bit displacement.

(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 4-6. Based Addressing (Type 2)



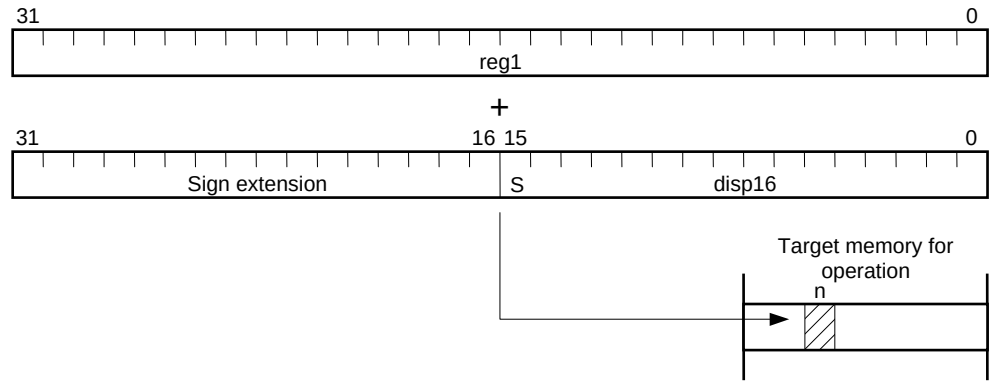
Remark This is an example of 8-bit displacement.

(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 4-7. Bit Addressing



Remark n: Bit position specified by 3-bit data (bit #3) (n = 0 to 7)
In case of 16-bit displacement

(5) 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, and DISPOSE instructions are used with this type of addressing.

CHAPTER 5 INSTRUCTIONS

5.1 Opcodes and Instruction Formats

The V850E2M CPU has two types of instructions: CPU instructions, which are defined as basic instructions, and coprocessor instructions, which are defined according to the application.

5.1.1 CPU instructions

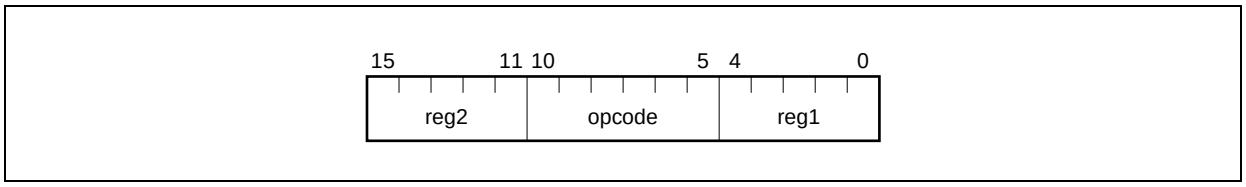
Instructions classified as CPU instructions are allocated in the opcode area other than the area used in the format of the coprocessor instructions shown in **5.1.2 Coprocessor instructions**.

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 **5.3 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 **5.1.3 Reserved instructions**.

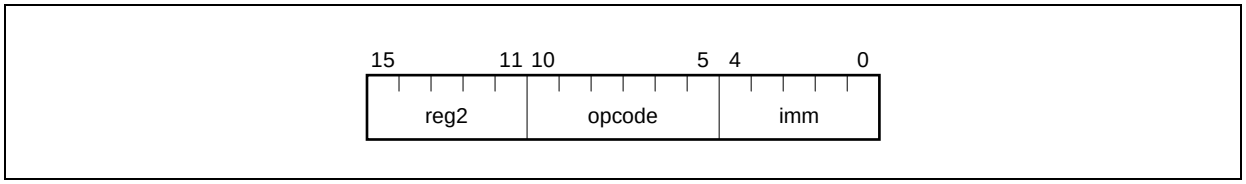
(1) reg-reg instruction (Format I)

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



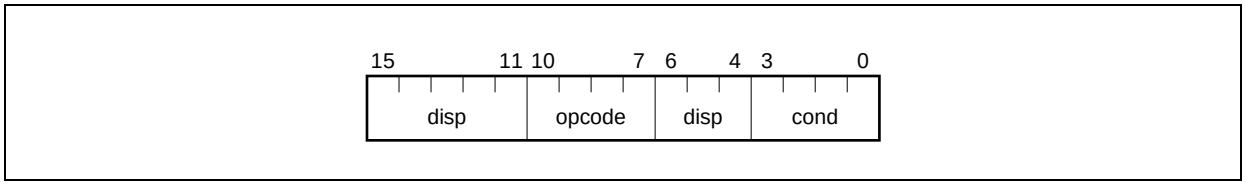
(2) imm-reg instruction (Format II)

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



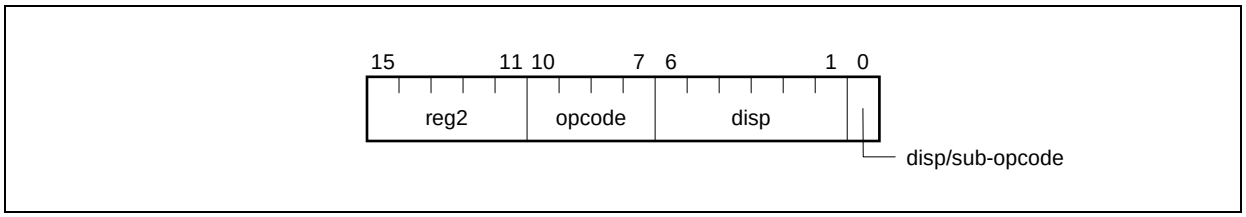
(3) Conditional branch instruction (Format III)

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

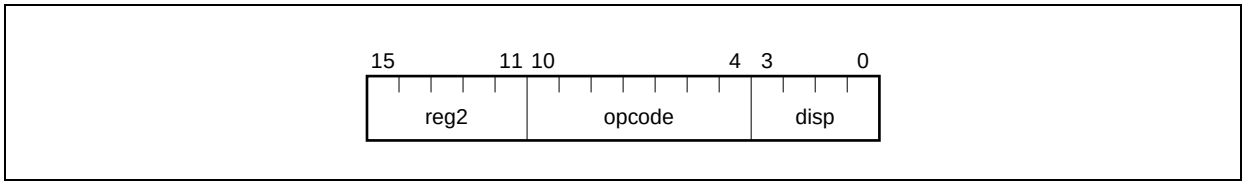


(4) 16-bit load/store instruction (Format IV)

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).

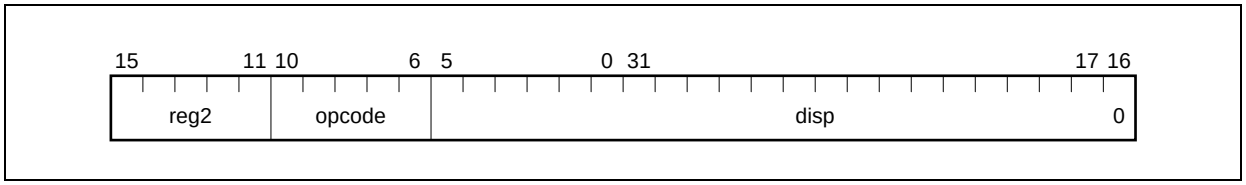


A 16-bit instruction format consists of a 7-bit opcode field, a general-purpose register specification field, and a 4-bit displacement field.



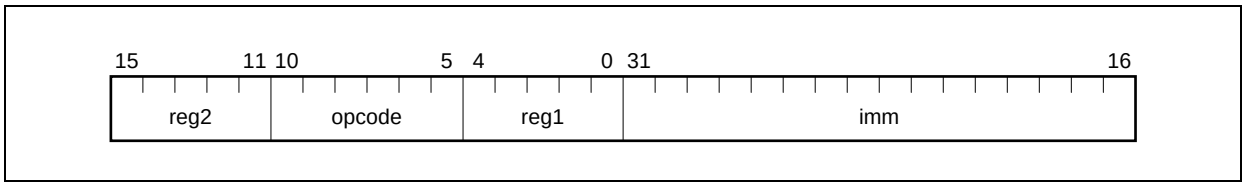
(5) Jump instruction (Format V)

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



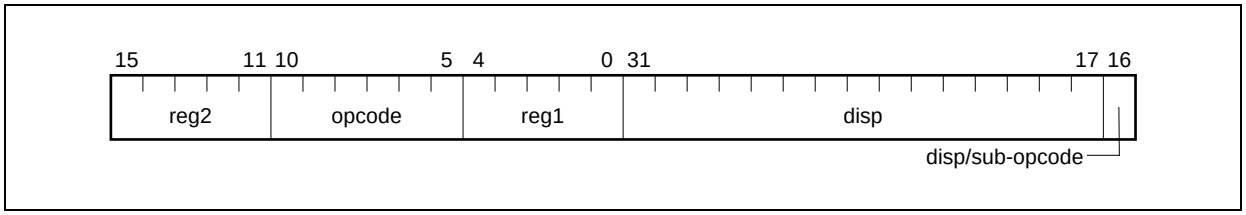
(6) 3-operand instruction (Format VI)

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



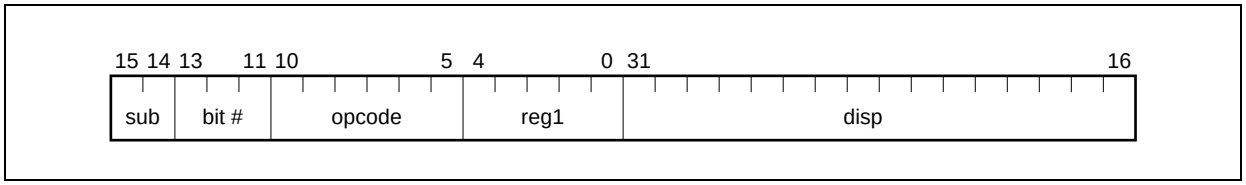
(7) 32-bit load/store instruction (Format VII)

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)

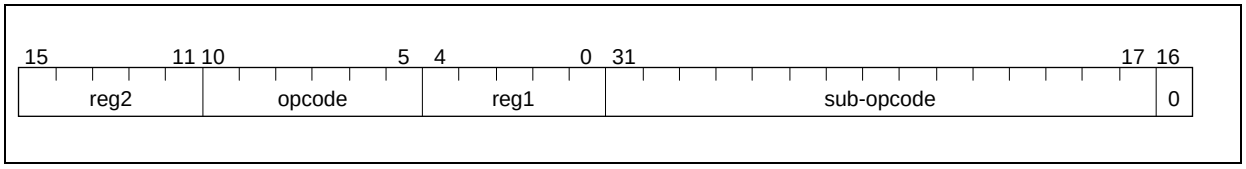
A 32-bit instruction format consists of a 6-bit opcode field, 2-bit sub-opcode field, 3-bit bit specification field, a general-purpose register specification field, and a 16-bit displacement field.



(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.

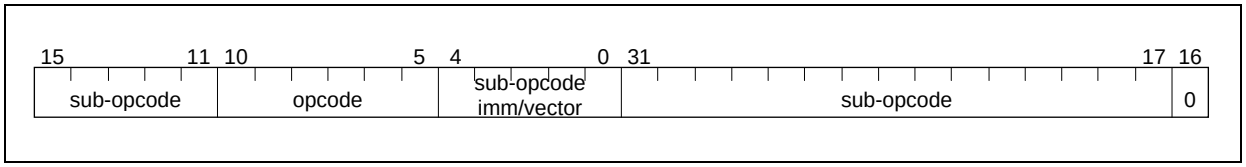
Caution Extended instruction format 1 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction 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.

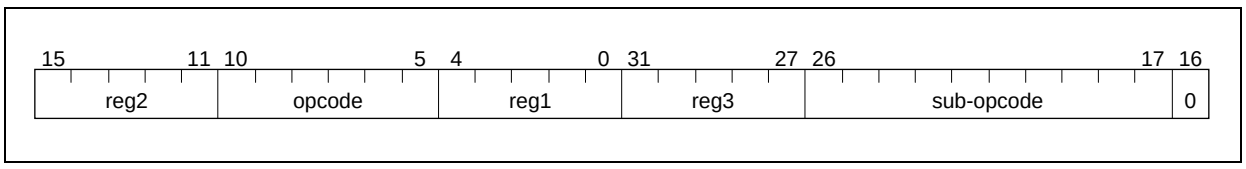
Caution Extended instruction format 2 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction 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.

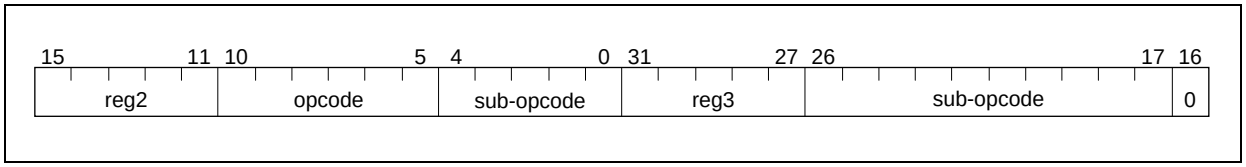
Caution Extended instruction format 3 may use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction 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.

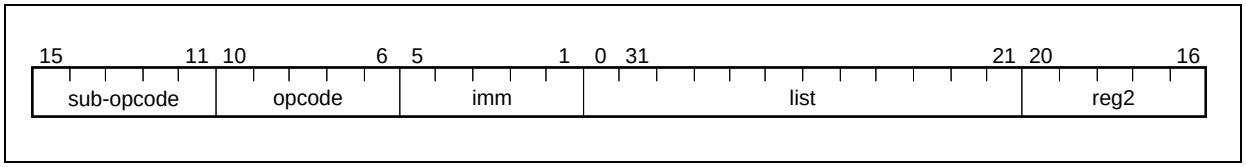
Caution Extended instruction format 4 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction Set.



(13) Stack manipulation instruction format (Format XIII)

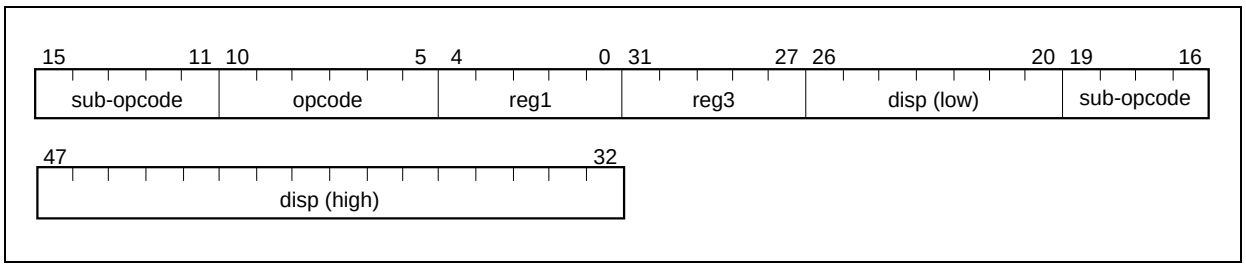
A 32-bit instruction format consists of a 5-bit opcode field, 5-bit immediate field, 12-bit register list field, 5-bit sub-opcode field, and one general-purpose register specification field (or 5-bit sub-opcode field).

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



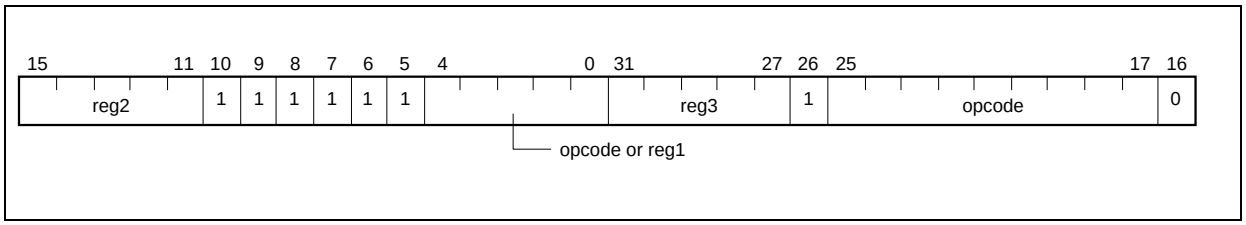
(14) Load/store instruction 48-bit format (Format XIV)

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.



5.1.2 Coprocessor instructions

Instructions in the following format are defined as coprocessor instructions.



Coprocessor instructions define the functions of each coprocessor.

In the V850E2M CPU, floating-point operation functions are defined as coprocessors.

For the instruction format of floating-point operation instructions, see **CHAPTER 4 INSTRUCTIONS** in **PART 4**.

(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 **CHAPTER 7 COPROCESSOR UNUSABLE STATUS**.

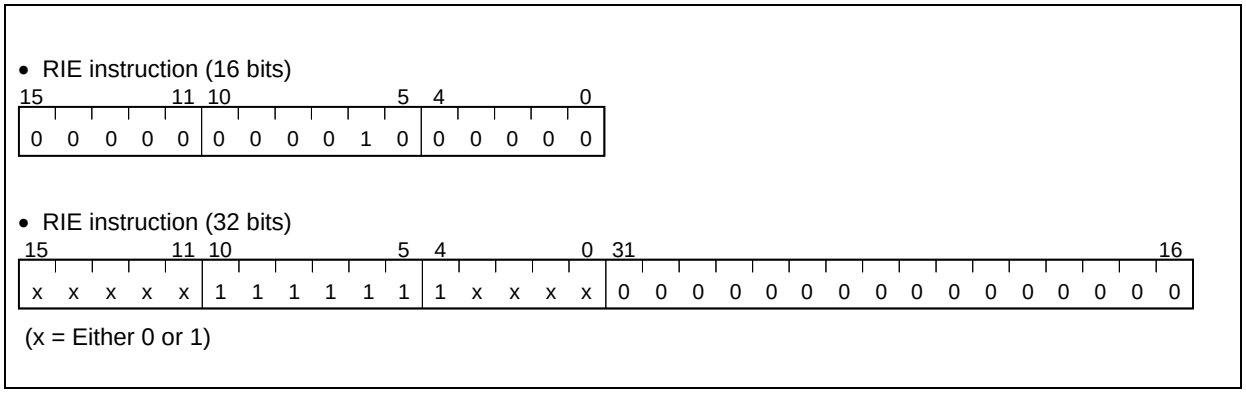
5.1.3 Reserved instructions

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 product specification 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 the V850E2M CPU, the following opcodes define the RIE instruction, which always causes a reserved instruction exception to occur.



5.2 Overview of 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.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.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 1 clock with 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 word and add
- MACU: Multiply word unsigned and add

(5) Arithmetic instructions:

Add, subtract, divide, 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:

- BSH: Byte swap halfword
- BSW: Byte swap word
- CMOV: Conditional move
- HSH: Halfword swap halfword
- HSW: Halfword swap word
- 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: Branch on condition code (BC, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BR, BSA, BV, BZ)
- JARL: Jump and register link
- JMP: Jump register
- JR: Jump relative

(14) 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

(15) 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
- 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
- NOP: No operation
- PREPARE: Function prepare
- RETI: Return from trap or interrupt
- RIE Reserved instruction exception
- STSR: Store system register
- SWITCH: Jump with table look up
- TRAP: Trap
- SYNCM: Synchronize memory
- SYNCP: Synchronize pipeline
- SYNCE: Synchronize exceptions
- SYSCALL: System call

5.3 Instruction Set

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

- Instruction format: Indicates the description and the instruction operand (for symbols, refer to **Table 5-1**).
- Operation: Indicates the function of the instruction (for symbols, refer to **Table 5-2**).
- Format: Indicates the instruction format (refer to **5.1 Opcodes and Instruction Formats**).
- Opcode: Indicates the bit field of the instruction opcode (for symbols, refer to **Table 5-3**).
- 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.
- Remark: Provides supplementary information on instruction.
- Caution: Provides precautionary notes.

Table 5-1. 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
vector×	Data to specify vector (× indicates the bit size)
cond	Condition code (refer to **Table 5-4 Condition Codes**)
cccc	4-bit data to specify condition code (refer to **Table 5-4 Condition Codes**)
sp	Stack pointer (r3)
ep	Element pointer (r30)
list12	Lists of registers

Table 5-2. Conventions of Operation

Symbol	Meaning
←	Assignment
GR [ ]	General-purpose register
SR [ ]	System register
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
saturate (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)
+	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 5-3. 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 (Refer to **Table 5-4 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 5-4. 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] Bcond disp9

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

[Format] Format III

[Opcode] 15 0

ddddd1011dddcccc

ddddddddd is the higher 8 bits of disp9.
cccc is the condition code of the condition indicated by cond (refer to **Table 5-5 Bcond Instructions**).

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

[Description] 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).

[Comment] 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 5-5. 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)
	BSA	1101	SAT = 1	Saturated
	BT	0010	Z = 1	True
	BV	0000	OV = 1	Overflow
	BZ	0010	Z = 1	Zero

Caution 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.

<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 ← PSW
adr ← CTBP + zero-extend (imm6 logically shift left by 1)
PC ← CTBP + zero-extend (Load-memory (adr, Halfword))

[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. In the CALLT instruction memory read operation executed in order to read the table, processor protection is performed.
3. When memory protection (PSW.DMP = 1) or peripheral device protection (PSW.PP = 1) is enabled, loading 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 performed.

<Special instruction>

CAXI

Compare and exchange for interlock

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 The lower 2 bits of GR [reg1] is masked to 0 as adr.

[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 result operation; otherwise, “0”
S	“1” if result is negative; otherwise, “0”
Z	“1” if result is 0; otherwise, “0”
SAT	--

[Description] First, the data in general-purpose register reg1 is read and the lower two bits are masked to “0”, then a 32-bit address aligned to the word boundary is generated. Word data is read from the generated address, then is 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.

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.

<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)
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]
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← clear-bit (token, reg2)
Store-memory (adr, token, Byte)

[Format] (1) Format VIII

(2) Format IX

[Opcode]

	15 0	31 16
(1)	10bbb111110RRRRR	ddddddddddddddddd
(2)	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.

[Comment] 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>

CMOV

Conditional move

Conditional transfer

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

(2) CMOV cccc, imm5, reg2, reg3

[Operation] (1) if conditions are satisfied

then GR [reg3] ← GR [reg1]

else GR [reg3] ← GR [reg2]

(2) if conditions are satisfied

then GR [reg3] ← sign-extended (imm5)

else GR [reg3] ← GR [reg2]

[Format] (1) Format XI

(2) Format XII

[Opcode]

(1)

15	0	31	16
rrrrr111111RRRRR		wwwww011001cccc0	

(2)

15	0	31	16
rrrrr111111iiiii		wwwww011000cccc0	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

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

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

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

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

[Comment] 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]

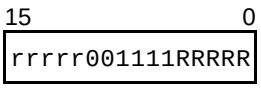
(2) result ← GR [reg2] – sign-extend (imm5)

[Format] (1) Format I

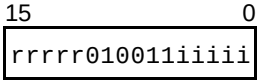
(2) Format II

[Opcode]

(1)



(2)



[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 ← CTPSW

[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 from the appropriate system register and returns from a routine under CALLT instruction. The following steps are taken:

(1) The return PC and PSW are loaded from the CTPC and CTPSW.

(2) The values are restored in PC and PSW and the control is transferred to the return address.

<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 immediately disable the acknowledgement of EI level maskable exceptions.

[Comment] Overwrite of flags in the PSW by this instruction becomes valid as of the next instruction.

<Special instruction>

DISPOSE	Function dispose Stack frame deletion

[Instruction format] (1) DISPOSE imm5, list12

(2) DISPOSE imm5, list12, [reg1]

[Operation] (1) adr ← sp + zero-extend (imm5 logically shift left by 2)

foreach (all regs in list12) {

GR[reg in list12] ← Load-memory (adr, Word)[Note]

adr ← adr + 4

}

sp ← adr

(2) adr ← sp + zero-extend (imm5 logically shift left by 2)

foreach (all regs in list12) {

GR[reg in list12] ← Load-memory (adr, Word)[Note]

adr ← adr + 4

}

sp ← adr

PC ← GR[reg1]

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

[Format] Format XIII

[Opcode]

(1)

15	0	31	16
0000011001iiiiiL		LLLLLLLLLLL00000	

(2)

15	0	31	16
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 bit21 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.

[Comment]

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 may 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.

[Comment] 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] ÷ GR [reg1]

(2) GR [reg2] ← GR [reg2] ÷ GR [reg1]
GR [reg3] ← GR [reg2] % GR [reg1]

[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.

[Comment]

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

[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.

[Comment] 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>

DIVQ

Divide word quickly

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.

[Comment]

(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 processing 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, refer to C.2 Clock Requirements.
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.

[Comment]

(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 processing 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, refer to C.2 Clock Requirements.
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.

[Comment] 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.

<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. When EP = 0, completed execution of the exception routine is reported externally (to the interrupt controller, etc.).

<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. When EP = 0, completed execution of the exception routine is reported externally (to the interrupt controller and elsewhere).

<Special instruction>

FETRAP

FE-level Trap

FE level software exception

[Instruction format] FETRAP vector4

[Operation]
FEPC ← PC + 2 (return PC)
FEPSW ← PSW
ECR.FECC ← exception code (31H-3FH)
FEIC ← exception code (31H-3FH)
PSW.EP ← 1
PSW.ID ← 1
PSW.NP ← 1
If (MPM.AUE==1) is satisfied
then PSW.IMP ← 0
PSW.DMP ← 0
PSW.NPV ← 0
PSW.PP ← 0
PC ← 00000030H

[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 an exception source code in the FEIC register and ECR.FECC bit, and sets (1) the PSW.NP, EP, and ID bits. If the MPM.AUE bit is set (1), it clears (0) the PSW.PP, NPV, DMP, and IMP bits. Execution then branches to the exception handler address (00000030H) and exception processing is started.

<Special instruction>

HALT	Halt Halt

[Instruction format] HALT

[Operation] Instruction execution is halted until the HALT state release request is generated.

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000100100000

[Flags]

CY --
OV --
S --
Z --
SAT --

[Description] Places the system 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.

A HALT state release request is input when the following exception requests occur.

- Reset input (RESET)
- EI level maskable interrupt input (INT0 to INT255)
- FE level maskable interrupt input (FEINT)
- FE level non-maskable interrupt input (FENMI)
- Peripheral device protection exception (PPI)
- Timing monitoring exception (TSI)
- System error exception (SYSERR)

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 ID = 1, HALT state is released when INT0 occurs).

[Comment] The HALT status is not released if interrupt inputs (INT0 to INT255, FEINT, and FENMI) are “disabled” by the following registers of the interrupt controller.

- EI level interrupt control registers (EIC0 to EIC255)
- EI level interrupt mask registers (IMR0 to IMR15)

<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

[Operation] (1) GR [reg2] ← PC + 4

PC ← PC + sign-extend (disp22)

(2) GR [reg1] ← PC + 6

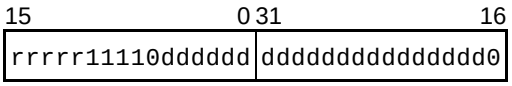
PC ← PC + disp32

[Format] (1) Format V

(2) Format VI

[Opcode]

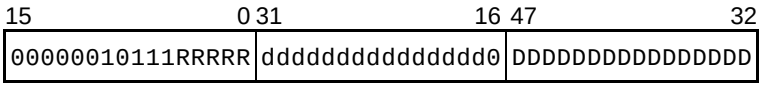
(1)



ddddddddddddddddddddd is the higher 21 bits of disp22.

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

(2)



DDDDDDDDDDDDDDDDddddddddddddddd is the higher 31 bits of disp32.

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

[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”.

[Comment] 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 reg2 in instruction format (1) JARL disp22, reg2.
Do not specify r0 for reg1 in instruction format (2) JARL disp32, reg1.

<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”.

[Comment] 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”.

[Comment] 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)

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

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

GR [reg3] ← sign-extend (Load-memory (adr, Byte))

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15	0	31	16
rrrrr111000RRRRR		ddddddddddddddd	

(2)

15	0	31	16	47	32
00000111100RRRRR		wwwwwddddddd0101		DDDDDDDDDDDDDDD	

Where RRRRR = reg1, wwwww = reg3.

ddddddd is the lower 7 bits of disp23.

DDDDDDDDDDDDDDD is the higher 16 bits of disp23.

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, 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)

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

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

GR [reg3] ← zero-extend (Load-memory (adr, Byte))

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15	031	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	031	1647	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.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)

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

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

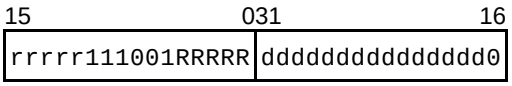
GR [reg3] ← sign-extend (Load-memory (adr, Halfword))

[Format] (1) Format VII

(2) Format XIV

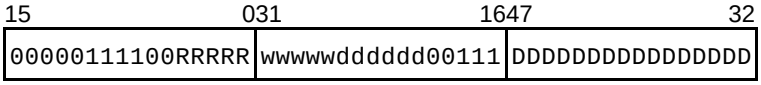
[Opcode]

(1)



Where ddddddddddddddd is the higher 15 bits of disp16.

(2)



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.

<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)

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

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

GR [reg3] ← zero-extend (Load-memory (adr, Halfword))

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15	0	31	16
rrrrr111111RRRRR		ddddddddddddddd1	

Where ddddddddddddddd is the higher 15 bits of disp16.

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

(2)

15	0	31	16	47	32
00000111101RRRRR		wwwwwddddddd00111		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.

Caution Do not specify r0 for reg2.

<Load instruction>

LD.W	Load word Load of word data

[Instruction format] (1) LD.W disp16 [reg1] , reg2

(2) LD.W disp23 [reg1] , reg3

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

GR [reg2] ← Load-memory (adr, Word)

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

GR [reg3] ← Load-memory (adr, Word)

[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.

<Special instruction>

LDSR	Load to system register Load to system register

[Instruction format] LDSR reg2, regID

[Operation] SR [regID] ← GR [reg2]

[Format] Format IX

[Opcode]

15	0	31	16
rrrrr111111RRRRR		0000000000100000	

Caution	**The fields to define reg1 and reg2 are swapped in this instruction. "RRR" is normally used for reg1 that is the source operand, and "rrr" is represented by reg2 that is the destination operand. In this instruction, "RRR" is used for the source operand that is represented by reg2, and "rrr" is used for the register destination.** **rrrrr: regID specification** **RRRRR: reg2 specification**

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] Loads the word data of general-purpose register reg2 to a system register specified by the system register number (regID). General-purpose register reg2 is not affected.

Caution	**The system register number regID is to identify a system register. Accessing system registers that are reserved or write-prohibited is prohibited.**

<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 re1, reg2, reg3, or reg3+1.

Caution	**General-purpose registers that can be specified as reg3 or reg4 must be an even-numbered register (r0, r2, r4, …, r30). The result is undefined if an odd-numbered register (r1, r3, …, r31) is specified.**

<Arithmetic instruction>

MOV	Move register/immediate (5-bit) /immediate (32-bit) Data 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]

15 0

(1)

rrrrr000000RRRRR

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

15 0

(2)

rrrrr010000iiiii

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

15 0 31 16 47 32

(3)

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.

[Comment] 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.

[Comment] 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.

[Comment] 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] (32) ← GR [reg2] (16) × GR [reg1] (16)

(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.

[Comment] In the case of a multiplier or a multiplicand, the higher 16 bits of general-purpose registers reg1 and reg2, are ignored.

Caution Do not specify r0 for reg2.

<Multiply instruction>

MULHI	Multiply halfword by immediate (16-bit) Multiplication of (signed) halfword immediate data

[Instruction format] MULHI imm16, reg1, reg2

[Operation] GR [reg2] ← GR [reg1] × 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.

[Comment] 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.

[Comment] 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] PC for this instruction is incremented by +2 (nothing else is done).

[Format] Format I

[Opcode]

15	0
0000000000000000	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] PC for this instruction is incremented by +2 (nothing else is done).

[Comment] 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)
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]
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← not-bit (token, reg2)
Store-memory (adr, token, Byte)

[Format] (1) Format VIII

(2) Format IX

[Opcode]

	15	0	31	16
(1)	01bbb111110RRRRR		ddddddddddddddd	
(2)	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”.

[Comment] 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>

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

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

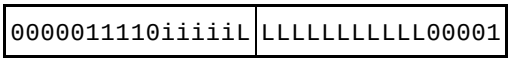
case
ff = 00: ep ← sp
ff = 01: ep ← sign-extend (imm16)
ff = 10: ep ← imm16 logically shift left by 16
ff = 11: ep ← imm32

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

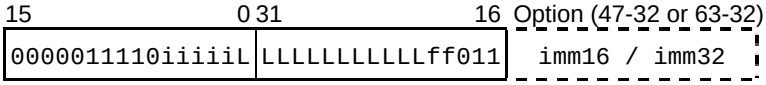
[Format] Format XIII

[Opcode]

(1)



(2)



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.

[Comment]

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 may be stopped after the read 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>

RETI	Return from trap or interrupt Return from EI level software exception or interrupt

[Instruction format] RETI

[Operation]
```
if PSW.EP = 1
then  PC ← EIPC
      PSW ← EIPSW
else if PSW.NP = 1
        then  PC ← FEPC
              PSW ← FEPSW
        else  PC ← EIPC
              PSW ← EIPSW
```

[Format] Format X

[Opcode]

15 0	31 16
0000011111100000	0000000101000000

[Flags]

CY	Value read from FEPSW or EIPSW is set.
OV	Value read from FEPSW or EIPSW is set.
S	Value read from FEPSW or EIPSW is set.
Z	Value read from FEPSW or EIPSW is set.
SAT	Value read from FEPSW or EIPSW is set.

[Description] Reads the return PC and PSW from the appropriate system register and returns from a software exception or interrupt routine. The following steps are taken:

(1) If the EP bit of PSW is “1”, the return PC and PSW are read from EIPC and EIPSW, regardless of the status of the NP bit of PSW.
If the EP bit of PSW is “0” and the NP bit of PSW is “1”, the return PC and PSW are read from FEPC and FEPSW.
If the EP bit of PSW is “0” and the NP bit of PSW is “0”, the return PC and PSW are read from EIPC and EIPSW.

(2) The values are restored in PC and PSW and the control is transferred to the return address.

When EP = 0, completed execution of the exception routine is reported externally (to the interrupt controller or elsewhere).

Cautions 1. The RETI instruction is defined for backward compatibility with the V850E1 and V850E2 CPU. Therefore, in principle, use of the RETI instruction is prohibited. Except for existing programs that cannot be revised, all RETI instructions should be replaced with EIRET or FERET instructions.
If the RETI instruction is used, the operation is undefined except when returning from an interrupt or EI level software exception.

2. To enable normal restoration of the PC and PSW when returning (via a RETI instruction) from an FE level non-maskable interrupt exception (FENMI), FE level maskable interrupt exception (FEINT), or an EI level software exception (TRAP), the NP and EP bits must be set as follows just before executing the RETI instruction.

- **When using RETI instruction to return from FE level non-maskable interrupt exception (FENMI): NP = 1 and EP = 0**
- **When using RETI instruction to return from FE level maskable interrupt exception (FEINT): NP = 1 and EP = 0**
- **When using RETI instruction to return from EI level software exception (EITRAP0/EITRAP1): EP = 1**

<Special instruction>

RIE	Reserved instruction exception Reserved instruction exception

[Instruction format] (1) RIE
(2) RIE imm5, imm4

[Operation]
FEPC ← PC (return PC)
FEPSW ← PSW
ECR.FECC ← exception code
FEIC ← exception code
PSW.NP ← 1
PSW.EP ← 1
PSW.ID ← 1
If (MPM.AUE==1) is satisfied
then PSW.IMP ← 0
PSW.DMP ← 0
PSW.NPV ← 0
PSW.PP ← 0
PC ← 00000030H

[Format] (1) Format I
(2) Format X

[Opcode]

(1)

15	0
0000000001000000	

(2)

15	0	31	16
iiiii1111111IIII		0000000000000000	

Where iiiii = imm5, IIII = imm4.

[Flags]
CY –
OV –
S –
Z –
SAT –

[Description] Saves the contents of the return PC (address of the RIE instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores an exception source code in the FEIC register and ECR.FECC bit, and sets (1) the PSW.NP, EP, and ID bits. If the MPM.AUE bit is set (1), it clears (0) the PSW.PP, NPV, DMP, and IMP bits.
Execution then branches to the exception handler address (00000030H) and exception processing is started.

<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

(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. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions. 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. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions.

(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. When the number of shifts is 0, general-purpose register reg3 retains the value prior to execution of instructions. 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

[Comment] Refer to 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]

(1)

15	0
rrrrr000110RRRRR	

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

(2)

15	0
rrrrr010001iiiii	

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

(3)

15	0	31	16
rrrrr111111RRRRR		wwwww01110111010	

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

[Description]
(1) Adds the word data of general-purpose register reg1 to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, when the result exceeds the maximum positive value 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.

[Comment] 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.

[Comment] 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.

[Comment] 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.

[Comment] 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)	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). When bit (0) is eventually found, 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). When bit (0) is eventually found, 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). When bit (1) is eventually found, 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
rrrrr11111100000		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). When bit (1) is eventually found, 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)
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]
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, reg2))
token ← set-bit (token, reg2)
Store-memory (adr, token, Byte)

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

[Opcode]

(1)

15 0	31 16
00bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100000

[Flags]

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

[Description] (1) Adds the word data of general-purpose register reg1 to 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”.

[Comment] 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

[Comment] 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. Since 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]

(1)

15	0	31	16
rrrrr111111RRRRR		0000000011000000	

(2)

15	0
rrrrr010110iiiii	

(3)

15	0	31	16
rrrrr111111RRRRR		wwwww00011000010	

[Flags]

CY	“1” if the last bit shifted out is “1”; otherwise, “0” including non-shift.
OV	0
S	“1” if the operation result is negative; otherwise, “0”.
Z	“1” if the operation result is “0”; otherwise, “0”.
SAT	--

[Description] (1) Logically left-shifts the word data of general-purpose register reg2 by ‘n’ (0 to +31), the position specified by the lower 5 bits of general-purpose register reg1, by shifting “0” to LSB. The result is written to general-purpose register reg2. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions. 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. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions.

(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. When the number of shifts is 0, general-purpose register reg3 retains the value prior to execution of instructions. 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. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions. 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. When the number of shifts is 0, general-purpose register reg2 retains the value prior to execution of instructions.

(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. When the number of shifts is 0, general-purpose register reg3 retains the value prior to execution of instructions. General-purpose registers reg1 and reg2 are not affected.

<Load instruction>

SLD.B	Short format load byte Load of (signed) byte data

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

[Operation] adr ← ep + zero-extend (disp7)
GR [reg2] ← sign-extend (Load-memory (adr, Byte))

[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>

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

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

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

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

[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.

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

[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.

Caution Do not specify r0 for reg2.

<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)
GR [reg2] ← Load-memory (adr, Word)

[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.

<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)
Store-memory (adr, GR [reg2] , Byte)

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

[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.

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

[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.

<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)
Store-memory (adr, GR [reg2], Byte)

(2) adr ← GR [reg1] + sign-extend (disp23)
Store-memory (adr, GR [reg3], Byte)

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15	0	31	16
rrrrr111010RRRRR		ddddddddddddddddd	

(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.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)
Store-memory (adr, GR [reg2], Halfword)

(2) adr ← GR [reg1] + sign-extend (disp23)
Store-memory (adr, GR [reg3], Halfword)

[Format] (1) Format VII

(2) Format XIV

[Opcode]

(1)

15	0	31	16
rrrrr111011RRRRR		ddddddddddddddd0	

Where ddddddddddddddd is the higher 15 bits of disp16.

(2)

15	0	31	16	47	32
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 lowest halfword data of general-purpose register reg3 to the generated address.

<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)

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

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

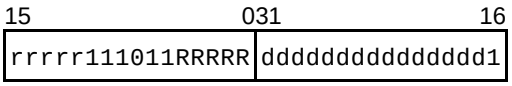
Store-memory (adr, GR [reg3], Word)

[Format] (1) Format VII

(2) Format XIV

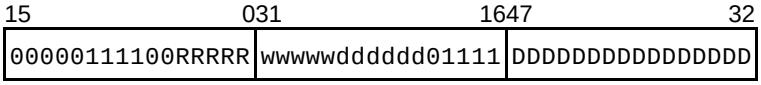
[Opcode]

(1)



Where ddddddddddddddd is the higher 15 bits of disp16.

(2)



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 lowest word data of general-purpose register reg3 to the generated 32-bit address.

<Special instruction>

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

[Instruction format] STSR regID, reg2

[Operation] GR [reg2] ← SR [regID]

[Format] Format IX

[Opcode]

15	0	31	16
rrrrr111111RRRRR		0000000001000000	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] Stores the system register contents specified by the system register number (regID) to general-purpose register reg2. The system-register contents are not affected.

Caution The system register number regID is to identify a system register. Operation is not guaranteed if the reserved system register ID is specified.

<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]

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)

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

[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, processor protection is performed.

3. When memory protection (PSW.DMP = 1) or peripheral device protection (PSW.PP = 1) is enabled, loading 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 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>

Synchronize exceptions

SYNCE

Exception synchronization instruction

[Instruction format] SYNCE

[Operation] Starts execution when exceptions are synchronized, and increments PC by +2 without executing anything.

[Format] Format I

[Opcode]

15	0
0000000000011101	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] Waits for the synchronization of all preceding exceptions before starting execution.
It does not perform any operation but is completed when its execution is started.
“Exception synchronization” means that all 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 before this instruction is executed, therefore, all imprecise exceptions (FPI and PPI exceptions) that are generated because of the preceding instructions are always acknowledged before execution of this instruction is completed. This instruction can be used to guarantee completion of exception processing by the preceding task before a task is changed or terminated in a multi-processing environment.

<Special instruction>

SYNCM	Synchronize memory Memory synchronize instruction

[Instruction format] SYNCM

[Operation] Starts execution when accesses to the memory device are synchronized, and increments PC by +2 without executing anything.

[Format] Format I

[Opcode]

15	0
0000000000011110	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] Waits for the synchronization of all preceding memory accesses before starting execution.
“Synchronization” refers to the status where the result of preceding memory accesses can be referenced by any master device within the system.
In cases such as when buffering is used to delay memory accesses and synchronization of all memory accesses has not occurred, the SYNCM instruction does not complete and waits for the synchronization.
The subsequent instructions will not be executed until the SYNCM instruction execution is complete.

<Special instruction>

SYNCP	Synchronize pipeline Pipeline synchronize instruction

[Instruction format] SYNCP

[Operation] Starts execution when pipeline is synchronized, and increments PC by +2 without executing anything.

[Format] Format I

[Opcode]

15	0
0000000000011111	

[Flags]

CY	--
OV	--
S	--
Z	--
SAT	--

[Description] Waits until execution of all previous instructions is completed before being executed. Execution of this instruction increments PC by +2.

<Special instruction>

SYSCALL	System call System call exception

[Instruction format] SYSCALL vector8

[Operation]
EIPC ← PC + 4 (return PC)
EIPSW ← PSW
EIIC ← exception code (8000H-80FFH)
ECR.EICC ← exception code (8000H-80FFH)
PSW.EP ← 1
PSW.ID ← 1
If (MPM.AUE==1) is satisfied
then PSW.IMP ← 0
PSW.DMP ← 0
PSW.NPV ← 0
PSW.PP ← 0
if (vector8 <= SCCFG.SIZE) is satisfied
then adr ← SCBP + zero-extend (vector8 logically shifted left by 2)
else adr ← SCBP
PC ← SCBP + Load-memory (adr, Word)

[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] This instruction calls the system service of an OS.

<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 code corresponding to vector8 to the EIIC register and ECR.EICC bit. The exception code is the value of vector8 plus 8000H.

<3> Sets (1) the PSW.ID and EP bits.

<4> Clears (0) the PSW.PP, NPV, DMP, and IMP bits when the MPM.AUE bit is 1.

<5> 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 above addition is handled as 0.

<6> Loads the word of the address generated in <5>.

<7> Generates a 32-bit target address by adding the value of the SCBP register to the data in <6>.

<8> Branches to the target address generated in <7>.

Cautions 1. This instruction is dedicated to calling the system service of an OS. For how to use it in the user program, refer to the Function Specification of each OS.

2. In the SYSCALL instruction memory read operation executed in order to read the table, processor protection is not performed.

3. When memory protection (PSW.DMP = 1) or peripheral device protection (PSW.PP = 1) is enabled, loading the data for generating a target address from a table allocated in an area to which access from a user program is prohibited can be performed.

<Special instruction>

TRAP	Trap Software exception

[Instruction format] TRAP vector5

[Operation]
EIPC ← PC + 4 (return PC)
EIPSW ← PSW
ECR.EICC ← exception code (40H to 5FH)
EIIC ← exception code (40H to 5FH)
PSW.EP ← 1
PSW.ID ← 1
If (MPM.AUE==1) is satisfied
then PSW.IMP ← 0
PSW.DMP ← 0
PSW.NPV ← 0
PSW.PP ← 0
PC ← 00000040H (when vector5: 00H to 0FH (exception code: 40H to 4FH))
00000050H (when vector5: 10H to 1FH (exception code: 50H to 5FH))

[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 source code in the EIIC register and ECR.EICC bit, and sets (1) the PSW.EP and ID bits. If the MPM.AUE bit is set (1), it clears (0) the PSW.PP, NPV, DMP, and IMP bits.
It then branches to an exception handler address corresponding to the vector (00H to 1FH) specified as “vector5” and starts exception processing.

<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)
token ← Load-memory (adr, Byte)
Z flag ← Not (extract-bit (token, bit#3))

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

[Format] (1) Format VIII

(2) Format IX

[Opcode]

(1)

15	0	31	16
11bbb111110RRRRR		ddddddddddddddd	

(2)

15	0	31	16
rrrrr111111RRRRR		0000000011100110	

[Flags]

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

[Description] (1) Adds the word data of general-purpose register reg1 to the16-bit displacement data, sign-extended to word length, to generate a 32-bit address; checks the bit specified by the 3-bit 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
r	r	r	r	r	0	0	1	0	0	1	R	R	R	R	R

[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.

CHAPTER 6 EXCEPTIONS

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”. The exception handler start address is set by the exception handler address switching function (see **6.4 Exception Handler Address Switching Function**).

Caution The V850E2M CPU handles interrupts of the V850E1 and V850E2 CPU as types of exceptions.

6.1 Outline of Exceptions

The following describes the elements that assign properties to exceptions, and shows how exceptions work.

- Exception cause list
- Exception types
- Exception processing flow
- Interrupts
- Priority of exception acknowledgment
- Exception acknowledgment condition
- Resume and restoration
- Exception level and context saving
- Return instructions

6.1.1 Exception cause list

The V850E2M CPU supports the following types of exceptions.

Table 6-1. Exception Cause List (1/2)

Name	Symbol	Cause	Priority	Exception Level	Type	Resume	Restoration	Acknowledgment Condition (×: 0 or 1)		Exception Code[Note 1]	Return PC[Note 1]	Register Refresh Value (s: save)					Return Instruction
								PSW				Handler Offset[Note 2]	PSW				
								ID	NP				Execution Level[Note 3]	NP	EP	ID	
CPU initialization	RESET	Reset input	1	–	Asynchronous	NG	NG	×	×	None	None	+0000H	0	0	0	1	None
FE level non-maskable interrupt	FENMI	FENMI input[Note 4]	3	FE	Interrupt	NG	NG	×	×	00000020H	currentPC	+0020H	**Note 5**	1	0	1	FERET
System error exception	SYSERR	SYSERR input[Note 7]	4	FE	**Note 6**	NG	NG	×	×	00000230H : 00000233H	currentPC	+0030H	**Note 5**	1	1	1	FERET
Peripheral device protection exception	PPI	Peripheral device protection violation	5	FE	Imprecise	OK	NG	×	0	00000432H	currentPC	+0030H	0	1	1	1	FERET
Timing monitoring exception	TSI	Timing monitoring violation	6	FE	Asynchronous	OK	OK	×	0	00000433H	currentPC	+0030H	0	1	1	1	FERET
FE level maskable interrupt	FEINT	FEINT input[Note 4]	7	FE	Interrupt	OK	OK	×	0	00000010H	currentPC	+0010H	**Note 5**	1	0	1	FERET
Floating-point operation exception (imprecise)	FPI	FPU instruction	8	EI	Imprecise	OK	NG	0	0	00000072H	currentPC	+0070H	**Note 5**	s	1	1	EIRET
EI level maskable interrupt	INT	INTn input[Note 4] (n = 0 to 255)	9	EI	Interrupt	OK	OK	0	0	00000080H : 00001070H	currentPC	+0080H : +1070H	**Note 5**	s	0	1	EIRET

Notes 1. The return PC and PSW, and the exception code storage destination are specified by the exception level (EI or FE) (nextPC: next instruction, currentPC: current instruction).

2. The base address is set by the exception handler switching function.

3. For details of the execution level, see **CHAPTER 4 EXECUTION LEVEL** in **PART 3**.

4. Input is from INTC.

5. The execution level changes to 0 when MPM.AUE = 1. It does not change when MPM.AUE = 0.

6. Each cause may be asynchronous or imprecise, depending on the implementation of the product.
Generally, the cause is asynchronous but it may be imprecise if an error that occurs because of an instruction, such as a data error, is defined.

7. The causes of the SYSERR depend on the implementation of the product.

Remark In the table, *Priority* refers to the order in which exceptions that have occurred at the same time and for which the acknowledgement conditions have been met are acknowledged.

Table 6-1. Exception Cause List (2/2)

Name	Symbol	Cause	Priority	Exception Level	Type	Resume	Restoration	Acknowledgment Condition (×: 0 or 1)		Exception Code[Note 1]	Return PC[Note 1]	Register Refresh Value (s: save)					Return Instruction
								PSW				Handler Offset[Note 2]	PSW				
								ID	NP				Execution Level[Note 3]	NP	EP	ID	
Execution protection exception	MIP	Execution protection violation	11	FE	Precise	OK[Note 4]	OK[Note 4]	×	×	00000430H	currentPC	+0030H	0	1	1	1	FERET
Memory error exception	MEP	Instruction access error input[Note 5]	12	FE	Precise	NG[Note 4]	NG[Note 4]	×	×	00000330H : 00000333H	currentPC	+0030H	**Note 6**	1	1	1	FERET
Data protection exception	MDP	Data protection violation	13[Notes 7]	FE	Precise	OK[Note 4]	OK[Note 4]	×	×	00000431H	currentPC	+0030H	0	1	1	1	FERET
Floating-point operation exception (precise)	FPP	FPU instruction		EI	Precise	OK[Note 4]	OK[Note 4]	×	×	00000071H	currentPC	+0070H	**Note 6**	s	1	1	EIRET
Coprocessor unusable exception	UCPOP	Coprocessor instruction		FE	Precise	OK[Note 4]	OK[Note 4]	×	×	00000530H : 00000537H	currentPC	+0030H	**Note 6**	1	1	1	FERET
Reserved instruction exception	RIEX	Reserved instruction		FE	Precise	OK[Note 4]	OK[Note 4]	×	×	00000130H	currentPC	+0030H	**Note 6**	1	1	1	FERET
FE level software exception	FETRAPEX	FETRAP instruction (vector = 1H to FH)		FE	Precise	OK[Note 4]	OK[Note 4]	×	×	00000031H : 0000003FH	nextPC	+0030H	**Note 6**	1	1	1	FERET
EI level software exception	EITRAP0	TRAP0n instruction (vector = 00 to 0FH)		EI	Precise	OK[Note 4]	OK[Note 4]	×	×	00000040H : 0000004FH	nextPC	+0040H	**Note 6**	s	1	1	EIRET
EI level software exception	EITRAP1	TRAP1n instruction (vector = 10H to 1FH)		EI	Precise	OK[Note 4]	OK[Note 4]	×	×	00000050H : 0000005FH	nextPC	+0050H	**Note 6**	s	1	1	EIRET
System call exception	SYSCALLEX	SYSCALL instruction (vector = 00H to FFH)		EI	Precise	OK[Note 4]	OK[Note 4]	×	×	00008000H : 000080FFH	nextPC	**Note 8**	**Note 6**	s	1	1	EIRET

Notes 1. The return PC and PSW, and the exception code storage destination are specified by the exception level (EI or FE) (nextPC: next instruction, currentPC: current instruction).

2. The base address is set by the exception handler switching function.

3. For details of the execution level, see **CHAPTER 4 EXECUTION LEVEL** in **PART 3**.

4. For the instruction access error input, see the **Hardware User's Manual of each product.**

5. These causes occur according to the operation order of each instruction.

6. The execution level changes to 0 when MPM.AUE = 1. It does not change when MPM.AUE = 0.

7. When this occurs during a critical section at the same exception level, values in the original return PC, PSW, etc. may be destroyed.

8. For the branch destination, see SYSCALL Instruction in **5.3 Instruction Set**.

Remark In the table, *Priority* refers to the order in which exceptions that have occurred at the same time and for which the acknowledgement conditions have been met are acknowledged.

6.1.2 Types of exceptions

The V850E2M CPU classifies exceptions into the following four types by their timing of generation and characteristics.

- Precise exception
- Imprecise exception
- Asynchronous exception
- Interrupt

(1) Precise exception

This exception is precise in that it is generated in synchronization with an instruction that has caused it. Examples of this instruction are a software exception that is always generated as result of executing an instruction, and an exception that is immediately generated if the result of instruction execution is illegal. Because execution can branch to exception processing before the following instruction is executed in case of a precise exception, the original processing can be correctly executed after exception processing in many cases[Note].

The following exceptions are classified as precise exceptions.

- Execution protection exception
- Memory error exception
- Data protection exception
- Floating-point operation exception (precise)
- Coprocessor unusable exception
- Reserved instruction exception
- FE level software exception
- EI level software exception
- System call exception

Note If a memory error exception occurs, the original processing cannot be restored because the timing of generation of this exception cannot be controlled.

(2) Imprecise exception

This exception is acknowledged before the operation of an instruction is executed, by aborting that instruction. This is an imprecise exception that is belatedly generated if the result of executing the instruction preceding the instruction that is to be aborted is illegal. In case of an imprecise exception, the original processing cannot be restored and re-executed after processing of the exception because the instruction following the instruction that has caused the imprecise exception may have already been completed, and because the status of the CPU when the exception was caused is not saved.

The following exceptions are classified as imprecise exceptions.

- Peripheral device protection exceptions
- Floating-point operation exceptions (imprecise)

(3) Asynchronous exception

This exception is acknowledged before the operation of an instruction is executed, by aborting that instruction. It is not generated as a result of executing the current instruction but is generated independently of the instruction.

The following exceptions are classified as asynchronous exceptions.

- CPU initialization
- System error exception (Each source depends on the implementation.)
- Timing supervision exception

(4) Interrupt

This exception is acknowledged before the operation of an instruction is executed, by aborting that instruction. It is not generated as a result of executing the current instruction but is generated independently of the instruction. An interrupt is an exception to execute any user program via interrupt controller.

The following exceptions are classified as interrupts.

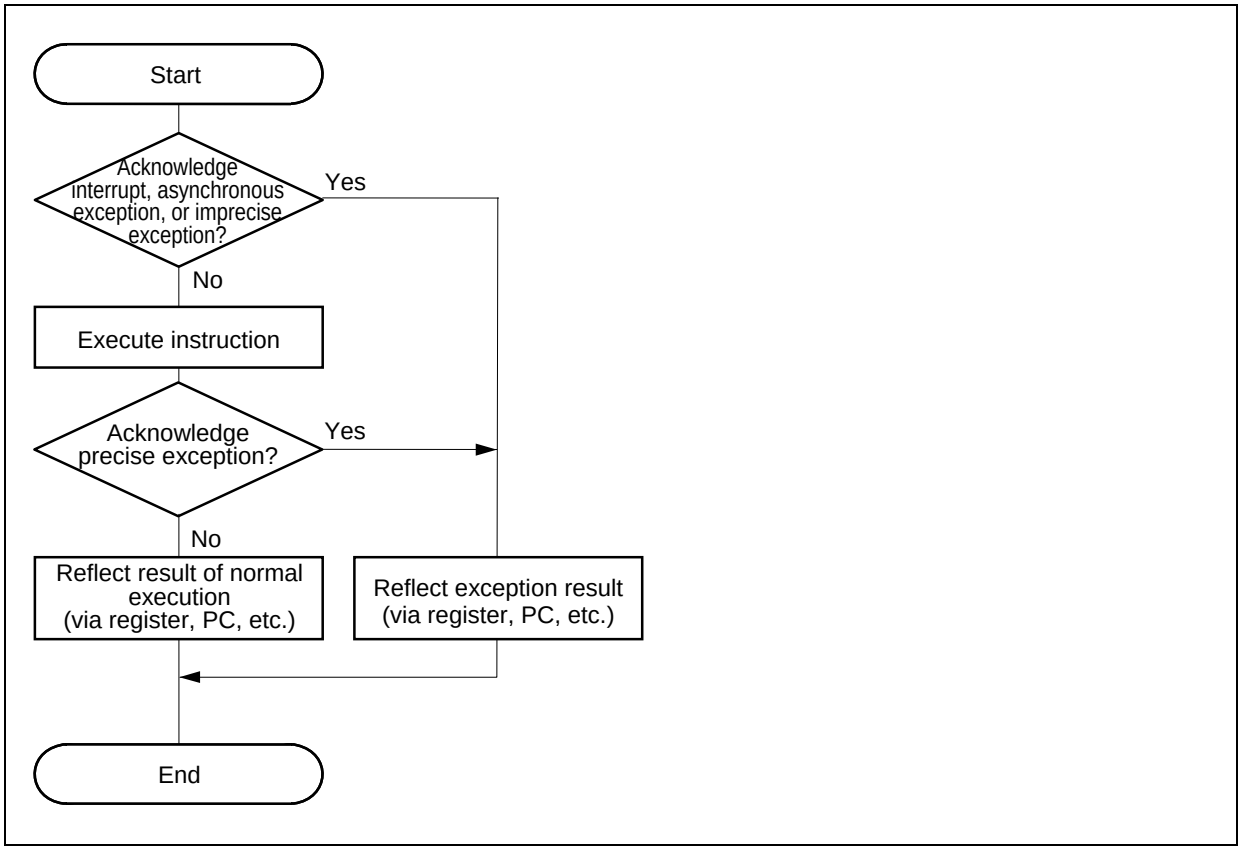
- FE level non-maskable interrupt
- FE level maskable interrupt
- EI level maskable interrupt

Unlike the other exceptions, the PSW.EP bit is cleared (0) when an interrupt is generated. 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 (0).

Caution	**The PSW.EP bit is cleared (0) only when an interrupt (INT0 to INT255, FEINT, or FENMI) is acknowledged. It is set (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 (1), the resources on the external interrupt controller may not be released, causing malfunctioning.**

6.1.3 Exception processing flow

The handling flow for exceptions in relation to instruction execution and results is reflected (by writing to registers, etc.) as shown below.



Acknowledgment or non-acknowledgment of an interrupt, asynchronous exception, and imprecise exception is decided before an instruction is executed. If the exception can be acknowledged, processing branches to exception processing. After exception handing, if the current instruction execution that has been aborted must be executed again, the return PC therefore stores the current instruction (Current PC).

By contrast, when a precise exception occurs, processing branches to exception processing unconditionally, as the instruction execution result. If multiple causes of precise exceptions exist at the same time, only the one with the highest priority is acknowledged. The return PC is determined according to that exception properties, and in cases where the instruction does not have to be re-executed after an exception, such as with a software trap or single step exception, the next instruction (Next PC) is stored. When re-execution is required, such as with a memory protection exception, the current instruction (Current PC) is stored.

6.1.4 Exception acknowledgment priority and pending conditions

Exception acknowledgment is when processing branches to the exception handler corresponding to the exception cause after an exception has occurred due to that exception cause. The CPU is able to acknowledge only one exception at a time. The following priority is used to determine which exception will be acknowledged. When multiple exceptions occur at the same time, exceptions that are not acknowledged are held pending (Except for CPU initialization. For details, refer to **6.2.5 Special operations**).

Table 6-2. Exception Priority

Priority	Exception	Timing
High ↑	CPU initialization (RESET)	Before instruction execution
	FE level non-maskable interrupt (FENMI)	
	System error exception (SYSERR)	
	Peripheral device protection exception (PPI)	
	Timing monitoring exception (TSI)	
	FE level maskable interrupt (FEINT)	
	Floating-point operation exception (imprecise) (FPI)	
	EI level maskable interrupt (INT)	
	Execution protection exception (MIP)	After instruction execution
	Memory error exception (MEP)	
	Data protection exception (MDP)Note	
	Floating-point operation exception (precise) (FPP)Note	
	Coprocessor unusable exception (UCPOP)Note	
	Reserved instruction exception (RIEX)Note	
	FE level software exception (FETRAPEX)Note	
	EI level software exception (EITRAP0/EITRAP1)Note	
↓ Low	System call exception (SYSCALLEX)Note	

Note The priority is the same, and the exception occurs based on the instruction operations.

6.1.5 Exception acknowledgment conditions

The acknowledgment of some exceptions may be held pending according to certain conditions.

Exceptions that are listed in Table 6-1 with “0” in the acknowledgment condition column can be acknowledged only when the relevant bit value is “0”. When one of these exceptions has a relevant bit value of “1”, acknowledgment of the exception is held pending until the relevant bit value becomes “0”, at which time the exception can be acknowledged.

6.1.6 Resume and restoration

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

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

6.1.7 Exception level and context saving

(1) Exception level

The V850E2M CPU manages exception causes in three exception levels (EI level, FE level, and DB level). When an exception occurs, the exception cause, return PC, and return PSW are automatically stored in the corresponding return register according to each level (Except for CPU initialization. For details, refer to **6.2.5 Special operations**).

Table 6-3. Exception Levels

EI Level Exceptions	FE Level Exceptions	DB Level Exceptions[Note]
EI level maskable interrupt EI level software exception Floating-point operation exception System call exception	System error exception FE level maskable interrupt FE level non-maskable interrupt FE level software exception Reserved instruction exception Memory error exception Execution protection exception Data protection exception Peripheral device protection exception Timing monitoring exception Coprocessor unusable exception	Debug exceptionNote

Note The DB level exceptions are used by the debug function for development tools

(2) Context saving

Exceptions with certain acknowledgment conditions may not be acknowledged at the start of exception processing, 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 processing are called basic context registers.

These basic context registers are provided for each level.

Table 6-4. Basic Context Registers

Exception Level	Basic Context Registers
EI level	**EIPC, EIPSW, EIIC, EIWR**
FE level	**FEPC, FEPSW, FEIC, FEWR**
DB level Note	**DBPC**Note**, DBPSW**Note**, DBIC**Note**, DBWR**Note

Note The DB level exceptions are used by the debug function for development tools

6.1.8 Return instructions

To return from exception processing, execute the return instruction (EIRET, 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 may be different from that when the exception does not occur.

(1) EIRET instruction

The EIRET instruction is used to return from exception processing of EI level.

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

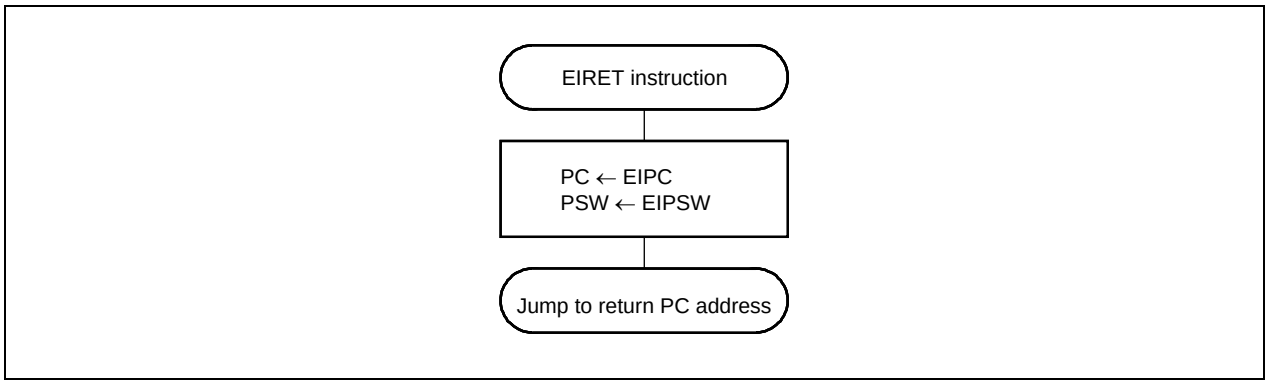
<1> Return PC and PSW are loaded from the EIPC and EIPSW registers.

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

When EP = 0, reports that the exception handler routine execution has been ended to the external units (interrupt controllers, etc.).

A return from EI level exception processing is illustrated below.

Figure 6-1. EIRET Instruction



(2) FERET instruction

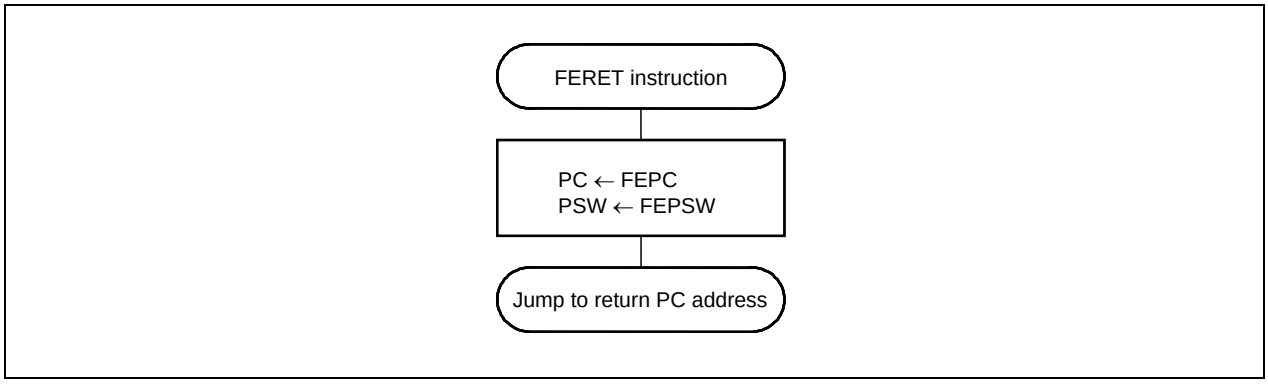
To return from FE level exception processing, execute the FERET instruction.

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

<1> Return PC and PSW are loaded from the FEPC and FEPSW registers.

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

Figure 6-2. FERET Instruction



(3) Execute the RETI instruction to return from an interrupt or EI level software exception (EITRAP0/EITRAP1)

Caution	**The RETI instruction is defined for backward compatibility with the V850E1 and V850E2 CPU. Therefore, in principle, use of the RETI instruction is prohibited. Except for existing programs that cannot be revised, all RETI instructions should be replaced with EIRET or FERET instructions.** **If the RETI instruction is used, the operation is undefined except when returning from an interrupt or EI level software exception (EITRAP0/EITRAP1).**

Execute the RETI instruction to return from an interrupt or EI level software exception (EITRAP0/EITRAP1).
When the RETI instruction is executed, the CPU performs the next processing and then passes control to the return PC address.

<1> When the PSW.EP bit is 0 and the PSW.NP bit is 1, the return PC and PSW are loaded from FEPC and FEPSW. Otherwise, the return PC and PSW are read from EIPC and EIPSW.
<2> Control is passed to the address indicated by the return PC and PSW that were loaded.

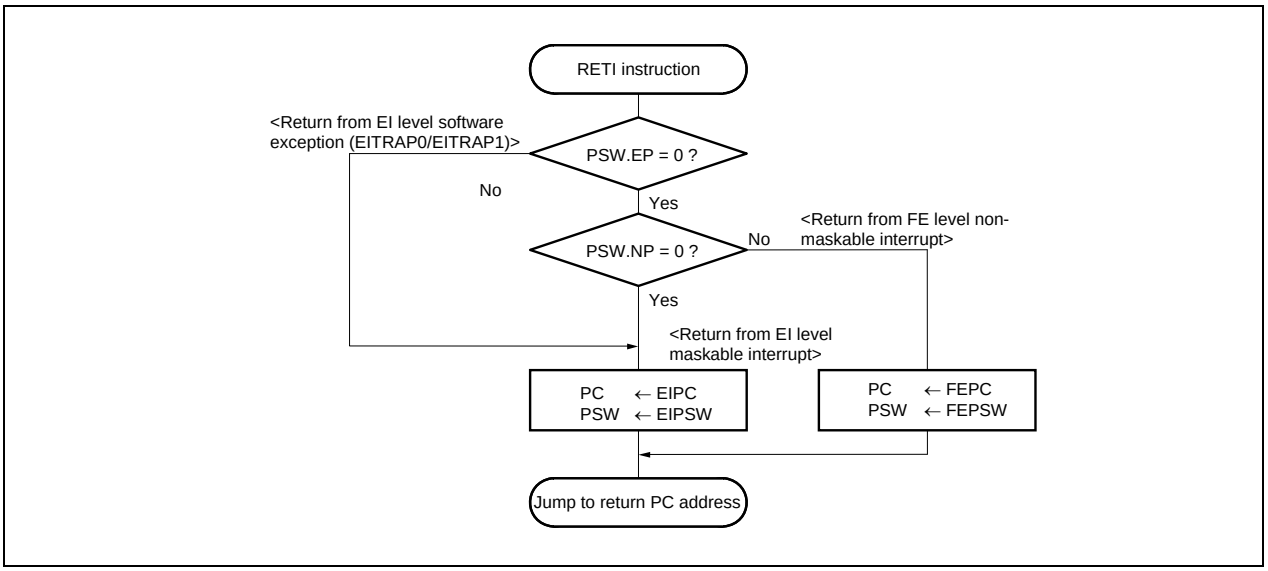
When returning from any type of exception processing, the LDSR instruction must be used just before the RETI instruction to correctly restore the PC and PSW, and the flags for the PSW.NP and PSW.EP bits must be set to the following status.

- When returning from FE level maskable interrupt servicing[Note] : PSW.NP bit = 1, PSW.EP bit = 0
- When returning from EI level maskable interrupt servicing : PSW.NP bit = 0, PSW.EP bit = 0
- When returning from EI level software exception (EITRAP0/EITRAP1) processing : PSW.EP bit = 1

Note The RETI instruction cannot be used to return from FENMI. After exception processing, perform a system reset. FENMI is acknowledged even when the PSW.NP bit is set (1).

The following figure illustrates return processing using the RETI instruction.

Figure 6-3. RETI Instruction



6.2 Operations When Exception Occurs

6.2.1 EI level exception without acknowledgment conditions

This exception can always be acknowledged because it cannot be disabled by changing the instruction or status of the PSW from being acknowledged.

If an EI level exception without acknowledgment conditions occurs, the CPU performs the following processing and transfers control to the exception handler routine.

<1> Saves the return PC to EIPC.

<2> Saves the current PSW to EIPSW.

<3> Writes the exception code to EIIC register[Note].

<4> Sets (1) the PSW.ID bit.

<5> Sets (1) the PSW.EP bit.

<6> Clears (0) the PSW.PP, NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.PP, NPV, DMP, and IMP bits will not be updated.

<7> Sets an exception handler address to the PC and transfers control to the exception handler routine.

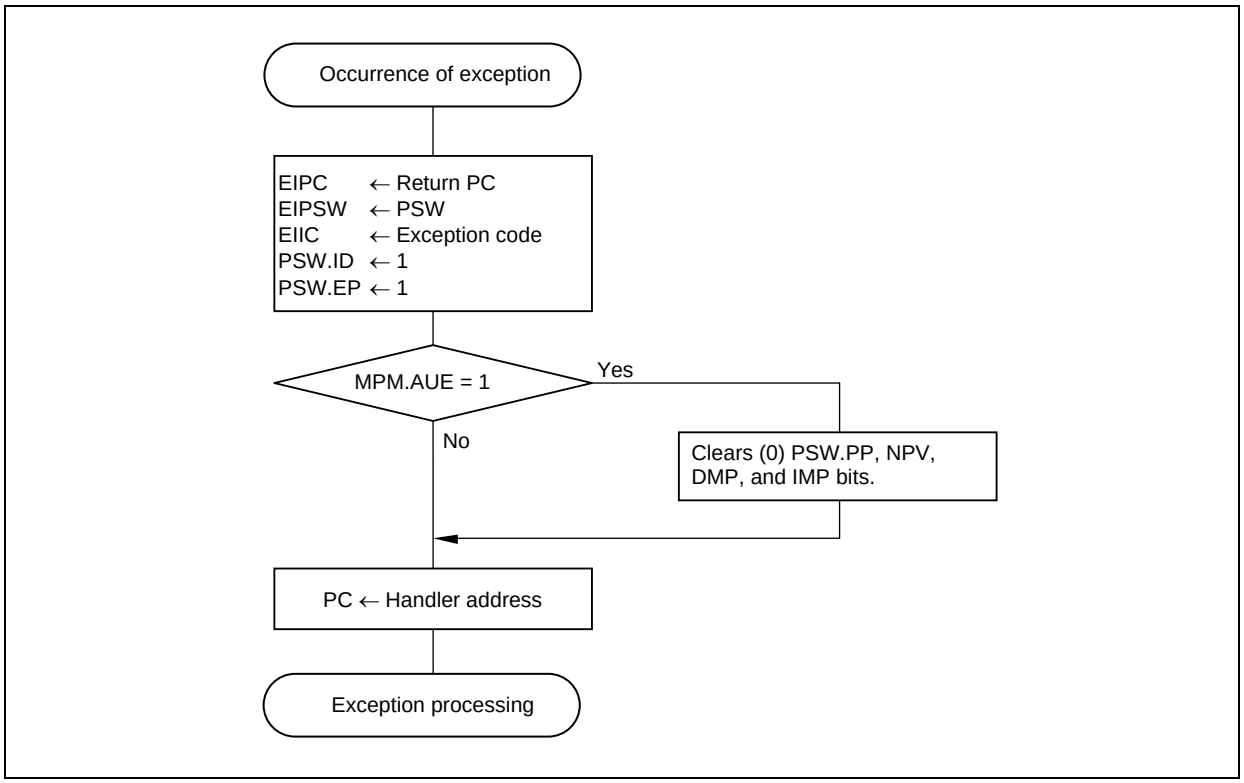
Note Although the exception code is also written to the lower 16 bits (EICC) of the ECR register, the EIIC register should be used except when using an existing program that cannot be revised.

EIPC and EIPSW are used as status save registers. An EI level exception without acknowledgment conditions is acknowledged even if it occurs while another EI level exception is being processed (while the PSW.NP or PSW.ID bit is 1). If an EI level exception without acknowledgment conditions occurs before the context of the EI level exception is saved, therefore, the original PC and PSW may be damaged.

Because only one pair of EIPC and EIPSW is available, the context must be saved in advance by a program before multiple exceptions are enabled.

The format of processing of an EI level exception without acknowledgment conditions is illustrated below.

Figure 6-4. Processing Format of EI Level Exception Without Acknowledgment Conditions



6.2.2 EI level exception with acknowledgment conditions

This exception can be held pending by the PSW.ID and NP bits from being acknowledged.

If an EI level exception with acknowledgment conditions is generated, the CPU performs the following processing and transfers control to the exception handler routine.

<1> Holds the exception pending if the PSW.NP bit is set (1).

<2> Holds the exception pending if the PSW.ID bit is set (1).

<3> Saves the return PC to EIPC.

<4> Saves the current PSW to EIPSW.

<5> Writes an exception code to EIIC[Note].

<6> Sets (1) the PSW.ID bit.

<7> Clears (0) the PSW.EP bit if an interrupt occurs. Sets (1) the PSW.EP bit if any other exception occurs.

<8> Clears (0) the PSW.PP, NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.PP, NPV, DMP, and IMP bits will not be updated.

<9> Sets an exception handler address to the PC and transfers control to the exception handler.

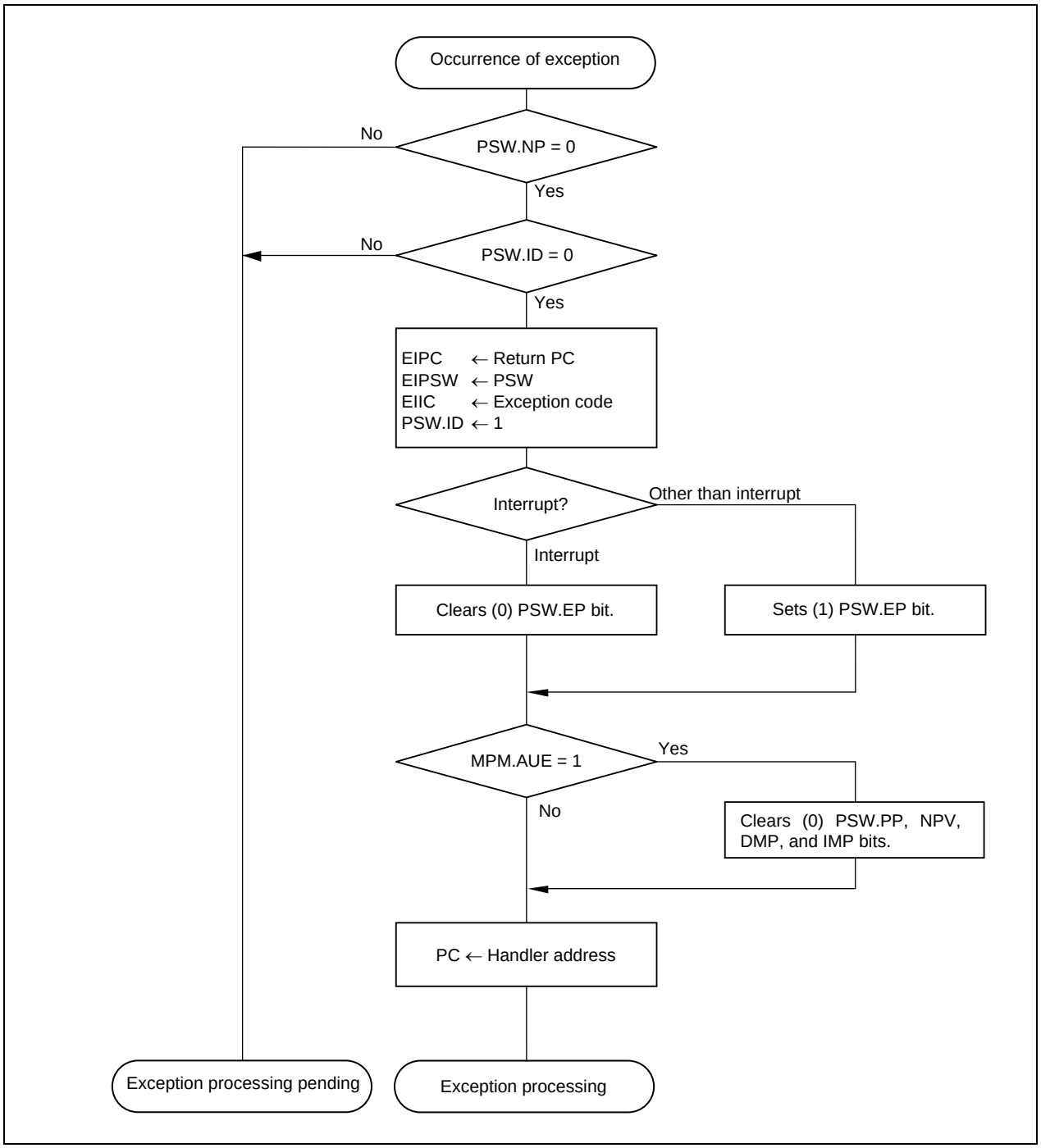
Note Although the exception code is also written to the lower 16 bits (EICC) of the ECR register, the EIIC register should be used except when using an existing program that cannot be revised.

EIPC and EIPSW are used as status save registers. An EI level exception with acknowledgment conditions that has occurred is held pending while other EI level exception is being processed (while the PSW.NP or PSW.ID bit is 1). In this case, if the PSW.NP and ID bits are cleared (0) by using the LDSR or EI instruction, the EI-level exception with acknowledgment conditions which have been held pending is acknowledged.

Because only one pair of EIPC and EIPSW is available, the context must be saved in advance by the program before multiple exceptions are enabled.

The format of processing of an EI level exception with acknowledgment conditions is illustrated below.

Figure 6-5. Processing Format of EI Level Exception with Acknowledgment Conditions



6.2.3 FE level exception without acknowledgment conditions

This exception cannot be disabled by an instruction or by changing the status of PSW from being acknowledged and can always be acknowledged.

If an FE level exception without acknowledgment conditions is generated, the CPU performs the following processing and transfers control to the exception handler routine.

<1> Saves the return PC to FEPC.
<2> Saves the current PSW to FEPSW.
<3> Writes the exception code to FEIC[Note 1].
<4> Sets (1) the PSW.NP and ID bits.
<5> Clears (0) the PSW.EP bit if an interrupt occurs. Sets (1) the PSW.EP bit if any other exception occurs.
<6> Clears (0) the PSW.PP, NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.PP, NPV, DMP, and IMP bits will not be updated[Note 2].
<7> Sets an exception handler address to the PC and transfers control to the handler.

Notes 1. Although the exception code is also written to the higher 16 bits (FECC) of the ECR register, the FEIC register should be used except for when using the existing program that cannot be revised.

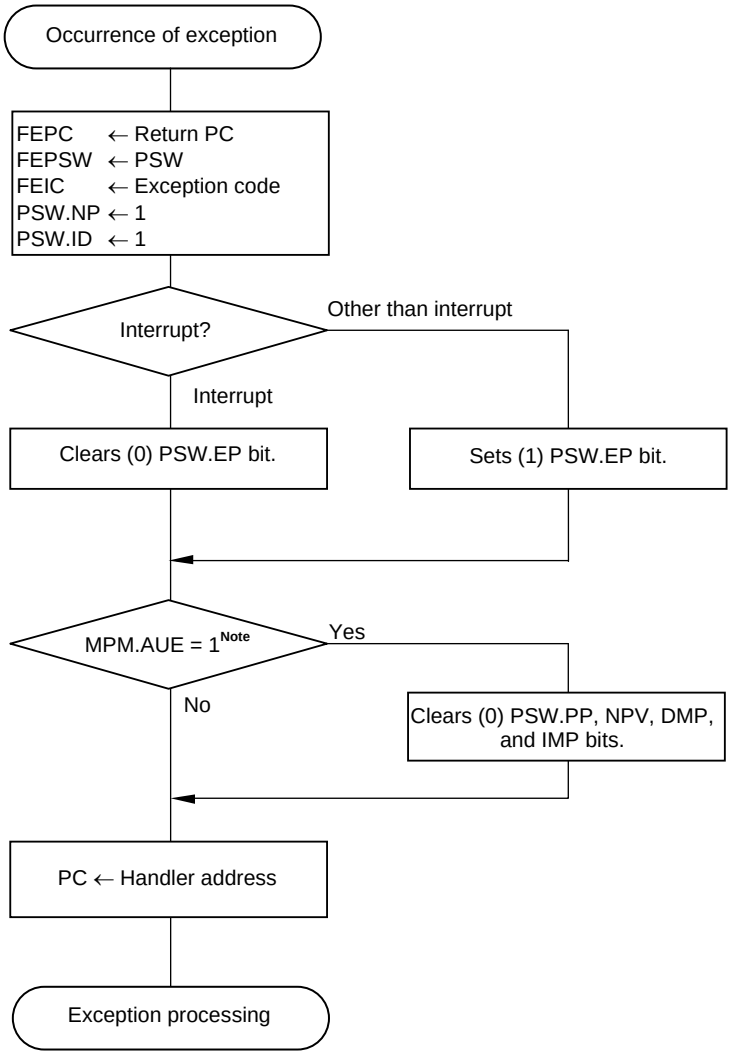
2. The PSW.PP, NPV, DMP, and IMP bits are always cleared (0) if an exception related to processor protection (MDP or MIP exception) occurs.

FEPC and FEPSW are used as status save registers. An FE level exception without acknowledgment conditions is acknowledged even if it occurs while other FE level exception is being processed (while the PSW.NP bit is 1). If the exception occurs before the context of the FE exception level is saved, therefore, the original PC and PSW may be damaged.

Because only one pair of FEPC and FEPSW is available, the context must be saved in advance by a program before multiple exceptions are enabled.

The format of processing of an FE level exception without acknowledgment conditions is illustrated below.

Figure 6-6. Processing Format of FE Level Exception Without Acknowledgment Conditions



Note If a processor protection exception (MDP or MIP) occurs, the PSW.PP, NPV, DMP, and IMP bits are always cleared (0) regardless of the status of MPM.AUE.

6.2.4 FE level exception with acknowledgment conditions

This exception can be held pending by the PSW.NP bit from being acknowledged.

If an FE level exception with acknowledgment conditions is generated, the CPU performs the following processing and transfers control to the exception handler routine.

<1> Holds the exception pending if the PSW.NP bit is set (1).

<2> Saves the return PC to FEPC.

<3> Saves the current PSW to FEPSW.

<4> Writes an exception code to FEIC[Note 1].

<5> Sets (1) the PSW.NP and ID bits.

<6> Clears (0) the PSW.EP bit if an interrupt occurs. Sets (1) the PSW.EP bit if any other exception occurs.

<7> Clears (0) the PSW.PP, NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.PP, NPV, DMP, and IMP bits will not be updated[Note 2].

<8> Sets an exception handler address to the PC and transfers control to the exception handler.

Notes 1. Although the exception code is also written to the lower 16 bits (FECC) of the ECR register, the FEIC register should be used except for when using the existing program that cannot be revised.

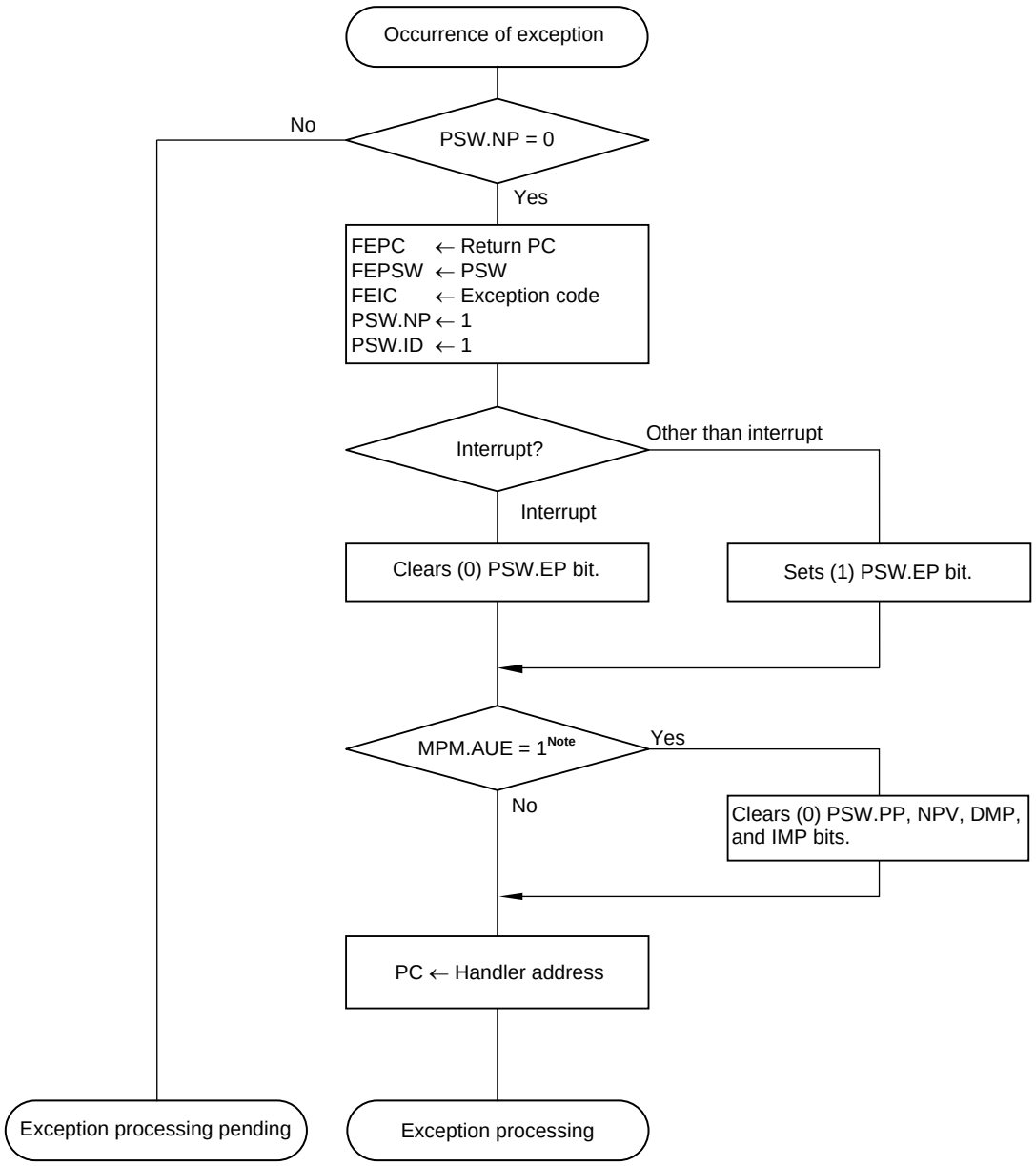
2. The PSW.PP, NPV, DMP, and IMP bits are always cleared (0) if an exception related to processor protection (PPI or TSI exception) occurs.

FEPC and FEPSW are used as status save registers. An FE level exception with acknowledgment conditions that has occurred is held pending while an other FE level exception is being processed (while the PSW.NP is 1). In this case, if the PSW.NP bit is cleared (0) by using the LDSR instruction, the FE-level exception with acknowledgment conditions which has been held pending is acknowledged.

Because only one pair of FEPC and FEPSW is available, the context must be saved in advance by a program before multiple exceptions are enabled.

The format of processing of an FE level exception with acknowledgment conditions is illustrated below.

Figure 6-7. Processing Format of FE Level Exception with Acknowledgment Conditions



Note If a processor protection exception (PPI or TSI) occurs, the PSW.PP, NPV, DMP, and IMP bits are always cleared (0), regardless of the status of MPM.AUE.

6.2.5 Special operations

(1) EP bit of PSW register

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

Depending on the status of the EP bit, the operation changes when the EIRET, FERET, or RETI instruction is executed. If the EP bit is cleared (0), the end of the exception processing routine is reported to the external interrupt controller. This function is necessary for correctly controlling the resources 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 (0).

(2) PP, NPV, DMP, and IMP bits of PSW register

If a processor protection exception is acknowledged, the PSW.PP, NPV, DMP, and IMP bits are unconditionally cleared (0). If an exception other than the processor protection exception is acknowledged, the operation differs depending on the setting of the MPM.AUE bit. If the MPM.AUE bit is set (1) (if the execution level auto transition function is enabled), the PSW.PP, NPV, DMP, and IMP bits are cleared (0). If the MPM.AUE bit is cleared (0) (if the execution level auto transition function is disabled), the value of the PSW.PP, NPV, DMP, and IMP bits is not updated but the previous value is retained.

(3) Coprocessor unusable exception

The generated opcode of the coprocessor unusable exception changes depending on the function specification of the product.

If an opcode defined as a coprocessor instruction is not implemented on the product or if its use is not enabled depending on the operation status, the coprocessor unusable exception (UCPOP) immediately occurs when an attempt to execute the coprocessor instruction is made.

For details, refer to **CHAPTER 7 COPROCESSOR UNUSABLE STATUS**.

(4) 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 (RIEX) occurs.

However, which of the following two types of operations each opcode is to perform may be defined by the product specification.

- 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.

(5) System call exception

For a system call exception, a table entry to be referenced is selected by the value of a vector specified by the opcode and the value of the SSCFG.SIZE bit, and the exception handler address is calculated in accordance with the contents of that table entry and the value of the SCBP register.

If table size n is specified by SSCFG.SIZE, for example, a table entry is selected as follows. Note that, where n < 255, table entry 0 is referenced from vector n+1 to 255.

Vector	Exception Code	Table Entry to Be Referenced
0	0000 8000H	Table entry 0
1	0000 8001H	Table entry 1
2	0000 8002H	Table entry 2
(omitted)	:	:
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
(omitted)	:	:
254	0000 80FEH	Table entry 0
255	0000 80FFH	Table entry 0

Caution Place an error processing routine at table entry 0 because it is also selected when a vector exceeding n specified by SCCFG.SIZE is specified.

(6) Reset

An operation which is the same as an exception is performed to initialize the CPU by reset. However, reset does not belong to any of EI level exception, 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 default value. 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.

6.3 Exception Management

The V850E2M CPU has the following functions to manage exceptions in order to prevent mutual interference between tasks during multi-programming.

- Exception synchronization function when exception is acknowledged or when execution returns
- Exception synchronization instruction (SYNCE)
- Function to check pending exception
- Function to cancel pending exception

The V850E2M CPU defines imprecise exceptions that have a delay time until the exception processing is started after the cause of the exception has been generated, and asynchronous exceptions that are generated asynchronously with instruction execution though they are linked with a task. The imprecise exception and asynchronous exception that are generated for a specific task must be processed when a task is changed or terminated, so that the next task is not executed without the exception being processed.

The V850E2M CPU has an exception management function to wait for all exceptions caused by a task before the task is changed or terminated, so that the exceptions are sequentially processed. This prevents the influence of illegal processing of a certain task from reaching the other tasks. It also prevents termination processing of a task from being completed without the exceptions being processed.

Exceptions must be managed in the following cases.

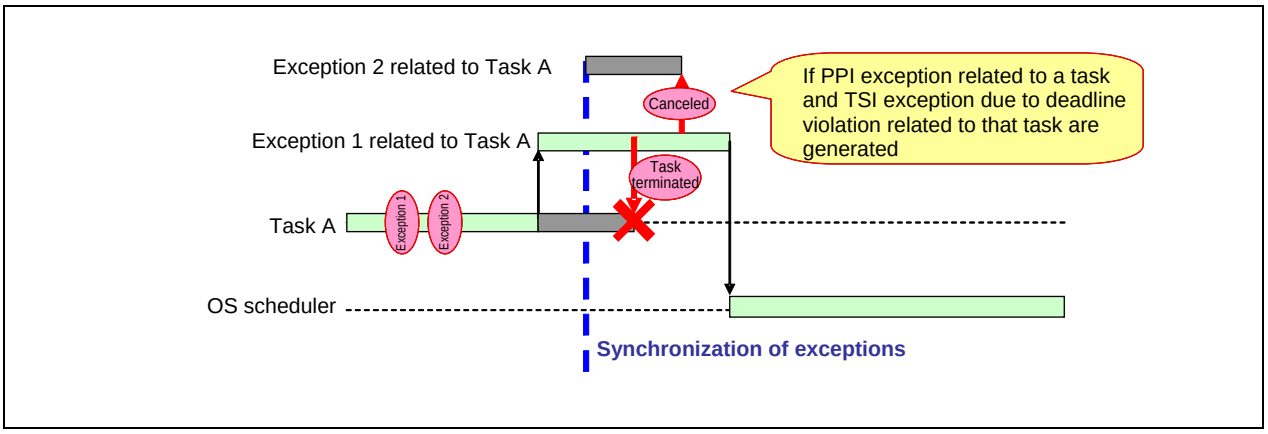
(1) To abort and terminate a task causing exceptions

If two or more exceptions are simultaneously generated for a task, care must be exercised in processing these exceptions. In accordance with the exception priority, one of the exceptions is acknowledged first and processed. While this exception is being processed, the other exception that is held pending must be managed. Consequently, exceptions that are generated, caused by a task, must be synchronized until the task is terminated.

For example, if a floating-point exception (FPI exception) and a peripheral device protection exception (PPI exception) are generated at the same time, peripheral device protection violation is acknowledged first, according to the exception priority, and the other floating-point exception is held pending. The peripheral device protection exception may be a relatively serious error and a situation where its task is forcibly terminated is assumed. In this case, if execution returns from the peripheral device protection exception simply because the task has been terminated, the pending floating-point exception is acknowledged as soon as execution of another task is started. However, the task that has caused this floating-point exception has already been terminated and deleted from the management table of the OS. Consequently, it may be mistakenly recognized that another task has caused the exception.

If a task is terminated by exception processing related to that task, therefore, exception synchronization processing is performed and then the other exception related to the task which has been held pending must be canceled.

Figure 6-8. To Terminate Task Causing Exception

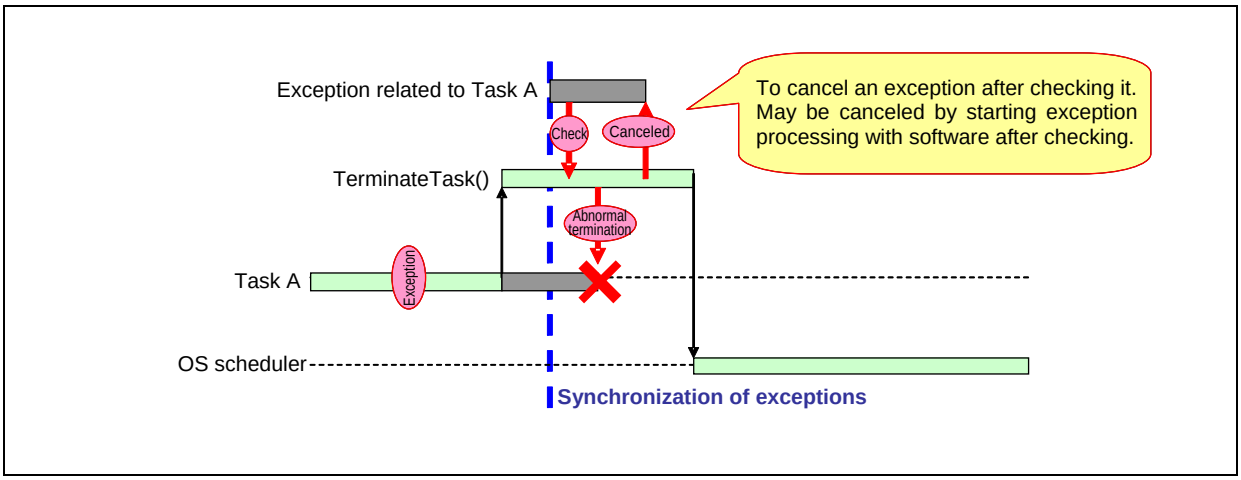


(2) If exception occurs immediately before task is terminated

If an imprecise exception or asynchronous exception occurs immediately before a task is terminated, the termination processing is started before the exception is acknowledged. As a result, it may be wrongly recognized that the task, which should be judged to have caused an abnormality, has been correctly terminated.

To terminate a task, therefore, exception synchronization processing must be performed to process the exception first or the other exception that has been held pending must be canceled.

Figure 6-9. If Exception Occurs Immediately Before Task Is Terminated



6.3.1 Synchronizing exception when exception is acknowledged or when execution returns

When the CPU acknowledges an EI level or FE level exception or when it returns from an EI level or FE level exception, it waits for the next imprecise exception or asynchronous exception, and acknowledges the exceptions that can be acknowledged in accordance with their priority.

- Peripheral device protection exception (PPI exception)
- Floating-point exception (FPI exception)
- Timing supervision exception in run-time supervision mode (TSI exception)

These exceptions are linked to a specific task. The CPU waits for and synchronizes the exceptions when it acknowledges an EI level or FE level exception that triggers switching the task or when it returns from the EI level or FE level exception. As a result of the CPU's synchronizing the exceptions, the software can easily manage the exceptions.

6.3.2 Exception synchronization instruction

The peripheral device protection exception and floating-point exception are synchronized by the SYNCE instruction. To acknowledge these imprecise exceptions at any point of time, follow this 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 may be masked by the acknowledgment condition set in (1) and the exception may 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.

6.3.3 Checking and cancelling 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 may 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 processing, 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 may 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 product specification. Refer to the manuals of each product and hardware.
(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.

Exception	Cause	Cancelling Bit	Remark
FPI exception	FPU instruction	FPU bank FPEC register FPIVD bit	If this bit is cleared, disabling the succeeding FPU instruction is canceled.
PPI exception	Instruction involving memory access	Peripheral I/O area PPEC register PPVD bit	If this bit is cleared, disabling the succeeding access instruction is canceled.
TSI exception	Timing supervision	Peripheral I/O area TSUCFGn register (n = 0 to 5) VS bit	–

6.4 Exception Handler Address Switching Function

The V850E2M CPU can use the exception handler address switching function to change the exception handler address. The exception handler address where processing is passed after an exception is determined by the value set by this exception handler switching function.

The exception handler address switching function uses the following two banks among the system register banks. For details, see **2.4 CPU Function Bank/Exception Handler Address Switching Function Banks**.

The exception handler address are divided into the following three types.

- CPU initialization (RESET)
- EI level maskable interrupts (INT0 to INT255)
- All other types of exceptions

6.4.1 Determining exception handler addresses

The current exception handler address is indicated by the register assigned to exception handler switching function bank 1 (ESWH1).

(1) Start address for CPU initialization (RESET)

This address is indicated by the EH_RESET register.

(2) EI level maskable interrupt (INT0 to INT255)

This interrupt is indicated by the EH_BASE register and the RINT bit in the EH_CFG register. The settings in this register can be changed by software.

When the RINT bit is cleared (0), 256 different exception handler addresses that include the EH_BASE register and offset addresses are used as the exception handler addresses for INT0 to INT255.

When the RINT bit is set (1), the exception handler addresses for INT0 to INT255 are reduced and a single exception handler address specified by adding 0080H to EH_BASE is used.

Once an 4096-byte address range has been reserved for INT0 to INT255 exception handler addresses, reduction to 16 bytes can be performed by setting (1) the RINT bit.

Caution	**Even when using a reduced exception handler address, exception codes can be used to distinguish among exception causes for INT0 to INT255.**

(3) Exception handler addresses for other types of exceptions

These addresses are indicated by the EH_BASE register. Offset addresses for each exception are added to addresses indicated in the EH_BASE register to create the address that is used as the particular exception handler address. The settings in this register can be changed by software.

6.4.2 Purpose of exception handler address switching

Exception handler address switching setting is made by software after startup.

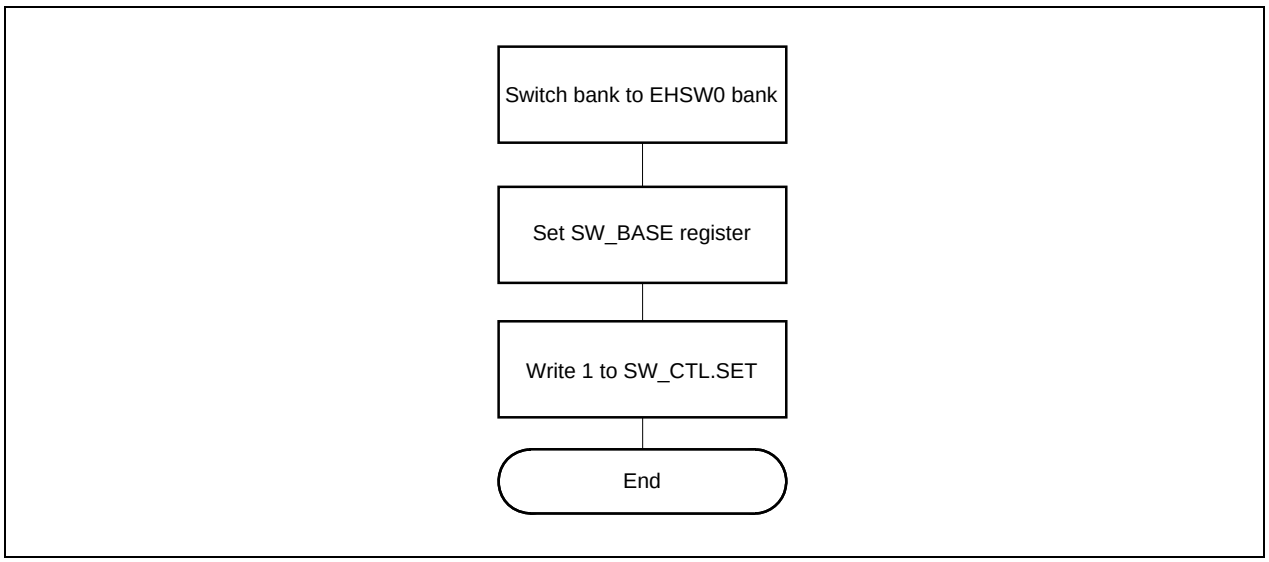
(1) Switch according to software

After system startup, addresses can be switched to set up a temporary consistent instruction address area when instructions are not consistent at addresses near the exception handler address, for various reasons (such as a flash memory rewrite).

6.4.3 Settings for exception handler address switching function

(1) Switch according to software

The following methods can be used to change the exception handler address via the following steps while the CPU is operating.



When switching exception handler addresses, try to prevent exceptions from occurring between when the switch is started and ended (or, if an exception does occur, try to prevent any problems it would cause). For example, this can be done by prohibiting exceptions, by using control to prevent system-wide exceptions from occurring, or by assigning only programs that operate normally to the exception handler addresses before and after the switch.

Caution The CPU initialization (RESET) start address cannot be changed by software.

CHAPTER 7 COPROCESSOR UNUSABLE STATUS

The V850E2M CPU defines a function limited to a specific application as a coprocessor. The V850E2M CPU can implement a floating-point operation function (FPU) as a coprocessor.

7.1 Coprocessor Unusable Exception

A coprocessor unusable exception (UCPOP) occurs in the following cases if an opcode defined as a coprocessor instruction is to be executed.

- If the coprocessor function is not defined
- If the coprocessor function is not implemented for the product
- If the coprocessor function is disabled by a function of the product

The coprocessor unusable exception is assigned an exception code for each coprocessor function. The correspondence between the coprocessor functions and exception causes is shown in the following table.

Coprocessor Function	Exception to Occur	Exception Code
Single-precision FPU extension function	UCPOP0	530H
Double-precision FPU extension function	UCPOP1	531H
Undefined	UCPOP0 to UCPOP7[Note]	530H to 537H[Note]

Note Which exception occur for an undefined opcode is defined by the product specification. For details, refer to the manuals of each product.

7.2 System Registers

System registers are defined as a part of some coprocessor functions. The operation of the system register of the corresponding coprocessor function is undefined in the following cases because of the architecture.

- If the coprocessor function is not implemented on the product
- If the coprocessor function is disabled by a function of the product

CHAPTER 8 RESET

8.1 Status of Registers After Reset

If a reset signal is input by a method defined by the product specification, the program registers and system registers are placed in the status shown in Table 8-1, and program execution is started. Initialize the contents of each register to an appropriate value in the program.

Table 8-1. Status of Registers After Reset

Register		Status After Reset (Initial Value)
Program registers	General-purpose register (r0)	00000000H (fixed)
	General-purpose registers (r1 to r31)	Undefined
	Program counter (PC)	00000000H
System registers	EIPC – Status save register when acknowledging EI level exception	Undefined
	EIPSW – Status save register when acknowledging EI level exception	00000020H
	FEPC – Status save register when acknowledging FE level exception	Undefined
	FEPSW – Status save register when acknowledging FE level exception	00000020H
	ECR – Exception cause	00000000H
	PSW – Program status word	00000020H
	SCCFG – SYSCAL operation configuration	Undefined
	SCBP – SYSCALL base pointer	Undefined
	EIIC – EI level exception cause	00000000H
	FEIC – FE level exception cause	00000000H
	DBIC[Note] – DB level interrupt cause	00000000H
	CTPC – CALLT execution status save register	Undefined
	CTPSW – CALLT execution status save register	00000020H
	CTBP – CALLT base pointer	Undefined
	EIWR – EI level exception working register	
	FEWR – FE level exception working register	
	BSEL – Register bank selection	00000000H

Note The DBIC register is used by the debug function for development tools.

8.2 Start

The CPU executes a reset to start execution of a program from the reset address specified by the exception handler address switching function.

INT exceptions are not acknowledged immediately after a reset. If the program will use INT exceptions, be sure to clear (0) the PSW.ID bit.

PART 3 PROCESSOR PROTECTION FUNCTION

CHAPTER 1 OVERVIEW

The V850E2M CPU conforms to the V850E2v3 Architecture and supplies a processor protection function that detects or prevents illegal use of system resources and inappropriate possession of the CPU execution time by non-trusted programs or program loops, thereby enabling a highly-reliable system to be set up.

Caution The available processor protection functions vary depending on the product.

1.1 Features

(1) Resource access control

The V850E2M CPU supplies a function to control accesses to the following four types of resources.

- **System register protection**

  Damage to the system registers by a non-trusted program can be prevented.

- **Memory protection**

  Up to five instruction or constant protection areas and six data protection areas can be placed on the address space. As a result, execution or data manipulation by the user program that is not allowed is detected, so that illegal execution or data manipulation can be prevented. Each area is specified using both upper-limit and lower-limit addresses, so that the address space can be efficiently used with fine detail.

- **Peripheral device protection**

  An illegal access defined for each system to a peripheral device can be detected and prevented.

- **Timing supervision**

  Inappropriate CPU time possession by a non-trusted program can be prevented, and resources and time of disabling interrupts can be managed.

(2) Management by execution level

The V850E2M CPU has more than one status bit to control accesses to the resources, and combinations of these status bits are defined as execution levels.

The user can control accesses in accordance with a situation by selecting an execution level in accordance with the situation, or by using an execution level auto transition function that automatically changes when some special instructions are executed if an exception occurs or when execution returns from an exception.

(3) Selectable and scalable specification

To use the execution level auto transition function, the OS (and programs similar to it), common library, and user task must comply with a specific program model.

The V850E2M CPU employs scalable specification that allows processor protection to be used even when the execution level auto transition function is not selected. Therefore, the processor protection can be easily introduced to the existing software resources. Moreover, the operation can be performed, in a status without a processor protection function as usual.

CHAPTER 2 REGISTER SET

2.1 System Register Bank

Table 2-1 shows the system register banks related to the processor protection function.

The processor protection setting bank, the processor protection violation bank and the software paging bank are selected by setting 00001001H, 00001000H and 00001010H to a system register (BSEL) by using an LDSR instruction.

System registers numbered 28 to 31 are shared by the banks, and the EIWR, FEWR, DBWR[Note], and BSEL registers of the CPU function bank are referenced regardless of the set value of the BSEL register.

Note The DBWR register is used by the debug function for development tools.

- Processor protection violation bank
  (Group number: 10H, Bank number: 00H, Abbreviation: MPV/PROT00 bank, Stores processor protection violation registers.)
- Processor protection setting bank
  (Group number: 10H, Bank number: 01H, Abbreviation: MPU/PROT01 bank, Stores processor protection setting registers.)
- Software paging bank
  (Group number: 10H, Bank number: 10H, Abbreviation: PROT10 bank, Stores processor protection setting/violation registers.)

The following system register of the CPU function bank is used as a register related to the processor protection function.

- PSW register

Table 2-1. System Register Bank

Group	Processor Protection Function (10H)					
Bank	Processor protection violation (00H)			Processor protection setting (01H)		Software paging (10H)
Bank label	MPV, PROT00			MPU, PROT01		PROT10
Register No.	Name	Function		Name	Function	Name
0	**VSECR**	**System register protection violation cause**		MPM	Setting of processor protection operation mode	MPM
1	**VSTID**	**System register protection violation task identifier**		MPC	Specification of processor protection command	MPC
2	**VSADR**	**System register protection violation address**		TID	Task identifier	TID
3	Reserved for function expansion			Reserved for function extension		VMECR
4	VMECR	Memory protection violation cause				VMTID
5	VMTID	Memory protection violation task identifier				VMADR
6	VMADR	Memory protection violation address		**IPA0L**	**Instruction/constant protection area 0 lower-limit address**	IPA0L
7	Reserved for function expansion			**IPA0U**	**Instruction/constant protection area 0 upper-limit address**	IPA0U
8				**IPA1L**	**Instruction/constant protection area 1 lower-limit address**	IPA1L
9				**IPA1U**	**Instruction/constant protection area 1 upper-limit address**	IPA1U
10				**IPA2L**	**Instruction/constant protection area 2 lower-limit address**	IPA2L
11				**IPA2U**	**Instruction/constant protection area 2 upper-limit address**	IPA2U
12				**IPA3L**	**Instruction/constant protection area 3 lower-limit address**	IPA3L
13				**IPA3U**	**Instruction/constant protection area 3 upper-limit address**	IPA3U
14				**IPA4L**	**Instruction/constant protection area 4 lower-limit address**	IPA4L
15				**IPA4U**	**Instruction/constant protection area 4 upper-limit address**	IPA4U
16				**DPA0L**	**Data protection area 0 lower-limit address (for stack)**	DPA0L
17				**DPA0U**	**Data protection area 0 upper-limit address (for stack)**	DPA0U
18				**DPA1L**	**Data protection area 1 lower-limit address**	DPA1L
19				**DPA1U**	**Data protection area 1 upper-limit address**	DPA1U
20				**DPA2L**	**Data protection area 2 lower-limit address**	DPA2L
21				**DPA2U**	**Data protection area 2 upper-limit address**	DPA2U
22				**DPA3L**	**Data protection area 3 lower-limit address**	DPA3L
23				**DPA3U**	**Data protection area 3 upper-limit address**	DPA3U
24	**MCA**	**Memory protection setting check address**		**DPA4L**	**Data protection area 4 lower-limit address**	DPA4L
25	**MCS**	**Memory protection setting check size**		**DPA4U**	**Data protection area 4 upper-limit address**	DPA4U
26	**MCC**	**Memory protection setting check command**		**DPA5L**	**Data protection area 5 lower-limit address (2^n specification only)**	DPA5L
27	**MCR**	**Memory protection setting check result**		**DPA5U**	**Data protection area 5 upper-limit address (2^n specification only)**	DPA5U
28	EIWR	Work register for EI level exception				
29	FEWR	Work register for FE level exception				
30	DBWR[Note]	Work register for DB level exception				
31	BSEL	Selection of register bank				

Note The DBWR register is used by the debug function for development tools.

2.2 System Registers

(1) Processor protection setting registers

The processor protection setting registers select a processor protection mode and specifies a subject to protection. A system register can be read or written by the LDSR or STSR instruction, by specifying one of the system register numbers shown in this table.

Table 2-2 lists the processor protection setting registers.

Table 2-2. Processor Protection Setting Registers

System Register Number	Name	Function	Able to Specify Operands?		System Register Protection[Note]
			LDSR	STSR	
0	MPM	Setting of processor protection operation mode	√	√	√
1	MPC	Specification of processor protection command	√	√	√
2	TID	Task identifier	√	√	√
3 to 5	(Reserved for future function expansion (Operation is not guaranteed if these registers are accessed.))		×	×	√
6	IPA0L	Instruction/constant protection area 0 lower-limit address	√	√	√
7	IPA0U	Instruction/constant protection area 0 upper-limit address	√	√	√
8	IPA1L	Instruction/constant protection area 1 lower-limit address	√	√	√
9	IPA1U	Instruction/constant protection area 1 upper-limit address	√	√	√
10	IPA2L	Instruction/constant protection area 2 lower-limit address	√	√	√
11	IPA2U	Instruction/constant protection area 2 upper-limit address	√	√	√
12	IPA3L	Instruction/constant protection area 3 lower-limit address	√	√	√
13	IPA3U	Instruction/constant protection area 3 upper-limit address	√	√	√
14	IPA4L	Instruction/constant protection area 4 lower-limit address	√	√	√
15	IPA4U	Instruction/constant protection area 4 upper-limit address	√	√	√
16	DPA0L	Data protection area 0 lower-limit address	√	√	√
17	DPA0U	Data protection area 0 upper-limit address	√	√	√
18	DPA1L	Data protection area 1 lower-limit address	√	√	√
19	DPA1U	Data protection area 1 upper-limit address	√	√	√
20	DPA2L	Data protection area 2 lower-limit address	√	√	√
21	DPA2U	Data protection area 2 upper-limit address	√	√	√
22	DPA3L	Data protection area 3 lower-limit address	√	√	√
23	DPA3U	Data protection area 3 upper-limit address	√	√	√
24	DPA4L	Data protection area 4 lower-limit address	√	√	√
25	DPA4U	Data protection area 4 upper-limit address	√	√	√
26	DPA5L	Data protection area 5 lower-limit address	√	√	√
27	DPA5U	Data protection area 5 upper-limit address	√	√	√

Note Refer to **CHAPTER 5 SYSTEM REGISTER PROTECTION**.

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

(2) Processor protection violation registers

The processor protection violation registers (memory protection violation report registers) report a violation causes, violation task identifiers, and violation addresses. A system register can be read or written by the LDSR or STSR instruction, by specifying one of the system register numbers shown in this table.

Table 2-3 lists the processor protection violation registers.

Table 2-3. Processor Protection Violation Registers

System Register Number	Name	Function	Able to Specify Operands?		System Register Protection[Note]
			LDSR	STSR	
0	VSECR	System register protection violation cause	√	√	√
1	VSTID	System register protection violation task identifier	√	√	√
2	VSADR	System register protection violation address	√	√	√
3	(Reserved for future function expansion (Operation is not guaranteed if this register is accessed.))		×	×	√
4	VMECR	Memory protection violation cause	√	√	√
5	VMTID	Memory protection violation task identifier	√	√	√
6	VMADR	Memory protection violation address	√	√	√
7 to 23	(Reserved for future function expansion (Operation is not guaranteed if these registers are accessed.))		×	×	√
24	MCA	Memory protection setting check address	√	√	√
25	MCS	Memory protection setting check size	√	√	√
26	MCC	Memory protection setting check command	√	√	√
27	MCR	Memory protection setting check result	√	√	√

Note Refer to **CHAPTER 5 SYSTEM REGISTER PROTECTION**.

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

(3) Software paging registers

The software paging registers realize software paging operation using memory protection. These registers are maps of the processor protection setting registers and processor protection violation registers.

With the V850E2M CPU, it is a general rule to set a fixed memory protection area for each task to protect the memory. However, an operation that sequentially changes protection setting by an exception program that is started by a processor protection exception if a memory access is requested when the program accesses the memory is assumed to provide for a case where the number of memory protection areas runs short in an extremely large software system. This operation method is called software paging, and a bank consisting of system registers suitable for this operation is defined as a software paging bank.

By using this bank, the switching of the bank can be reduced to once during a processor protection exception processing. As a result, the software overhead can be reduced by decreasing the number of general-purpose registers necessary for software paging and by reducing the execution cycles necessary for saving or restoring the context.

Table 2-4 lists the software paging registers. A system register can be read or written by the LDSR or STSR instruction, by specifying one of the system register numbers shown in the following table.

Table 2-4. Software Paging Registers

System Register Number	Name	Function	Able to Specify Operands?		System Register Protection[Note]
			LDSR	STSR	
0	MPM	Setting of processor protection operation mode	√	√	√
1	MPC	Specification of processor protection command	√	√	√
2	TID	Task identifier	√	√	√
3	VMECR	Memory protection violation cause	√	√	√
4	VMTID	Memory protection violation task identifier	√	√	√
5	VMADR	Memory protection violation address	√	√	√
6	IPA0L	Instruction/constant protection area 0 lower-limit address	√	√	√
7	IPA0U	Instruction/constant protection area 0 upper-limit address	√	√	√
8	IPA1L	Instruction/constant protection area 1 lower-limit address	√	√	√
9	IPA1U	Instruction/constant protection area 1 upper-limit address	√	√	√
10	IPA2L	Instruction/constant protection area 2 lower-limit address	√	√	√
11	IPA2U	Instruction/constant protection area 2 upper-limit address	√	√	√
12	IPA3L	Instruction/constant protection area 3 lower-limit address	√	√	√
13	IPA3U	Instruction/constant protection area 3 upper-limit address	√	√	√
14	IPA4L	Instruction/constant protection area 4 lower-limit address	√	√	√
15	IPA4U	Instruction/constant protection area 4 upper-limit address	√	√	√
16	DPA0L	Data protection area 0 lower-limit address	√	√	√
17	DPA0U	Data protection area 0 upper-limit address	√	√	√
18	DPA1L	Data protection area 1 lower-limit address	√	√	√
19	DPA1U	Data protection area 1 upper-limit address	√	√	√
20	DPA2L	Data protection area 2 lower-limit address	√	√	√
21	DPA2U	Data protection area 2 upper-limit address	√	√	√
22	DPA3L	Data protection area 3 lower-limit address	√	√	√
23	DPA3U	Data protection area 3 upper-limit address	√	√	√
24	DPA4L	Data protection area 4 lower-limit address	√	√	√
25	DPA4U	Data protection area 4 upper-limit address	√	√	√
26	DPA5L	Data protection area 5 lower-limit address	√	√	√
27	DPA5U	Data protection area 5 upper-limit address	√	√	√

Note Refer to **CHAPTER 5 SYSTEM REGISTER PROTECTION**.

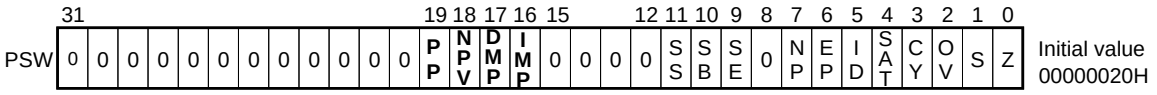
Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

2.2.1 PSW – Program Status Word

Bits related to the processor protection function are assigned to bits 16 to 19 of the PSW register in the CPU function bank.

Figure 2-1. Memory Protection Operation Status Bits in PSW



Bit Position	Bit Name	Description
19	PP	Status bit of peripheral device protection. This bit indicates whether the CPU trusts an access by the program currently under execution to a peripheral device. 0: T state (CPU trusts the access to the peripheral device.) (initial value) 1: NT state (CPU does not trust the access to the peripheral device.) The peripheral device protection function limits accesses if the PP bit indicates the T state. If it indicates the NT state, it strictly limits accesses.
18	NPV	Status bit of system register protection. This bit indicates whether the CPU trusts an access by the program currently under execution to the system register. 0: T state (CPU trusts the access to the system register.) (initial value) 1: NT state (CPU does not trust the access to the system register.) The system register protection function does not limit accesses if the NPV bit indicates the T state. If it indicates the NT state, it limits the accesses.
17	DMP	Status bit of memory protection for data access (data area). This bit indicates whether the CPU trusts a data access by the program currently under execution. (initial value: 0) 0: T state (CPU trusts data access.) (initial value) 1: NT state (CPU does not trust data access.) The memory protection function does not limit data access if the DMP bit indicates the T state. If it indicates the NT state, it limits data access.
16	IMP	Status bit of memory protection for program area This bit indicates whether the CPU trusts an access by the program currently under execution to a program area. (initial value: 0) 0: T state (CPU trusts the access to the program area.) (initial value) 1: NT state (CPU does not trust the access to the program area.) The memory protection function does not limit an access to the program area if the IMP bit indicates the T state. If it indicates the NT state, it limits an access to the program area.

2.2.2 MPM – Setting of processor protection operation mode

The processor protection mode register determines the basic operation status of the processor protection function. Usually, a value is set to this register only once when the system is started and is not changed arbitrarily as the program is executed.

Be sure to set bits 31 to 3 to “0”.

(1/2)

	31																										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	0	0	0	0	0	SPS	AUE	MPE	Initial value 00000000H

Bit Position	Bit Name	Description
2	SPS	This bit enables or disables stack inspection of the memory protection function (initial value: 0). By setting the SPS bit (1), a range accessible by an instruction that executes sp indirect access can be limited to an area specified by the DPA0L/DPA0U register. Depending on the set value of the SPS bit, the DPAnL.S and IPAnL.S bits take the following value.
1	AUE	This bit automatically changes the execution level if an exception occurs, and sets a function to update the PSW.PP, NPV, DMP, and IMP bits (initial value: 0). 0: Does not automatically update. The value of the PSW.PP, NPV, DMP, and IMP bits are not changed even if an exception occurs[Notes 2, 3]. 1: Automatically updates. The value of the PSW.PP, NPV, DMP, and IMP bits is cleared to 0 if an exception occurs.

	SPS bit = 0	SPS bit = 1
IPAnL.S bit	1	0
DPA0L.S bit[Note 1]	1	1
DPAmL.S bit	1	0

Remark n = 0 to 4, m = 1 to 5

Notes 1. The area specified by the DPA0L/DPA0U register can always be accessed in sp indirect access mode.

2. The exception that changes the execution level to the DB level and the processor protection violation exceptions (MDP, MIP, PPI, and TSI exceptions) are excluded. When these exceptions are acknowledged, the PSW.PP, NPV, DMP, and IMP bits are always updated to 0 under any circumstances.

3. The PSW.NPV bit is fixed to 0 and cannot be changed if the MPM.AUE bit is cleared (0).

(2/2)

Bit Position	Bit Name	Description
0	MPE	This bit enables or disables the operation of the processor protection function (initial value: 0). 0: Processor protection function disabled • PSW.PP bit is fixed to 0. The peripheral device protection function limits accesses and detects only violation of a special peripheral device. The PPI exception does not occur[Note]. • PSW.NPV bit is fixed to 0. The system register protection function is disabled and access to all the system registers is enabled. • PSW.DMP and IMP bits are fixed to 0. The memory protection function is disabled and all memory accesses are enabled. The MIP and MDP exceptions does not occur. • TSCCFGn.TSCCFGnACT bit (n = 0 to 5) is fixed to 0. The timing supervision function is disabled. The TSI exception does not occur[Note]. 1: Processor protection function enabled • Updating the PSW.PP bit is enabled. The peripheral device protection function strictly limits accesses and detects violation in accordance with the setting. The PPI exception may occur. • Updating the PSW.NPV bit is enabled. The system register protection function is enabled and detects violation in accordance with the setting. • Updating the PSW.DMP and IMP bits is enabled. The memory protection function is enabled and detects violation in accordance with setting. The MIP and MDP exceptions may occur. • Updating the TSCCFGn.TSCCFGnACT bit (n = 0 to 5) is enabled. The timing supervision function is enabled and operates in accordance with setting. The TSI exception may occur.

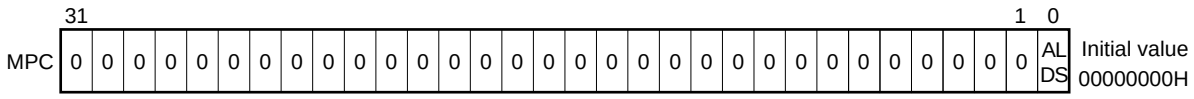
Note However, the PPI or TSI exception that has been detected before the MPE bit is cleared to 0 may be held pending. In this case, the PPI or TSI exception may be acknowledged when the PSW.NP bit is cleared to 0 even if the MPE bit is 0.

Remark For details on the PP, NPV, IMP, and DMP bits, refer to **2.2.1 PSW – Program Status Word**.

2.2.3 MPC – Specification of processor protection command

This register consists of bits that are used for special operation of the processor protection function.

Be sure to set bits 31 to 1 to “0”.

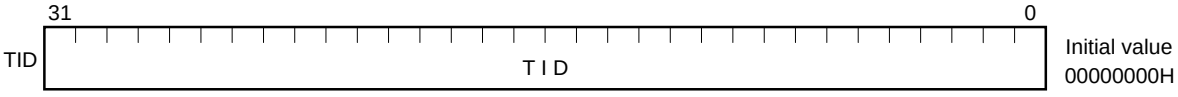


Bit Position	Bit Name	Meaning
0	ALDS	When this bit is set (1), the protection operation bits of the entire memory protection area is immediately cleared (0). The target bits are as follows. IPAmL.E bit (m = 0 to 4) DPAnL.E bit (n = 0 to 5) When all these bits have been cleared (0), the ALDS bit is also cleared (0).

Caution Even when the bits cleared (0) by the function of the ALDS bit are set (1) by the LDSR instruction immediately after the LDSR instruction that has set the ALDS bit (1), the result in accordance with the execution sequence of the instruction can be obtained (the target bits are set (1)).

2.2.4 TID – Task identifier

This register is used to set an identifier of a task under execution. The TID register setting is not automatically changed. Be sure to set an appropriate value to the TID register by program when switching the task.



Bit Position	Bit Name	Meaning
31 to 0	TID	These bits set an identifier of a task under execution.

2.2.5 Other system registers

For details on the other system registers, refer to **CHAPTER 5 SYSTEM REGISTER PROTECTION, CHAPTER 6 MEMORY PROTECTION**, and **CHAPTER 11 SPECIAL FUNCTION**.

CHAPTER 3 OPERATION SETTING

To use the processor protection function, an operation related to overall processor protection must be first set. Switch the system register bank to the processor protection setting bank (group number: 10H, bank number: 01H) and set an appropriate value to the MPM register.

3.1 Starting Use of Processor Protection Function

To enable the processor protection function, first the MPM.MPE bit must be set (1). If the MPE bit is cleared (0), the PSW.PP, NPV, DMP, and IMP bits, and TSCCFGn.TSCCFGnACT bit (n = 0 to 5) are fixed to 0, and each function of the processor protection function does not operate. When the MPM.MPE bit is set (1), use of the processor protection function is started as follows.

- The PSW.PP, NPV, DMP, and IMP bits can be updated.
  (The system register protection function, memory protection function, and peripheral device protection function can be used.)
- The TSCCFGnACT bit of the setting register (TSCCFGn) of each counter of the timing supervision function can be updated.(The timing supervision function can be used.)

3.2 Setting of Execution Level Auto Transition Function

A function to automatically change the execution level is enabled by setting the MPM.AUE bit (1). To operate a system with a program model that automatically changes the execution level, be sure to set the AUE bit before using the processor protection function.

For details, refer to **CHAPTER 4 EXECUTION LEVEL**.

3.3 Stopping Use of Processor Protection Function

To disable the processor protection function that has been once enabled, clear (0) the MPM.MPE bit. As a result, the PSW.PP, NPV, DMP, and IMP bits, and TSCCFGn.TSCCFGnACT bit (n = 0 to 5) are fixed to 0, and use of the processor protection function is stopped as follows.

- The PSW.PP, NPV, DMP, and IMP bits are fixed to 0.
  (The system register protection function, memory protection function, and peripheral device protection function cannot be used.)
- The TSCCFGnACT bit of the setting register (TSCCFGn) of each counter of the timing supervision function is fixed to 0.(The timing supervision function cannot be used.)

Caution	**When use of the processor protection function has been stopped, some processor protection exceptions (PPI and TSI exceptions) that have been detected before the MPE bit is cleared to 0 may be held pending. In this case, these exceptions are acknowledged when the PSW.NP bit is cleared to 0, even after the MPE bit has been changed to 0.**

CHAPTER 4 EXECUTION LEVEL

The V850E2M CPU indicates the reliability status of the program currently under execution and a right of access to the resources of the program by controlling the following 4 bits of the PSW (Program Status Word). These bits are called protection bits, and specific combinations of these bits are called execution levels.

- PP bit:

  Indicates whether the CPU trusts an access by the program under execution to the peripheral device.
- NPV bit:

  Indicates whether the CPU trusts an access by the program under execution to the system registers.
- DMP bit:

  Indicates whether the CPU trusts data access by the program under execution.
- IMP bit:

  Indicates whether the CPU trusts an access by the program under execution to the program area.

4.1 Nature of Program

Programs under execution are classified into “trusted programs” and “non-trusted programs” according to their design quality. Generally, the “trusted programs” are programs that do not pose any threat to systems such as OS (and programs similar to it) and device derivers. The “non-trusted programs” have not yet been confirmed that they do not pose any threat to systems such as user programs under development and programs of third parties.

For each of the following four protected subjects; system registers, data area, program area, and peripheral devices, PSW.PP, NPV, DMP, and IMP bits are defined as information by which the hardware can distinguish between the operation of a trusted program and the operation of a non-trusted program. These bits have the following meaning for the related resources, and are set to an appropriate value by the OS (and programs similar to it) before execution of each program is started.

Status of Protection Bit	Status Name	Program Quality
0	T state	Trusted (trusted program)
1	NT state	Not trusted (non-trusted program)

4.2 Protection Bits on PSW

Protection bits (PP, NPV, DMP, and IMP bits) that indicate the reliability status of the program under execution on the resources subject to processor protection are placed on PSW. Bits 19, 18, 17, and 16 of the PSW are defined as PP, NPV, DMP, and IMP bits. Set these bits appropriately when using the processor protection function.

Caution The PSW.PP, NPV, DMP, and IMP bits are fixed to 0 when the MPM.MPE bit is 0. Because these bits are subject to system register protection, they cannot be written when the NPV bit is 1.

4.2.1 T state (trusted state)

Set 0 to the protection bits if the operation of the program under execution on the resource corresponding to each bit can be fully trusted and if the program does not perform an illegal operation. The state in which 0 is set to each of the protection bits is considered as a state in which operation on the resource corresponding to the bit is “trusted” and is called a T state.

Usually, when a program is in the T state, no violation is detected and the program performs a privileged operation.

Cautions 1. The peripheral device protection function detects violation of an operation on only special peripheral devices even in the T state.
2. The concept of trusted state does not apply to the timing supervision function.

4.2.2 NT state (non-trusted state)

Set 1 to each of the protection bits if an operation by the program under execution on the resource corresponding to the bit cannot be trusted and if the program may perform an illegal operation. A state in which each bit is set to 1 is considered as a state in which the operation on the resource corresponding to the bit “cannot be trusted” and is called an NT state.

If a program is in the NT state, violation is detected in accordance with setting and, in some cases, an exception occurs.

4.3 Definition of Execution Level

The V850E2M CPU assumes that some combinations of the statuses of the PSW.PP, NPV, DMP, and IMP bits which are typically used are defined and used as execution levels. Use with any combinations other than these is possible but not recommended.

Table 4-1 shows the execution levels and examples of their use.

Table 4-1. Execution Level

Execution Level	PP Bit (peripheral device protection)	NPV Bit (system register protection)	DMP Bit (memory protection data area)	IMP Bit (memory protection program area)	Example of Use of Execution Level
0	0	0	0	0	Exception handler, OS kernel, etc.
1	1	0	0	0	Device driver, etc.
2	1	0	1	1	Common library, etc.
3	1	1	1	1	User task

4.4 Transition of Execution Level

With the V850E2M CPU, the execution level is mainly changed in the following three ways.

- Execution of write instruction to system registers
- Occurrence of exception
- Execution of return instruction

While the MPM.MPE bit is cleared (0), the PSW.PP, NPV, DMP, and IMP bits are fixed to 0 in any of the above cases, and the execution level does not change from 0.

Caution For the V850E2M CPU, executing the CALLT instruction does not cause a transition of the execution level. Common subroutines called as a result of executing this instruction operate with the same processor protection status as the caller.

4.4.1 Transition by execution of write instruction to system register

The PSW.PP, NPV, DMP, and IMP bits can be rewritten by executing a write instruction (LDSR instruction) to a system register, so that the user can change the execution level to any level. The rewritten execution level becomes valid when the next instruction is executed.

Cautions 1. When the PSW.NPV bit is set (1), the PSW.PP, NPV, DMP, and IMP bits cannot be changed because the system registers are protected. Consequently, the execution level is not changed (refer to CHAPTER 5 SYSTEM REGISTER PROTECTION).

2. When the MPM.AUE bit is cleared (0), the PSW.NPV bit is fixed to 0 and cannot be changed.

3. If the setting of the PSW.IMP bit is changed by using the LDSR instruction, it may take several instructions to reflect the new setting. In this case, the new setting can be accurately reflected by performing a branch by executing the EIRET or FERET instruction.

4.4.2 Transition as result of occurrence of exception

The execution level auto transition function is enabled when the MPM.AUE bit is set (1). If an exception occurs in this status, the PSW.PP, NPV, DMP, and IMP bits are automatically cleared (0) and the execution level changes to level 0.

Caution If an exception that causes the execution level to change to the DB level or a memory protection exception (MDP, MIP, PPI, or TSI) occurs, the PSW.PP, NPV, DMP, and IMP bits are automatically cleared (0) regardless of the setting of the AUE bit.

4.4.3 Transition by execution of return instruction

When a return instruction (RETI, EIRET, FERET, or CTRET) to return execution from an exception or the CALLT instruction is executed, the value of the return PSW (EIPSW, FEPSW, or CTPSW) corresponding to the executed return instruction is copied to the PSW. If the MPM.MPE bit is set (1) at this time, the value of the bits corresponding to the PP, NPV, DMP, and IMP bits of the register that stores the value of the return PSW is copied to the PP, NPV, DMP, and IMP bits of the PSW. The value of the PSW.PP, NPV, DMP, and IMP bits do not change if the MPE bit is cleared (0). The NPV bit is always fixed to 0.

If an exception occurs, the value of the PSW before the exception occurs is saved for each exception level. If the value of the register that stores the value of the return PSW is not changed while the exception is processed, the PP, NPV, DMP, and IMP bits are restored to the status before occurrence of the exception when the exception return instruction is executed. When viewed from the program that is interrupted by the exception, therefore, the execution level does not change before and after the exception processing.

Cautions 1. The PSW.PP, NPV, DMP, and IMP bits are updated by the return instruction even when the execution level auto transition function is disabled.

2. The execution level also changes when execution is returned by the CTRET instruction from a subroutine that has been called by the CALLT instruction. However, the value of the PSW when the CALLT instruction has been executed is saved to CTPSW, and CTPSW is protected by the system register protection function. Therefore, the state will not change to the T state even if the user illegally changes the bit corresponding to the protected bit of CTPSW.

4.5 Program Model

By using the execution level auto transition function, two program models each having a different execution level management policy can be selected. Select a program model suitable for your system.

- **Program model that automatically changes execution level**

  The execution level auto transition function is enabled and the execution level is changed if an exception occurs. This program model is suitable for being used by an OS (and programs similar to it) or a program that is hierarchically managed.

- **Program model that always operates at constant execution level**

  The execution level auto transition function is disabled and the execution level is not changed even when an exception is acknowledged. The execution level is changed only by a special instruction executed by the user program, so that processor protection can be easily applied to the existing software resources.

4.6 Task Identifier

The V850E2M CPU is equipped with a register that identifies to which group the program currently being executed belongs when two or more different program groups are executed. This group of programs is called a task. An identifier for each task is defined as a task identifier and set to the TID register.

The processor protection function uses this task identifier as one piece of violation information when an exception occurs.

CHAPTER 5 SYSTEM REGISTER PROTECTION

The V850E2M CPU can control accesses to specific system registers to protect the setting of the system from being illegally changed by a program that is not trusted (non-trusted program).

If a system register protection violation occurs, the violation information is saved in the system registers of the MPU violation bank.

5.1 Register Set

Table 5-1 shows the system registers related to the system register protection function.

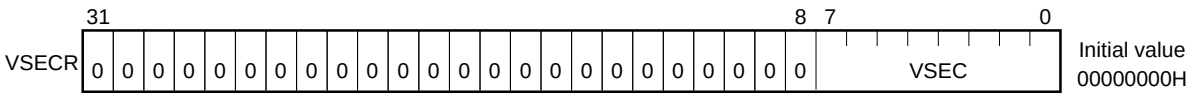
Table 5-1. System Register Bank

Group	Processor Protection Function (10H)					
Bank	Processor protection violation (00H)			Processor protection setting (01H)		Software paging (10H)
Bank label	MPV, PROT00			MPU, PROT01		PROT10
Register No.	Name	Function		Name	Function	Name
0	**VSECR**	**System register protection violation cause**		MPM	Setting of processor protection operation mode	MPM
1	**VSTID**	**System register protection violation task identifier**		MPC	Specification of processor protection command	MPC
2	**VSADR**	**System register protection violation address**		TID	Task identifier	TID
3	Reserved for function expansion			Reserved for function expansion		VMECR
4	VMECR	Memory protection violation cause				VMTID
5	VMTID	Memory protection violation task identifier				VMADR
6	VMADR	Memory protection violation address		**IPA0L**	**Instruction constant protection area 0 lower-limit address**	IPA0L
7	Reserved for function expansion			**IPA0U**	**Instruction constant protection area 0 upper-limit address**	IPA0U
8				**IPA1L**	**Instruction constant protection area 1 lower-limit address**	IPA1L
9				**IPA1U**	**Instruction constant protection area 1 upper-limit address**	IPA1U
10				**IPA2L**	**Instruction constant protection area 2 lower-limit address**	IPA2L
11				**IPA2U**	**Instruction constant protection area 2 upper-limit address**	IPA2U
12				**IPA3L**	**Instruction constant protection area 3 lower-limit address**	IPA3L
13				**IPA3U**	**Instruction constant protection area 3 upper-limit address**	IPA3U
14				**IPA4L**	**Instruction constant protection area 4 lower-limit address**	IPA4L
15				**IPA4U**	**Instruction constant protection area 4 upper-limit address**	IPA4U
16				**DPA0L**	**Data protection area 0 lower-limit address (for stack)**	DPA0L
17				**DPA0U**	**Data protection area 0 upper-limit address (for stack)**	DPA0U
18				**DPA1L**	**Data protection area 1 lower-limit address**	DPA1L
19				**DPA1U**	**Data protection area 1 upper-limit address**	DPA1U
20				**DPA2L**	**Data protection area 2 lower-limit address**	DPA2L
21				**DPA2U**	**Data protection area 2 upper-limit address**	DPA2U
22				**DPA3L**	**Data protection area 3 lower-limit address**	DPA3L
23				**DPA3U**	**Data protection area 3 upper-limit address**	DPA3U
24	**MCA**	**Memory protection setting check address**		**DPA4L**	**Data protection area 4 lower-limit address**	DPA4L
25	**MCS**	**Memory protection setting check size**		**DPA4U**	**Data protection area 4 upper-limit address**	DPA4U
26	**MCC**	**Memory protection setting check command**		**DPA5L**	**Data protection area 5 lower-limit address (2^n specification only)**	DPA5L
27	**MCR**	**Memory protection setting check result**		**DPA5U**	**Data protection area 5 upper-limit address (2^n specification only)**	DPA5U

5.1.1 VSECR – System register protection violation cause

This register indicates the number of times violation has been detected by the system register protection function.

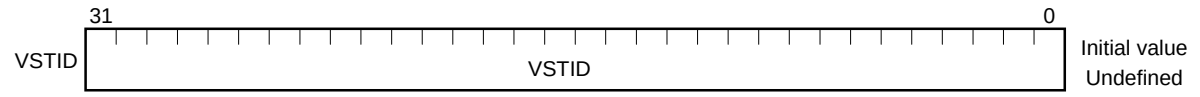
Be sure to set 0 to bits 31 to 8.



Bit Position	Bit Name	Description
7 to 0	VSEC	These bits save the number of times the system register protection violation has been detected. When the value of the VSEC bit has reached 255, it is not incremented any longer and remains as 255. When violation processing has been performed, clear the VSECR register by program.

5.1.2 VSTID – System register protection violation task identifier

This register saves the contents of the task identifier (TID) when the first instruction that is detected by the system register protection function as violation is executed.

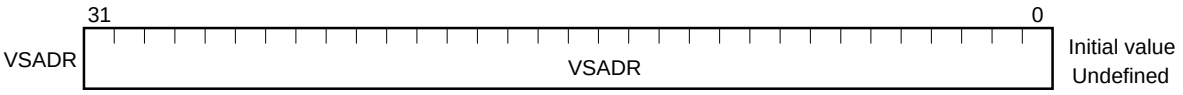


Bit Position	Bit Name	Description
31 to 0	VSTID	These bits store the value of the TID register (in the processor protection setting bank) if system register protection violation is detected while the VSEC bit of the VSECR register is 0. The VSTID register saves the identifier of the task that first violates system register protection after violation processing.

5.1.3 VSADR – System register protection violation address

This register saves the PC of the first instruction that is detected as violation by the system register protection function.

Bit 0 is fixed to 0.



Bit Position	Bit Name	Description
31 to 0	VSADR	These bits store the PC of the instruction if system register protection violation is detected while the VSEC bit of the VSECR register is 0. The VSADR register saves the PC of the instruction that first violates system register protection after violation processing.

Caution For a CPU whose instruction addressing range is partially limited to 512 MB, a value resulting from a sign-extension of bit 28 of the VSADR register is automatically set to bits 31 to 29.

5.2 Access Control

A write access to the system register is controlled by the PSW.NPV bit[Note].

When the PSW.NPV bit is cleared (0) (T state), all the system registers that can be specified by the LDSR instruction can be written. On the other hand, when the NPV bit is set (1) (NT state), a write access by the LDSR instruction to specific system registers that are protected and to specific bits of such registers is blocked, and the written value is not reflected on the registers.

By controlling write accesses in this way, the setting of the system registers can be protected from being changed by a program that is not trusted (non-trusted program).

Note Only the write access by the LDSR instruction is controlled. The EI, DI, and return instructions (EIRET, FERET, and RETI) and operations to update the other system registers are not subject to access control.

Cautions 1. When the execution level auto transition function is disabled (AUE = 0), the NPV bit is fixed to 0. As a result, write operations are not blocked by the system register protection function.

2. Note that the PSW.NPV bit itself is also subject to system register protection. If the PSW.NPV bit has been once set (1), it cannot be cleared (0) unless an exception occurs.

5.3 Registers to Be Protected

Refer to the item of system register protection on the list of the system register of below.

- **Main banks**

  PART2 Table 2-2. System Register List (Main Banks)
- **Exception handler switching function 0, 1**

  PART2 Table 2-3. System Register Bank
- **User 0 bank**

  PART2 Table 2-4. System Register List (User 0 Bank)
- **Processor Protection setting bank**

  PART3 Table 2-2. Processor Protection Setting Registers
- **Processor Protection violation bank**

  PART3 Table 2-3. Processor Protection Violation Registers
- **Software Paging bank**

  PART3 Table 2-4. Software Paging Registers
- **FPU status bank**

  PART4 Table 2-1. System Register Bank

Caution All system register numbers for which no function is defined are subject to system register protection.

5.4 Detection of Violation

If the program under execution is not trusted (non-trusted program), set the PSW.NPV bit (1) so that the CPU operates in the NT state. If a write access is made by the LDSR instruction to a system register protected by the system register protection function while the CPU operates in the NT state, system register protection violation is immediately detected.

The following operations are performed when the system register protection violation has been detected.

- The write access operation by the LDSR instruction is blocked (the written value is not reflected on the register value).
- If the value of the VSECR register is 0, the following operations are performed.
  - The value of the TID register when the LDSR instruction is executed is stored in the VSTID register.
  - The PC of the LDSR instruction is stored in the VSADR register.
- The value of the VSECR register is incremented by 1.

Caution	**The PSW register has bits that are protected and bits that are not protected. Therefore, it does not detect violation even if an illegal write access is made to it. However, the write access to the protected bits is blocked and the written value is not reflected on the value of these bits.**

5.5 Operation Method

Check the status of detection of system register violation by using the VSECR, VSTID, and VSADR registers and take an appropriate action each time the task has been changed by the OS (and programs similar to it) or at a specific interval. If system register violation has been detected more than once, the chances are the system registers have been illegally accessed between the previous check and the latest check.

In addition, be sure to clear the VSECR register before returning to the normal processing.

CHAPTER 6 MEMORY PROTECTION

The V850E2M CPU can control accesses to the following two areas on the address space; the program area that is referenced when an instruction is executed (instruction access) and the data area that is referenced when an instruction that accesses the memory is executed (data access), to protect the system from illegal accesses by a program that is not trusted (non-trusted program).

For memory protection for the V850E2M CPU, memory protection areas are specified using a maximum and minimum address. Areas that have a granularity of 16 bytes can be specified. Therefore, suitable protection can be set up by using only a few areas. The specified addresses are retained in 32-bit system registers, and they completely cover a 4 GB logical address space.

6.1 Register Set

Table 6-1 lists the system registers related to the memory protection function.

Table 6-1. System Register Bank

Group	Processor Protection Function (10H)					
Bank	Processor protection violation (00H)			Processor protection setting (01H)		Software paging (10H)
Bank label	MPV, PROT00			MPU, PROT01		PROT10
Register No.	Name	Function		Name	Function	Name
0	**VSECR**	**System register protection violation cause**		MPM	Setting of processor protection operation mode	MPM
1	**VSTID**	**System register protection violation task identifier**		MPC	Specifying processor protection command	MPC
2	**VSADR**	**System register protection violation address**		TID	Task identifier	TID
3	Reserved for function expansion			Reserved for function expansion		VMECR
4	**VMECR**	**Memory protection violation cause**				VMTID
5	**VMTID**	**Memory protection violation task identifier**				VMADR
6	**VMADR**	**Memory protection violation address**		**IPA0L**	**Instruction/constant protection area 0 lower-limit address**	IPA0L
7	Reserved for function expansion			**IPA0U**	**Instruction/constant protection area 0 upper-limit address**	IPA0U
8				**IPA1L**	**Instruction/constant protection area 1 lower-limit address**	IPA1L
9				**IPA1U**	**Instruction/constant protection area 1 upper-limit address**	IPA1U
10				**IPA2L**	**Instruction/constant protection area 2 lower-limit address**	IPA2L
11				**IPA2U**	**Instruction/constant protection area 2 upper-limit address**	IPA2U
12				**IPA3L**	**Instruction/constant protection area 3 lower-limit address**	IPA3L
13				**IPA3U**	**Instruction/constant protection area 3 upper-limit address**	IPA3U
14				**IPA4L**	**Instruction/constant protection area 4 lower-limit address**	IPA4L
15				**IPA4U**	**Instruction/constant protection area 4 upper-limit address**	IPA4U
16				**DPA0L**	**Data protection area 0 lower-limit address (for stack)**	DPA0L
17				**DPA0U**	**Data protection area 0 upper-limit address (for stack)**	DPA0U
18				**DPA1L**	**Data protection area 1 lower-limit address**	DPA1L
19				**DPA1U**	**Data protection area 1 upper-limit address**	DPA1U
20				**DPA2L**	**Data protection area 2 lower-limit address**	DPA2L
21				**DPA2U**	**Data protection area 2 upper-limit address**	DPA2U
22				**DPA3L**	**Data protection area 3 lower-limit address**	DPA3L
23				**DPA3U**	**Data protection area 3 upper-limit address**	DPA3U
24	**MCA**	**Memory protection setting check address**		**DPA4L**	**Data protection area 4 lower-limit address**	DPA4L
25	**MCS**	**Memory protection setting check size**		**DPA4U**	**Data protection area 4 upper-limit address**	DPA4U
26	**MCC**	**Memory protection setting check command**		**DPA5L**	**Data protection area 5 lower-limit address (2^n specification only)**	DPA5L
27	**MCR**	**Memory protection setting check result**		**DPA5U**	**Data protection area 5 upper-limit address (2^n specification only)**	DPA5U

6.1.1 IPAnL – Instruction/constant protection area n lower-limit address (n = 0 to 4)

This register is used to set the lower-limit address and operation of the instruction/constant protection area.

Be sure to set bit 3 to “0”.

IPAnL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AL31	AL30	AL29	AL28	AL27	AL26	AL25	AL24	AL23	AL22	AL21	AL20	AL19	AL18	AL17	AL16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	0	T	S	E

Initial value 00000002H

Bit Position	Bit Name	Description
31 to 4	AL31 to AL4	These bits set the lower-limit address or base address of the protection area depending on the protection area specification mode set by the T bit. Because bits 3 to 0 of the IPAnL register are used for the other setting of the protection area, bits 3 to 0 (AL3 to 0) of the lower-limit address or base address are implicitly 0.
2	T	This bit selects a mode to specify a range of the protection area. When the T bit is cleared (0), an upper-limit/lower-limit specification mode is selected. When it is set (1), a mask/base specification mode is selected. 0: Upper-limit/lower-limit specification mode (AU31 to 0 specify an upper-limit address and AL31 to 0 specify a lower-limit address.) 1: Base/mask specification mode (AU31 to 0 specify a mask address and AL31 to 0 specify a base address.)
1	S[Note]	This bit enables or disables data access to the protection area in the sp (r3) register indirect access mode. When the S bit is cleared (0), accessing data placed in the protection area of the data area in the sp (r3) register indirect access mode is prohibited. If an instruction that accesses the access-prohibited protection area is executed, data protection violation is detected, and MDP exception is immediately acknowledged. 0: Disables data access to protection area in sp (r3) register indirect access mode. 1: Enables data access to protection area in sp (r3) register indirect access mode.
0	E	This bit enables or disable the setting of the protection area. When the E bit is cleared (0), the contents of all the other setting bits are invalid, and no protection area is set. 0: Invalid (Instruction/constant protection area n is not used.) 1: Valid (Instruction/constant protection area n is used.)

Note The S bit is fixed to 0 or 1 depending on the set value of the MPM.SPS bit. For details, refer to **2.2.2 MPM – Setting of processor protection operation mode**.

Remark n = 0 to 4

6.1.2 IPAnU – Instruction/constant protection area n upper-limit address (n = 0 to 4)

This register is used to set the upper-limit address of the instruction/constant protection area.

Be sure to set bits 3 and 2 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
IPAnU	AU31	AU30	AU29	AU28	AU27	AU26	AU25	AU24	AU23	AU22	AU21	AU20	AU19	AU18	AU17	AU16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	AU15	AU14	AU13	AU12	AU11	AU10	AU9	AU8	AU7	AU6	AU5	AU4	0	0	R	X	Initial value 00000000H

Bit Position	Bit Name	Description
31 to 4	AU31 to AU4	These bits set the upper-limit address or mask value of the protection area, depending on the protection area specification mode set by the T bit. Because bits 3 to 0 of the IPAnU register are used for the other setting of the protection area, bits 3 to 0 (AU3 to 0) of the upper-limit address or mask value are implicitly 1. To specify a mask value, be sure to set a value of consecutive 1’s from the least significant bit (the operation cannot be guaranteed if a value having 1 and 0 alternately placed, such as 000050FFH, is specified).
1	R	This bit enables or disables a read access to the protection area. When the R bit is cleared (0), a read access to the data placed in the protection area of the data area is prohibited. If an instruction that reads the access-prohibited protection area is executed, data protection violation is detected and the MDP exception is immediately acknowledged. 0: Disables read access to the protection area. 1: Enables read access to the protection area.
0	X	This bit enables or disables instruction execution for the protection area. When the X bit is cleared (0), execution of the program placed in the protection area of the program area is prohibited. If an instruction is executed for the protection area, instruction protection violation is detected and the MIP exception is immediately acknowledged. 0: Disables instruction execution for the protection area. 1: Enables instruction execution for the protection area.

Remark n = 0 to 4

6.1.3 DPAnL – Data protection area n lower-limit address (n = 0 to 5)

This register is used to set the lower-limit address and operation of the data protection area.

Be sure to set bit 3 to “0”.

DPAnL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AL31	AL30	AL29	AL28	AL27	AL26	AL25	AL24	AL23	AL22	AL21	AL20	AL19	AL18	AL17	AL16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
AL15	AL14	AL13	AL12	AL11	AL10	AL9	AL8	AL7	AL6	AL5	AL4	0	T	S	E	Initial value **Note 1**

Bit Position	Bit Name	Description
31 to 4	AL31 to AL4	These bits set the lower-limit address or base address of the protection area depending on the protection area specification mode set by the T bit. Because bits 3 to 0 of the DPAnL register are used for the other setting of the protection area, bits 3 to 0 (AL3 to AL0) of the lower-limit address or base address are implicitly 0.
2	T	This bit selects a mode to specify a range of the protection area. When the T bit is cleared (0), an upper-limit/lower-limit specification mode is selected. When it is set (1), a mask/base specification mode is selected. 0: Upper-limit/lower-limit specification mode (AU31 to AU0 specify an upper-limit address and AL31 to AL0 specify a lower-limit address.) 1: Base/mask specification mode (AU31 to AU0 specify a mask address and AL31 to AL0 specify a base address.)
1	S[Note 2]	This bit enables or disables data access to the protection area in the sp (r3) register indirect access mode. When the S bit is cleared (0), accessing data placed in the protection area of the data area in the sp (r3) register indirect access mode is prohibited. If an instruction that accesses the access-prohibited protection area for data is executed, data protection violation is detected and MDP exception is immediately acknowledged. 0: Disables data access to protection area in sp (r3) register indirect access mode. 1: Enables data access to protection area in sp (r3) register indirect access mode.
0	E	This bit enables or disable the setting of the protection area. When the E bit is cleared (0), the contents of all the other setting bits are invalid, and no protection area is set. 0: Invalid (Data protection area n is not used.) 1: Valid (Data protection area n is used.)

Notes 1. The default value differs depending on the channel.
DPA0L to DPA4L: 0000 0002H, DPA5L: 0000 0006H

2. The S bit is fixed to 0 or 1 depending on the set value of the MPM.SPS bit. For details, refer to **2.2.2 MPM – Setting of processor protection operation mode**.

Remark n = 0 to 5

6.1.4 DPAnU – Data protection area n upper-limit address (n = 0 to 5)

This register is used to set the upper-limit address of the data protection area.

Be sure to set bits 3 and 0 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DPAnU	AU31	AU30	AU29	AU28	AU27	AU26	AU25	AU24	AU23	AU22	AU21	AU20	AU19	AU18	AU17	AU16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
AU15	AU14	AU13	AU12	AU11	AU10	AU9	AU8	AU7	AU6	AU5	AU4	0	W	R	0	Initial value **Note**

Bit Position	Bit Name	Description
31 to 4	AU31 to AU4	These bits set the upper-limit address or mask value of the protection area, depending on the protection area specification mode set by the T bit. Because bits 3 to 0 of the DPAnU register are used for the other setting of the protection area, bits 3 to 0 (AU3 to AU0) of the upper-limit address or mask value are implicitly 1. To specify a mask value, be sure to set a value of consecutive 1’s from the least significant bit (the operation cannot be guaranteed if a value having 1 and 0 alternately placed, such as 000050FFH, is specified).
2	W	This bit enables or disables a write access to the protection area. When the W bit is cleared (0), a write access to the data placed in the protection area of the data area is prohibited. If an instruction that writes data to the access-prohibited protection area is executed, data protection violation is detected and the MDP exception is immediately acknowledged. 0: Disables write access to the protection area. 1: Enables write access to the protection area.
1	R	This bit enables or disables a read access to the protection area. When the R bit is cleared (0), a read access to the data placed in the protection area of the data area is prohibited. If an instruction that reads the protection area is executed, data protection violation is detected and the MDP exception is immediately acknowledged. 0: Disables read access to the protection area. 1: Enables read access to the protection area.

Note The initial value differs depending on the channel.

DPA0U: 0000 0006H, DPA1U to DPA5U: 0000 0000H

Remark n = 0 to 5

6.1.5 VMECR – Memory protection violation cause

The VMECR register indicates a protection violation cause in case the MIP or MDP exception occurs.

Be sure to set bits 31 to 7 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VMECR	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	VMMS	VM RMW	VMS	VMW	VMR	VMX	0

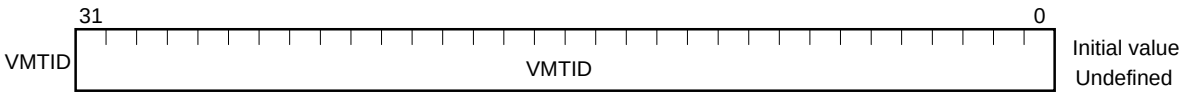
Initial value 00000000H

Bit Position	Bit Name	Description
6	VMMS	This bit indicates whether a data protection exception occurs. It is set (1) if a data protection exception occurs during misaligned access by the LD, ST, SLD, or SST instruction. Otherwise, this bit is cleared (0).
5	VMRMW	This bit indicates whether a data protection exception occurs. It is set (1) if a data protection exception occurs during access by the SET1, CLR1, NOT1, or CAXI instruction. Otherwise, this bit is cleared (0).
4	VMS	This bit is set if an MDP exception occurs due to sp indirect access violation. Otherwise, it is cleared (0). If this bit is set (1), the VMX bit is always cleared (0). This bit may be set (1) together with the VMR and VMW bits.
3	VMW	This bit is set (1) if an MDP exception occurs due to write access violation. Otherwise, it is cleared (0). If this bit is set (1), the VMX and VMR bits are always cleared (0). This bit may be set (1) together with the VMS bit.
2	VMR	This bit is set (1) if an MDP exception occurs due to read access violation. Otherwise, it is cleared (0). If this bit is set (1), the VMX and VMW bits are always cleared (0). This bit may be set (1) together with the VMS bit.
1	VMX	This bit is set if an MIP exception occurs due to instruction execution violation. Otherwise, it is cleared (0). If this bit is set (1), the VMR, VMW, and VMS bits are always cleared (0).

Remark For the status of each bit in case of an exception, refer to **9.3 Identifying Violation Cause**.

6.1.6 VMTID – Memory protection violation task identifier

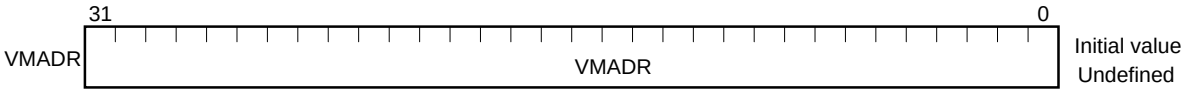
The VMTID register stores the identifier of a task in case of the MIP or MDP exception.



Bit Position	Bit Name	Description
31 to 0	VMTID	These bits store the contents of the identifier of the task (value of the TID register) when an instruction that causes either execution protection violation or data protection violation is executed.

6.1.7 VMADR – Memory protection violation address

The VMADR register stores the address when the MIP or MDP exception has occurred.



Bit Position	Bit Name	Description
31 to 0	VMADR	These bits store the PC of the instruction that violates execution protection if execution protection violation is detected, or the address of a data access if data protection violation is detected.

Caution The PC value of the instruction that has detected execution protection violation may not match the address of the instruction that has actually violated execution protection if the former instruction is placed, extending from one address to another.

6.2 Access Control

The PSW.IMP bit controls accesses to the program area that is referenced when an instruction is executed. If the IMP bit is cleared (0) (T state), instruction execution in the entire program area is enabled, and an instruction at any position can be freely executed. On the other hand, if the IMP bit is set (1) (NT state), an execution of instruction to the entire program area is prohibited in general, and only instructions to a range enabled as a protection area where instruction execution is enabled.

The PSW.DMP bit controls accesses to the data area that is referenced when an instruction that accesses the memory is executed. If the DMP bit is cleared (0) (T state), memory access in the entire data area is enabled, and data at any position can be freely read and written. On the other hand, if the DMP bit is set (1) (NT state), memory access to the entire data area is disabled in general, and only the data in a range enabled as a protection area can be manipulated. In addition, access control that considers writing, reading and stack manipulation is performed.

By these access control features, illegal instruction execution or data access by a program not trusted (non-trusted program) can be prevented.

6.3 Setting Protection Area

In principle, the program area or data area is prohibited from being accessed. To use memory protection, a protection area where access is enabled is specified in these areas for each program not trusted (non-trusted program). The protection area can be enabled or disabled depending on the type of the access (execution, read, or write).

The V850E2M CPU uses the following two registers as a pair to set a protection area.

- IPAnL/IPAnU (n = 0 to 4)
- DPAmL/DPAmU (m = 0 to 5)

Each protection area is set by two combinations of registers: an upper-limit register and a lower-limit register. The registers defined for the V850E2M CPU allow the placement of up to 11 protection areas, including up to five program protection areas and six data protection areas.

The settings that can be specified for each area are shown in **Table 6-2** below. Note that, depending on the register, the values of some bits are fixed.

Table 6-2. Setting Protection Area

Register		xPAnU					xPAnL				
		Bits 31 to 4	Bit 3	Bit 2	Bit 1	Bit 0	Bits 31 to 4	Bit 3	Bit 2	Bit 1	Bit 0
	Field function	Upper-limit address (mask value)	(RFU)	Write enable	Read enable	Execution enable	Lower-limit address (base address)	(RFU)	Area specification mode	sp indirect access enable	Area enable
	Field name	AU		W	R	X	AL		T	S	E
IPA0U/L	Instruction protection area setting	Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	0	0/1Note	(0)
IPA1U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	0	0/1Note	(0)
IPA2U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	0	0/1Note	(0)
IPA3U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	0	0/1Note	(0)
IPA4U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	0	0/1Note	(0)
DPA0U/L	Data protection area setting (for stack)	Upper-limit address	0	1	1	0	Lower-limit address	0	0	1	(0)
DPA1U/L	Data protection area setting	Upper-limit address	0	(0)	(0)	0	Lower-limit address	0	0	0/1Note	(0)
DPA2U/L		Upper-limit address	0	(0)	(0)	0	Lower-limit address	0	0	0/1Note	(0)
DPA3U/L		Upper-limit address	0	(0)	(0)	0	Lower-limit address	0	0	0/1Note	(0)
DPA4U/L		Upper-limit address	0	(0)	(0)	0	Lower-limit address	0	0	0/1Note	(0)
DPA5U/L		Mask value	0	(0)	(0)	0	Base address	0	1	0/1Note	(0)

Note S = 1 if the MPM.SPS bit is 0. S = 0 if the MPM.SPS bit is 1.

Remark 0: Be sure to set this bit to 0.
1: Be sure to set this bit to 1.
(0): This bit may be set by the user. The value in parentheses indicates the initial value.

Therefore, the program and data areas are correlated with setting registers as follows.

Program area (enabling/disabling instruction execution)

- IPAnL/IPAnU (n = 0 to 4)

Data area (enabling/disabling read or write execution)

- IPAnL/IPAnU (n = 0 to 4) … Enables only read.
- DPA0L/DPA0U … Always enables sp indirect access. Always enables read and write.
- DPAnL/DPAnU (n = 1 to 4)
- DPA5L/DPA5U … Mask/base specification mode

The IPAnL/IPAnU registers can be used to set up the protection areas for instruction execution permission and read access permission. These permissions must be specified if the SWITCH and CALLT instructions are used to jump to tables and there are tables placed in the instruction code.

For data areas, only the DPA0L/DPA0U register is always permitted sp relative access. This is because there is only one stack referenced during program execution and only one area for which sp relative access is performed.

The default value of each register is as follows.

Register	Initial Value
IPA0L to IPA4L	0000 0002H
IPA0U to IPA4U	0000 0000H
DPA0L	0000 0002H
DPA0U	0000 0006H
DPA1L to DPA4L	0000 0002H
DPA1U to DPA4U	0000 0000H
DPA5L	0000 0006H
DPA5U	0000 0000H

The function of each bit is described below.

6.3.1 Valid bit (E bit)

This bit indicates whether setting of a protection area is enabled or disabled.

When the E bit is cleared (0), all the contents of the other setting bits are invalid, and a protection area is not set.

6.3.2 Execution enable bit (X bit)

This bit enables or disables instruction execution for the protection area.

When the X bit is cleared (0), execution of a program placed in the protection area of the program area is disabled.

If an instruction for the protection area is executed, an instruction protection violation is detected and the MIP exception is immediately acknowledged.

6.3.3 Read enable bit (R bit)

This bit enables or disables a read access to the protection area.

When the R bit is cleared (0), a read access to data placed in the protection area of the data area is prohibited.

If an instruction that reads the access-prohibited protection area is executed, data protection violation is detected and the MPD exception is immediately acknowledged.

6.3.4 Write enable bit (W bit)

This bit enables or disables a write access to the protection area.

When the W bit is cleared (0), a write access to data placed in the protection area of the data area is prohibited.

If an instruction that writes data to the protection area is executed, data protection violation is detected and the MPD exception is immediately acknowledged.

6.3.5 sp indirect access enable bit (S bit)

This bit enables or disables a data access in the sp (r3) register indirect access mode to the protection area.

When the S bit is cleared (0), a data access in the sp (r3) register indirect access mode to data placed in the protection area of the data address space is prohibited.

If an instruction that accesses the protection area for data in the sp (r3) register indirect access mode is executed, data protection violation is detected and the MPD exception is immediately acknowledged.

Caution	**The S bit of each register is fixed to 0 or 1 depending on the set value of the MPM.SPS bit. For details, refer to 2.2.2 MPM – Processor protection operation mode.**

6.3.6 Protection area specification mode bit (T bit)

This bit selects a mode of specifying a range of the protection area.

When the T bit is cleared (0), an upper-limit/lower-limit specification mode is selected. When it is set (1), a mask/base specification mode is selected.

(1) Upper-limit/lower-limit specification mode

This mode specifies a range of the protection area by specifying the upper-limit address with the AU31 to AU0 bits and the lower-limit address with the AL31 to AL0 bits. If a value greater than the upper-limit address is set, the protection area is invalid.

(2) Mask/base specification mode

This mode specifies a range of the protection area by specifying a mask value with the AU31 to AU0 bits and a base address with the AL31 to AL0 bits. An address range specified by masking the specified base address with the mask value is the protection area.

The protection area is of size of the power of 2 that is indicated by the upper-limit and lower-limit addresses as follows.

Lower-limit address = (AL31 to AL0 and (not AU31 to AU0))

Upper-limit address = (AL31 to AL0 or AU31 to AU0)

6.3.7 Protection area lower-limit address (AL31 to AL0 bits)

The lower-limit address or base address of the protection area is indicated by the set protection area specification mode (T bit).

Because the bits 3 to 0 of the IPAnL and DPAmL registers are used for the other setting of the protection area, the bits 3 to 0 (AL3 to AL0) of the lower-limit address or base address are implicitly 0 (n = 0 to 4, m = 0 to 5).

6.3.8 Protection area upper-limit address (AU31 to AU0 bits)

The upper-limit address or mask value of the protection area is indicated by the protection area specification mode set by the T bit.

Because the bits 3 to 0 of the IPAnU and DPAmU registers are used for the other setting of the protection area, the bits 3 to 0 (AU3 to AU0) of the upper-limit address or mask value are implicitly 1.

When specifying a mask value, be sure to set a value of consecutive 1’s from the least significant bit (the operation is not guaranteed if a value having 1 and 0 alternately placed, such as 000050FFH, is specified) (n = 0 to 4, m = 0 to 5).

6.4 Notes on Setting Protection Area

6.4.1 Crossing of protection area boundaries

If the range of a protection area is set in duplicate, regardless of whether in an instruction/constant protection area or data protection area, setting access control of the crossing part takes precedence.

(1) Instruction/constant protection area

If two or more protection area are set at certain addresses and if execution is enabled in one of the protection areas, enabling execution is assumed. If read is enabled in one of the areas, enabling read is assumed.

(2) Data protection area

If two or more protection areas are set at certain addresses and if read is enabled in one of the protection areas, enabling read is assumed.

The same applies to enabling write and sp indirect access.

6.4.2 Invalid protection area setting

Setting of a protection area is invalid in the following case.

- If a value greater than the upper-limit address is set to the lower-limit address

6.5 Stack Inspection Function

The V850E2M CPU prescribes use of a general-purpose register r3 as a stack pointer (sp) for a program model in terms of stack manipulation. It also defines instructions to generate and delete a stack frame that implicitly uses sp as the stack pointer. To support this program model and to strictly manage the stack, a stack inspection function is provided.

Stack indirect access can be enabled or disabled in each protection area by using the S bit of each protection area setting register[Note].

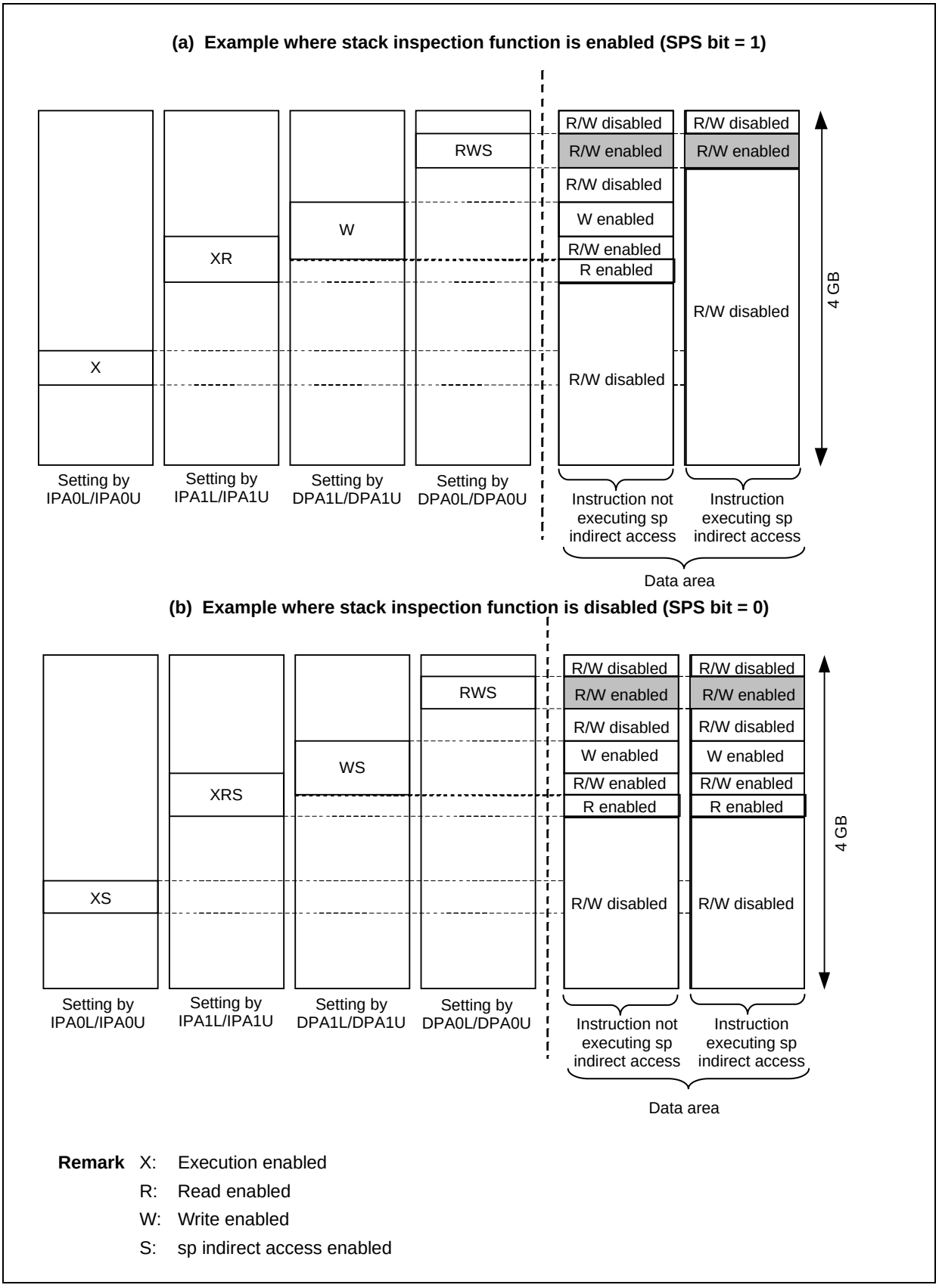
If a memory access instruction that executes an indirect access using sp as the base register is executed in a protection area where sp indirect access is disabled (S bit is 0), that access is detected as violation. If an indirect access using a register other than sp as the base register is executed in the same area, the access is controlled in accordance with the read enable bit (R bit) and write enable bit (W bit).

In a protection area where sp indirect access is enabled (S bit is 1), the access is controlled in accordance with the read enable bit (R bit) and write enable bit (W bit) in any case.

This stack inspection function detects an sp indirect access violation in other protection areas if the value of sp is incorrect with a program model that places only one protection area where sp indirect access is enabled, so that the memory can be protected more powerfully.

Note The V850E2M CPU controls the value of the S bit of each protection area all at once in accordance with the value of the MPM.SPS register.

Figure 6-1. Example of Stack Inspection Function



Remark X: Execution enabled
R: Read enabled
W: Write enabled
S: sp indirect access enabled

6.6 Special Memory Access Instructions

The V850E2M CPU has instructions that access the memory more than once while one of the instructions is executed. For these instructions, the memory protection function performs a special operation. Instructions subject to special protection operations are described below.

- Load and store instructions that execute misaligned access (LD, ST, SLD, and SST)
- Some bit manipulation instructions (SET1, NOT1, and CLR1) and CAXI instruction
- Stack frame manipulation instructions (PREPARE and DISPOSE)
- SYSCALL instruction

6.6.1 Load and store instructions executing misaligned access

With the V850E2M CPU, data can be allocated at all addresses regardless of the data format (byte, halfword, or word). A misaligned access indicates an access to an address other than a halfword boundary (the least significant bit of the address is 0) when the data to be processed is in the halfword format, and an access to an address other than the word boundary when the data to be processed is in a word format.

When a misaligned access is made, access is enabled if all the addresses to be accessed are in one protection area, and if read is enabled when the load instruction is executed or if write is enabled when the store instruction is executed.

Caution	**Even if two protection areas are defined at consecutive addresses without overlapping each other, a misaligned access extending between the two protection areas is judged as protection violation.**

6.6.2 Some bit manipulation instructions and CAXI instruction

Some bit manipulation instructions (SET1, NOT1, and CLR1) and CAXI instruction detect data protection violation if the address to be accessed is enabled from being read but not from being written.

6.6.3 Stack frame manipulation instructions

The stack frame manipulation instructions (PREPARE and DISPOSE) generates as many memory accesses as the number of registers specified. The memory protection function detects violation of each of these memory accesses and, as soon as it has detected violation, it aborts execution of the stack frame manipulation instruction at occurrence of a data protection exception. However, memory accesses preceding the memory access from which violation has been detected are executed. The DISPOSE instruction writes a general-purpose register corresponding to the executed memory access. If the access has been aborted, sp is not updated.

The stack frame manipulation instruction that has been aborted is executed from the beginning again when execution returns from the exception. Consequently, the same memory access as that which was executed once before execution was aborted is executed again.

6.6.4 SYSCALL instruction

The SYSCALL instruction is used to call a service supplied by a management program such as an OS (and programs similar to it). The service is a trusted program and the address table to branch to the service is also trusted. Therefore, memory protection is not applied to the memory access by the SYSCALL instruction even if the PSW.DMP bit is set (1).

Consequently, the MDP exception is never detected while the SYSCALL instruction is executed.

6.7 Protection Violation and Exception

If an instruction is executed on or a data access is made to an address that is not enabled, instruction protection violation or data protection violation is detected. If violation is detected, the following operations are performed.

For details of the MIP and MDP exceptions, refer to **CHAPTER 9 PROCESSOR PROTECTION EXCEPTION**.

(1) If instruction protection violation is detected

- Execution of the instruction placed at the address where instruction protection violation has been detected does not start.
- An access to the address where instruction protection violation has been detected does not make any request to the outside of the CPU.
- The MIP exception occurs and exception processing starts immediately.

(2) If data protection violation is detected

- Execution of the instruction that accesses the address where data protection violation has been detected is aborted.
- An access to the address where data protection violation has been detected does not make any request to the outside of the CPU.
- The MDP exception occurs and exception processing starts immediately.

CHAPTER 7 PERIPHERAL DEVICE PROTECTION

Peripheral device protection controls access to protect peripheral devices (areas) from illegal access by untrusted programs.

The V850E2M CPU assumes that the connected peripheral devices (such as I/O devices and small scale memory) differ for each application system. If there are many such devices, they are placed at discrete addresses that are finer than the granularity of areas subject to memory protection.

When using peripheral device protection, different settings can be specified for each peripheral device, and protection is possible using a granularity that is not possible by using protection areas, the number of which is limited by memory protection.

7.1 Register Set

Table 7-1 lists the peripheral device registers related to the peripheral device protection function.

For the V850E2M CPU, the register set used for peripheral device protection is placed in the peripheral device register area. The base address for the peripheral device protection registers is defined by hardware and is FFFF5100H for the V850E2M CPU. The PPSn, PPPn, PPVn, and PPTn registers make up one set and can be used to set up protection for 32 peripheral devices, with each bit corresponding to one peripheral device. Multiple sets might be available, depending on the number of peripheral devices for a product and the peripheral device protection policy. A total of nine sets are provided for the V850E2M CPU, and they are specified by suffixing a value from 0 to 8 to the register names.

Caution These registers related to the peripheral device protection function are also subject to peripheral device protection. For details, refer to 7.9 Protection Setting for Peripheral Device Protection Setting Registers.

Table 7-1. Peripheral Device Protection Function Register Set

Register Name	Offset Address	Accessible Size				Initial Value
		1	8	16	32	
PPM	+00H	√	√	√	√	00000000H
PPEC	+04H		√	√	√	00000000H
VPNECR	+10				√	Undefined[Note]
VPNADR	+14				√	Undefined[Note]
VPNTID	+18				√	Undefined[Note]
VPTECR	+20				√	Undefined[Note]
VPTADR	+24				√	Undefined[Note]
VPTTID	+28				√	Undefined[Note]
PPVn	+40H + (n*10h) + 0H	√	√	√	√	Refer to **Table 7-2.**
PPTn	+40H + (n*10h) + 4H	√	√	√	√	Refer to **Table 7-2.**
PPPn	+40H + (n*10h) + 8H	√	√	√	√	Refer to **Table 7-2.**
PPSn	+40H + (n*10h) + CH	√	√	√	√	Refer to **Table 7-2.**

Note This register is not initialized and its previous value is retained following a reset. The initial value is undefined.

Remark n = 0 to 8

The names, addresses, initial values, and protection target addresses of the PPSn, PPPn, PPVn, and PPTn registers for the V850E2M CPU are shown in Table 7-2. For a list peripheral devices that the register bits correspond to, see **APPENDIX D PERIPHERAL DEVICE PROTECTION AREAS.**

Table 7-2. PPSn, PPPn, PPVn, PPTn registers of V850E2M CPU

Register Name	Address	Reset Value	Protection Setting Target Address
PPV0	FFFF5140H	00030000H	CPU specific peripheral devices and CPU system peripheral devices
PPT0	FFFF5144H	00030000H	
PPP0	FFFF5148H	00030000H	
PPS0	FFFF514CH	00000000H	
PPV1	FFFF5150H	00000000H	FF400000 to FF5FFFFFH
PPT1	FFFF5154H	00000000H	
PPP1	FFFF5158H	00000000H	
PPS1	FFFF515CH	00000000H	
PPV2	FFFF5160H	00000000H	FF600000 to FF7FFFFFH
PPT2	FFFF5164H	00000000H	
PPP2	FFFF5168H	00000000H	
PPS2	FFFF516CH	00000000H	
PPV3	FFFF5170H	00000000H	FF800000 to FF81FFFFH
PPT3	FFFF5174H	00000000H	
PPP3	FFFF5178H	00000000H	
PPS3	FFFF517CH	00000000H	
PPV4	FFFF5180H	00000000H	FF820000 to FF83FFFFH
PPT4	FFFF5184H	00000000H	
PPP4	FFFF5188H	00000000H	
PPS4	FFFF518CH	00000000H	
PPV5	FFFF5190H	00000000H	FFFF8000 to FFFF9FFFH
PPT5	FFFF5194H	00000000H	
PPP5	FFFF5198H	00000000H	
PPS5	FFFF519CH	00000000H	
PPV6	FFFF51A0H	00000000H	FFFFA000 to FFFFBFFFH
PPT6	FFFF51A4H	00000000H	
PPP6	FFFF51A8H	00000000H	
PPS6	FFFF51ACH	00000000H	
PPV7	FFFF51B0H	00000000H	FFFFC000 to FFFFDFFFH
PPT7	FFFF51B4H	00000000H	
PPP7	FFFF51B8H	00000000H	
PPS7	FFFF51BCH	00000000H	
PPV8	FFFF51C0H	00000000H	FFFFE000 to FFFFFFFFH
PPT8	FFFF51C4H	00000000H	
PPP8	FFFF51C8H	00000000H	
PPS8	FFFF51CCH	00000000H	

7.1.1 PPM – Setting of peripheral device protection operation mode

This register is used to set an operation mode related to the peripheral device protection function. It can be accessed in units of 32, 16, 8, or 1 bit.

Be sure to set bits 31 to 1 to “0”.

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

Initial value 00000000H

Bit Position	Bit Name	Description
0	PPMPPMSK	This bit selects whether a peripheral device protection exception (PPI exception) is to be used. 0: Uses PPI exception (initial value). 1: Does not use PPI exception. When the PPMPPMSK bit is set (1), the PPEC.PPECPPVD bit remains unchanged even if peripheral device protection violation is detected in the NT state. Therefore, the PPI exception is not reported and the PPI exception does not occur. Be sure to manipulate the PPMPPMSK bit while the PSW.PP bit is 0 and PPEC.PPECPPVD bit is 0.

7.1.2 PPEC – Controlling peripheral device protection exception

This register controls the report status of an exception. It can be accessed in units of 32, 16, or 8 bits.

Be sure to set bits 31 to 1 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
PPEC	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	PPEC PPVD	Initial value 00000000H

Bit Position	Bit Name	Description
0	PPEC PPVD[Note]	This bit indicates the report status of the PPI exception. When this bit is set (1), the PPI exception is reported to the CPU and the PPI exception is not acknowledged. This bit is automatically cleared (0) as soon as the CPU has acknowledged the PPI exception. When the PPM.PPMPPMSK bit is set (1), this bit is automatically set (1) and the PPI exception is reported if peripheral device protection violation is detected during operation in the NT state. When the PPMPPMSK bit is 1, the value will not automatically change. When the PPECPPVD bit is set and when PSW.PP = 1, data access of all instructions that access memory is disabled (except data access by the SYSCALL instruction). When the PPECPPVD bit is set (1), report of the PPI exception can be canceled by clearing (0) the PPECPPVD bit by program. When report of the PPI exception has been canceled, the CPU does not acknowledge the PPI exception. 0: PPI exception is not reported (Initial value). 1: PPI exception is reported.

Note The PPECPPVD bit can only be cleared (0) by a write operation. It cannot be set (1).

7.1.3 VPNECR – Peripheral device protection NT state violation cause

This 32-bit register saves information on an access that has violated peripheral device protection in the NT state. It can be accessed in units of 32 bits.

Be sure to set bits 31 to 16, 11, 7, 6, and 3 to 1 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VPNECR	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
VPNECRSP	VPNECROP	VPNECRRWP	VPNECRWP	0	VPNECRWD	VPNECRHW	VPNECRBY	0	0	VPNECRRD	VPNECRWR	0	0	0	VPNECRVD

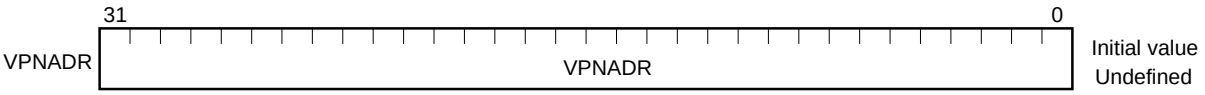
Initial value Undefined

Bit Position	Bit Name	Description
15	VPNECRSP	This bit is set (1) if an access that has violated peripheral device protection is an access to a special peripheral device when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
14	VPNECROP	This bit is set (1) if an access that has violated peripheral device protection is an access to an OS peripheral device when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
13	VPNECRRWP	This bit is set (1) if an access that has violated peripheral device protection is an access to a general peripheral device that is protected from write and read accesses when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
12	VPNECRWP	This bit is set (1) if an access that has violated peripheral device protection is an access to a general peripheral device that is protected from write accesses when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
10	VPNECRWD	This bit is set (1) if an access that has violated peripheral device protection is a word access when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
9	VPNECRHW	This bit is set (1) if an access that has violated peripheral device protection is a halfword access when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
8	VPNECRBY	This bit is set (1) if an access that has violated peripheral device protection is a byte access when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
5	VPNECRRD	This bit is set (1) if an access that has violated peripheral device protection is a read access when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
4	VPNECRWR	This bit is set (1) if an access that has violated peripheral device protection is a write access when the peripheral device protection violation is detected in the NT state. Otherwise, it is cleared (0).
0	VPNECRVD	This bit indicates that peripheral device protection violation has been detected in the NT state. It is set (1) if peripheral device protection violation is detected during an access to a peripheral device in the NT state. If the VPNECRVD bit is set, the VPNECR, VPNADR, and VPNTID registers are not updated but retained even if new peripheral device protection violation is detected in the NT state. Be sure to clear (0) the VPNECRVD bit after completion of the exception processing. Also be sure to clear (0) the VPNECRVD bit when using the peripheral device protection function.

7.1.4 VPNADR – Peripheral device protection NT state violation address

This 32-bit register saves an address to which an access that has violated peripheral device protection is made when the peripheral device protection violation is detected in the NT state. It can be accessed in 32-bit units.

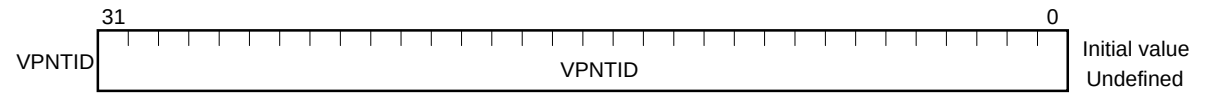
Caution The address saved to this register differs depending on the bus system of the product. For the correspondence between the address stored in this register and the logical address of the architecture, refer to the manual of the product.



Bit Position	Bit Name	Description
31 to 0	VPNADR	If peripheral device protection violation is detected in the NT state, these bits store the memory address at which the violation has occurred.

7.1.5 VPNTID – Peripheral device protection NT state violation task ID

This 32-bit register saves the ID of the task under execution when peripheral device protection violation is detected in the NT state. It can be accessed in 32-bit units.



Bit Position	Bit Name	Description
31 to 0	VPNTID	These bits store the contents of the identifier (value of the TID register) of the task that is executed when the instruction that violates peripheral device protection is executed in the NT state.

7.1.6 VPTECR – Peripheral device protection T state violation cause

This 32-bit register saves information on the access that has violated peripheral device protection when the peripheral device protection violation is detected in the T state. It can be accessed in 32-bit units. Be sure to set 0 to bits 31 to 16, 14 to 11, 7, 6, and 3 to 1.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VPTECR	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
VPTECRSP	0	0	0	0	VPTECRWD	VPTECRHW	VPTECRBY	0	0	VPTECRRD	VPTECRWR	0	0	0	VPTECRVD

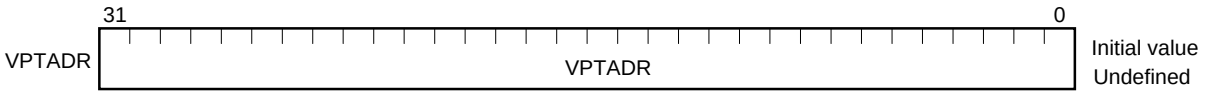
Initial value Undefined

Bit Position	Bit Name	Description
15	VPTECRSP	This bit is set (1) if an access that has violated peripheral device protection is an access to a special peripheral device when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
10	VPTECRWD	This bit is set (1) if an access that has violated peripheral device protection is a word access when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
9	VPTECRHW	This bit is set (1) if an access that has violated peripheral device protection is a halfword access when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
8	VPTECRBY	This bit is set (1) if an access that has violated peripheral device protection is a byte access when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
5	VPTECRRD	This bit is set (1) if an access that has violated peripheral device protection is a read access when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
4	VPTECRWR	This bit is set (1) if an access that has violated peripheral device protection is a write access when the peripheral device protection violation is detected in the T state. Otherwise, it is cleared (0).
0	VPTECRVD	This bit indicates that peripheral device protection violation has been detected in the T state. It is set (1) if peripheral device protection violation is detected during an access to a peripheral device in the T state. If the VPTECRVD bit is set, the VPTECR, VPTADR, and VPPTID registers are not updated but retained even if new peripheral device protection violation is detected in the T state. Be sure to clear (0) the VPTECRVD bit after completion of the exception processing. Also be sure to clear (0) the VPTECRVD bit when using the peripheral device protection function.

7.1.7 VPTADR – Peripheral device protection T state violation address

This 32-bit register saves an address to which the access that has violated peripheral device protection is made when the peripheral device protection violation is detected in the T state. It can be accessed in 32-bit units.

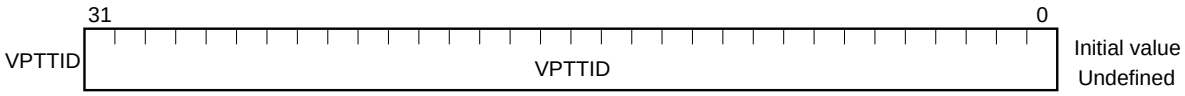
Caution The address saved to this register differs depending on the bus system of the product. For the correspondence between the address stored in this register and the logical address of the architecture, refer to the manual of the product.



Bit Position	Bit Name	Description
31 to 0	VPTADR	If peripheral device protection violation is detected in the T state, these bits store the memory address at which the violation has occurred.

7.1.8 VPTTID – Peripheral device protection T state violation task ID

This 32-bit register saves the ID of the task under execution when peripheral device protection violation is detected in the T state. It can be accessed in 32-bit units.



Bit Position	Bit Name	Description
31 to 0	VPTTID	These bits store the contents of the identifier (value of the TID register) of the task that is executed when the instruction that violates peripheral device protection is executed in the T state.

7.1.9 PPSn – Specification of special peripheral device

This register specifies a peripheral device corresponding to each of its bits as a special peripheral device. It can be accessed in units of 32, 16, 8, or 1 bit. This register is set to the initial value defined for each product at reset.

PPSn

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PPSn31	PPSn30	PPSn29	PPSn28	PPSn27	PPSn26	PPSn25	PPSn24	PPSn23	PPSn22	PPSn21	PPSn20	PPSn19	PPSn18	PPSn17	PPSn16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPSn15	PPSn14	PPSn13	PPSn12	PPSn11	PPSn10	PPSn9	PPSn8	PPSn7	PPSn6	PPSn5	PPSn4	PPSn3	PPSn2	PPSn1	PPSn0

Initial value
Refer to Table 7-2

Bit Position	Bit Name	Description
31 to 0	PPSn31 to PPSn0	Each of these bits specifies a corresponding peripheral device as a special peripheral device. When this bit is set to 1, it is recommended to set the bit of the PPPn, PPVn, and PPTn registers corresponding to this bit to 1. The special peripheral device detects all accesses as peripheral device protection violation even while a trusted program is being executed (T state). 0: Peripheral device corresponding to this bit is not a special peripheral device. 1: Peripheral device corresponding to this bit is a special peripheral device. The initial value is defined as the product specification in accordance with each system requirement, and may be fixed to a specific value.

7.1.10 PPPn – Specification of OS peripheral device

This register specifies a peripheral device of an area corresponding to each of its bits as an OS peripheral device. It can be accessed in units of 32, 16, 8, or 1 bit. This register is set to the initial value defined for each product after reset.

PPPn

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PPPn31	PPPn30	PPPn29	PPPn28	PPPn27	PPPn26	PPPn25	PPPn24	PPPn23	PPPn22	PPPn21	PPPn20	PPPn19	PPPn18	PPPn17	PPPn16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPPn15	PPPn14	PPPn13	PPPn12	PPPn11	PPPn10	PPPn9	PPPn8	PPPn7	PPPn6	PPPn5	PPPn4	PPPn3	PPPn2	PPPn1	PPPn0

Initial value
Refer to Table 7-2

Bit Position	Bit Name	Description
31 to 0	PPPn31 to PPPn0	Each of these bits specifies a corresponding peripheral device as an OS peripheral device. When this bit is set to 1, it is recommended to set the bit of the PPVn and PPTn register corresponding to this bit to 1. If the specified peripheral device has already been specified as a special peripheral device by the PPSn register, setting this bit to 1 is recommended. The OS peripheral device can be accessed only by a trusted program (T state). All accesses to the OS peripheral device from a program not trusted (NT state) are detected as peripheral device protection violation. 0: Peripheral device corresponding to this bit is not an OS peripheral device. 1: Peripheral device corresponding to this bit is an OS peripheral device. The initial value is defined as the product specification in accordance with each system requirement, and may be fixed to a specific value.

7.1.11 PPVn – Validating general peripheral device protection

This register specifies whether an access to a general peripheral device is detected as violation. It can be accessed in units of 32, 16, 8, or 1 bit. This register is set to the initial value defined for each product at reset.

PPVn

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PPVn31	PPVn30	PPVn29	PPVn28	PPVn27	PPVn26	PPVn25	PPVn24	PPVn23	PPVn22	PPVn21	PPVn20	PPVn19	PPVn18	PPVn17	PPVn16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPVn15	PPVn14	PPVn13	PPVn12	PPVn11	PPVn10	PPVn9	PPVn8	PPVn7	PPVn6	PPVn5	PPVn4	PPVn3	PPVn2	PPVn1	PPVn0

Initial value
Refer to Table 7-2

Bit Position	Bit Name	Description
31 to 0	PPVn31 to PPVn0	These bits validate or invalidate the access limitation of a non-trusted program under execution if the peripheral device corresponding to each of these bits is a general peripheral device. If the peripheral device has already been specified as a special peripheral device by the PPSn register or as an OS peripheral device by the PPPn register, it is recommended to set this bit to 1. An access to a general peripheral device is not limited and violation is not detected if this bit is cleared (0). If this bit is set (1), violation may be detected in accordance with the specification of the corresponding bit of the PPTn register. 0: Access to the general peripheral device corresponding to this bit is not limited. 1: Access to the general peripheral device corresponding to this bit is limited. The initial value is defined as the product specification in accordance with each system requirement, and may be fixed to a specific value.

7.1.12 PPTn – Specification protection type of general peripheral device

This register sets details of violation protection of an access to a general peripheral device. It can be accessed only by reading, in units of 32, 16, 8, or 1 bit.

PPTn

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PPTn31	PPTn30	PPTn29	PPTn28	PPTn27	PPTn26	PPTn25	PPTn24	PPTn23	PPTn22	PPTn21	PPTn20	PPTn19	PPTn18	PPTn17	PPTn16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPTn15	PPTn14	PPTn13	PPTn12	PPTn11	PPTn10	PPTn9	PPTn8	PPTn7	PPTn6	PPTn5	PPTn4	PPTn3	PPTn2	PPTn1	PPTn0

Initial value
Refer to
Table 7-2

Bit Position	Bit Name	Description
31 to 0	PPTn31 to PPTn0	These bits indicate the contents of the access limit of a non-trusted program under execution if the peripheral device corresponding to each of these bits is a general peripheral device. If the peripheral device has already been specified as a special peripheral device by the PPSn register or as an OS peripheral device by the PPPn register, it is recommended to set this bit to 1. If the access is limited by a bit of the PPVn register, an access to the specified peripheral device is limited as follows. 0: A read access to the general peripheral device corresponding to this bit is not limited. A write access to the general peripheral device corresponding to this bit is assumed as violation. 1: A read access to the general peripheral device corresponding to this bit is assumed as violation. A write access to the general peripheral device corresponding to this bit is assumed as violation. The initial value is defined as the product specification in accordance with each system requirement, and may be fixed to a specific value.

7.2 Standing of Memory Protection and Peripheral Device Protection

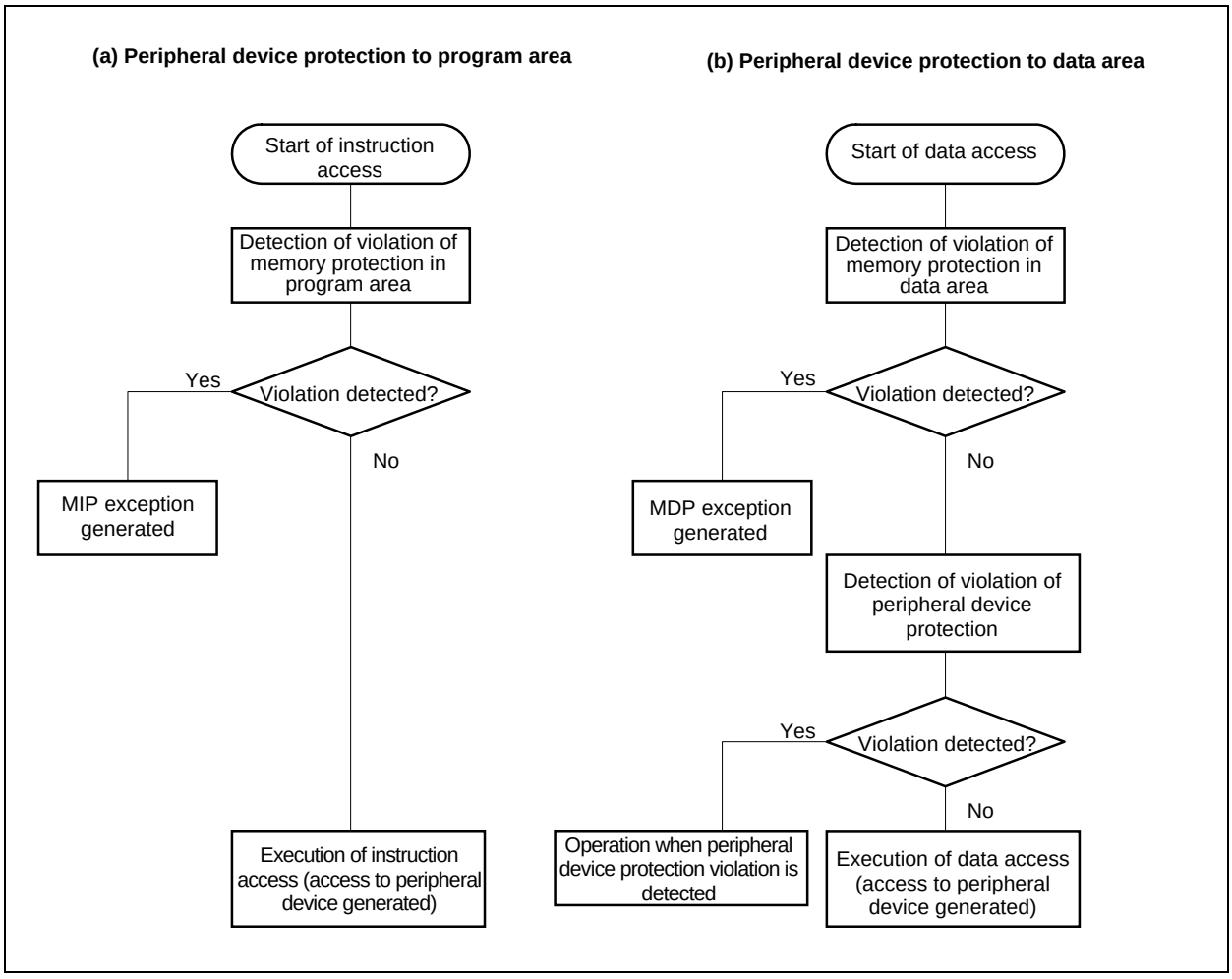
The V850E2M CPU uses memory mapped I/O, which means that memory protection is equally applied to the data areas in the CPU address space. Therefore, memory protection and peripheral device protection are applied each time the data in a peripheral device is accessed. This is because the V850E2M CPU employs memory-mapped I/O architecture and because memory protection is equally applied to the data area in the address space of the CPU[Note].

The CPU first makes a judgment, by using the memory protection function, whether a data access to a peripheral device is enabled or disabled, and detects violation. If the CPU judges that the access is disabled and detects violation, peripheral device protection does not operate and the MDP exception immediately occurs. If the CPU judges that the access is enabled, it makes a judgment, by using the peripheral device protection function, whether this access is enabled or disabled, and detects violation. If the CPU judges that the access is disabled and detects violation, the PPI exception may be generated in some cases, in accordance with the operation setting of peripheral device protection and the status of the CPU. If the CPU judges that the access is enabled, a data access to the peripheral device is executed.

Note Peripheral device protection is not applied to instruction accesses that occur as a result of program execution or branch.

Figure 7-1 shows the relationship between memory protection and peripheral device protection.

Figure 7-1. Standing of Memory Protection and Peripheral Device Protection



7.3 Types of Peripheral Devices

Generally, the user of a peripheral device is defined or the situation where the peripheral device is to be used is restricted in accordance with the usage assumed for each system. Peripheral device protection classifies each peripheral device (or a group of peripheral devices) into the following three categories for management.

- Special peripheral device
- OS peripheral device
- General peripheral device

7.3.1 Setting types of peripheral devices

A combination of the bits of the same number of the PPSn and PPPn registers indicate the type of each peripheral device. Table 7-2 shows which type of peripheral device protection is applied depending on the combination of these bits.

Do not change the PPS and PPP registers while the system is operating except when initializing the system or OS (and programs similar to it).

Table 7-2. Types of Peripheral Devices

Bit of PPSn	Bit of PPPn	Type
1	0 or 1 (1 is recommended)	Special peripheral device
0	1	OS peripheral device
0	0	General peripheral device

(1) Special peripheral device

Of the peripheral devices, the ones that control the most basic environment for the processor to operate are especially called special peripheral devices. For example, peripheral devices that control clock or power supply should be prevented from an inadvertent access under any circumstances. Against this background, a peripheral device specified as a special peripheral device is controlled so that it cannot be accessed even by a “trusted program” unless a specific procedure is followed.

(2) OS peripheral device

Of the peripheral devices, the ones that can be accessed only by a “trusted program” are treated as “OS peripheral devices” because the OS is the representative trusted program. Specify peripheral devices important for OS (and programs similar to it) to at least guarantee its own operation as OS peripheral devices. For example, peripheral devices, such as an interrupt controller and a DMA controller, that make the system unstable if they are used inadvertently should be specified as OS peripheral devices. An OS peripheral device is not enabled to be accessed by a user application that is a “non-trusted program”. Therefore, the OS (and programs similar to it) can guarantee its own operation no matter what operation the user application may perform.

(3) General peripheral device

Peripheral devices that can be accessed even by a “non-trusted program” are treated as “general peripheral devices”. All peripheral devices that are neither special peripheral devices nor OS special devices are treated as general peripheral devices. Assign peripheral devices that have little influence on the overall system even if they are treated with relatively loose regulations, such as general-purpose timers, A/D converters, and macros for communication, as general peripheral devices. For general peripheral devices, a mechanism to perform different access control for each user application is provided.

For details, refer to **7.3.2 Detailed protection setting of general peripheral device**.

7.3.2 Detailed protection setting of general peripheral device

All accesses from a non-trusted program to special peripheral devices and OS peripheral devices are detected as violation. However, a right to access a general peripheral device can be specified for each program under execution. Therefore, detailed protection policy must be specified for the general peripheral device by using combinations of the bits of the PPVn and PPTn registers.

The OS (and programs similar to it) can set a value appropriate for each program to the PPVn register before execution of the program is

started. As a result, accesses to the general peripheral device, which are different from one program to another under execution, can be controlled, and precise peripheral device protection can be realized.

In addition to the PPVn register, the OS (and programs similar to it) can change the PPTn register value. However, it is recommended not to frequently change the PPTn register during system operation in order to avoid a degradation in performance when switching tasks.

Table 7-3. Controlling Access to General Peripheral Device

Bit of PPVn	Bit of PPTn	Access Control of General Peripheral Device
0	0 or 1 (1 is recommended)	Read enabled/write enabled
1	0	Read enabled/write disabled
1	1	Read disabled/write disabled

7.4 Peripheral Device Protection Violation in T State

The CPU indicates the T state (PSW.PP bit = 0) when it executes a trusted program. In this state, peripheral device protection violation is detected only if an access to a special peripheral device is made. If a violation is detected in the T state, the peripheral device protection function performs the following operations.

- Blocks the access that has caused the violation and does not output it to the peripheral device.
- Stores information on the access that has caused the violation in the VPTECR, VPTTID, and VPTADR registers.

To avoid interrupting critical processing that might be performed while in the T state, exceptions are not generated. During operation, check the violation information stored in the VPTECR, VPTTID, and VPTADR registers at the appropriate times, and perform processing for any violations that occur.

When execution is blocked due to a read access violation detected in the peripheral device, the destination register of the load command is updated. At this time, the value stored in the destination register is defined as a product specification.

7.5 Peripheral Device Protection Violation in NT State

The CPU indicates the NT state (PSW.PP bit = 1) when it executes a program that is not trusted. In this state, peripheral device protection violation is detected when an access is made to any of the special peripheral, OS peripheral, and general peripheral devices. If violation is detected in the NT state, the peripheral device protection function performs the following operations that are different from those performed in the T state.

- The access that caused the violation is blocked to prevent effects on peripheral devices.
- Stores information on the access that has caused violation in the VPNECR, VPNTID, and VPNADR registers.
- Reports the PPI exception to the CPU and waits for the start of the PPI exception processing.
- Until the PPI exception processing is started, invalidates all the subsequent memory accesses in the NT state (not only accesses to the peripheral devices but also all accesses to the memory).

For invalidation, refer to **7.5.1 Invalidating subsequent accesses**.

7.5.1 Invalidating subsequent accesses

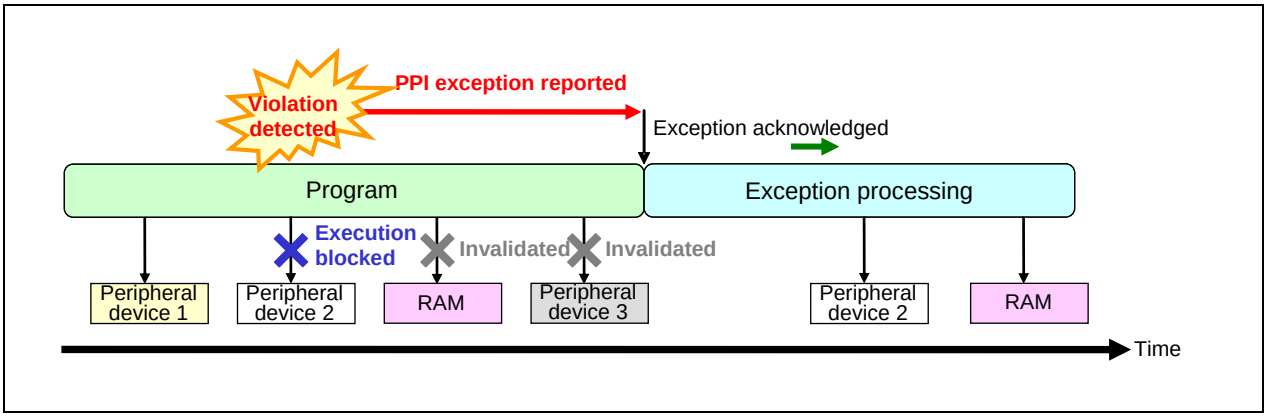
If a peripheral device protection violation by an untrusted program running in the NT state is detected, continuing to execute that program can be judged to be dangerous, and exception processing can be started by using a PPI exception to call the OS (and programs similar to it).

However, due to hardware limitations, the PPI exception is not reported immediately, and there might be a delay before the exception processing starts. At the same time, if there is software processing being performed that cannot be broken up, it might be undesirable to start exception processing. In cases like these, untrusted programs might continue to execute instructions from when the peripheral device protection violation is detected until the PPI exception processing starts.

To resolve this problem, all succeeding memory access is disabled from when a peripheral device protection violation is detected using the NT state until the PPI exception processing starts. After this access is disabled, the system is handled as though memory access requests do not exist, so updates to CPU-external resources by instructions issued after an instruction caused a peripheral device protection violation can be prevented.

Access is typically disabled until the PPI exception processing starts, but, if an interrupt or some other process causes the system to start executing a different program that is in the T state, access is not disabled while the program is in this state. If return processing is performed and the system again enters the NT state, memory access is again disabled.

Figure 7-2. Invalidating Subsequent Memory Accesses



This specification assumes that “non-trusted programs” are scheduled by the OS (and programs similar to it) and that, before other “non-trusted program” is executed, the OS (and programs similar to it) always performs PPI exception processing and then execute the other “non-trusted program”. By controlling the programs in this way, invalidation that took place in a “non-trusted program (NT state)” can be prevented from affecting the other “non-trusted program (NT state)”. “Trusted programs (T state)” are not invalidated in the first place.

When execution is blocked or invalidated due to a read access violation detected in the peripheral device, the destination register of the load instruction is updated.

7.6 Handling PPI Exception

If peripheral device protection violation is detected in a “non-trusted program” that operates in the NT state, it is judged that continuing execution of that program is dangerous, the OS (and programs similar to it) is called through PPI exception, and the exception processing is immediately started.

7.6.1 Canceling PPI exception

Acknowledging the PPI exception may be delayed because of delayed report or indivisible atomic operations of the program. Before processing of the PPI exception is started, the program that has generated the PPI exception may execute termination processing. To avoid this problem, be sure to cancel the PPI exception by using the SYNCE instruction and the PPECPPVD bit of the PPEC register to cancel invalidation before the termination processing. For details, refer to **6.3 Exception Management** in **PART 2**.

7.6.2 Operation method not using PPI exception

Depending on the application, the generation of an exception during operation can mess up a program sequence and lead to loss. For cases like this, operation in which peripheral device protection does not cause PPI exceptions is possible.

To perform such operation, set the PPM.PPMPPMSK bit to 1. If this bit is set, the PPEC.PPECPPVD bit is no longer set to 1 when a peripheral device protection violation is detected, and PPI exceptions do not occur. During operation, check the violation information stored in the VPNECR, VPNTID, and VPNADR registers at the appropriate times, and perform processing for any violations that occur.

7.6.3 Operation to be performed by PPI exception processing

When exception processing has been performed, clear the VPNECR.VPNECRVD bit so that violation information is correctly saved after execution is returned from the exception processing and when the next violation is detected. If this bit is not cleared, violation information corresponding to a memory access that has caused the next violation is not stored even if the next violation is detected.

7.7 List of Results of Detection of Peripheral Device Protection Violation

The results of detecting violations in the NT and T states, and the types of peripheral device protections can be summarized as shown in Table 7-4.

Table 7-4. Results of Detecting Peripheral Device Protection Violation

Type of Peripheral Device	PPSn	PPPn	PPTn	PPVn	Trusted Program (T state)		Non-trusted Program (NT state)	
					PSW.PP = 0		PSW.PP = 1	
					Read	Write	Read	Write
Special peripheral device	1	0 or 1 (1 is recommended)	0 or 1 (1 is recommended)	0 or 1 (1 is recommended)	Violation	Violation	Violation	Violation
OS peripheral device	0	1	0 or 1 (1 is recommended)	0 or 1 (1 is recommended)	–	–	Violation	Violation
General peripheral device	0	0	1	1	–	–	Violation	Violation
			0		–	–	–	Violation
			0 or 1 (1 is recommended)	0	–	–	–	–

Operations after detection of violation can be summarized as follows.

Table 7-5. Operations After Detection of Peripheral Device Protection Violation

Status on Violation Detection	T State	NT State	
PSW.PP	0	1	
PPM.PPMPPMSK	0 or 1	0	1
Blocking violating access	Blocked	Blocked	
Saving violation information	Stored in VPTECR/VPTTID/VPTADR	Stored in VPNECR/VPNTID/VPNADR	
PPI exception	Does not occur	Occurs	Does not occur
Invalidation of successive accesses	Not invalidated	Invalidated	

7.8 Accessing Special Peripheral Devices

Usually, violation by an access to a special peripheral device is detected and blocked in both the NT and T states. To access special peripheral devices in order to control the system behavior, perform the following procedure.

<1> Transfer execution to a trusted program that operates in the T state.
<2> Clear the bits of the PPSn register corresponding to the special peripheral device to be accessed, and temporarily change the device to an OS peripheral device.
<3> Access the peripheral device.
<4> Restore the bits of the PPSn register which have been changed in <2> to the original status (set).

Caution The system security can be increased by making it impossible to access special peripheral devices by using procedures other than the above. Because changing special peripheral devices to OS peripheral devices reduces the system security, it is recommended to minimize how much time is spent making changes by using the above procedure.

7.9 Protection Setting of Peripheral Device Protection Setting Registers

The peripheral I/O registers that set peripheral device protection itself are subject to peripheral device protection. The peripheral device protection setting registers are protected by either of the following policies.

- The peripheral device protection setting registers are implicitly OS peripheral devices.
- Define the PPSn, PPPn, PPVn, and PPTn bits corresponding to the peripheral device protection setting registers.

Cautions 1. However, be sure to fix the bits on the PPSn corresponding to the peripheral device protection setting registers to 0.

2. The bits on PPPn, PPVn, and PPTn corresponding to the peripheral device protection setting registers may be fixed to any value or could be variable bits. However, if a peripheral device other than an OS peripheral device is indicated by these bits, there is no guarantee that the OS (and programs similar to it) will offer protection functions.

7.10 Special Peripheral Device Access Instruction

7.10.1 SYSCALL instruction

The SYSCALL instruction is used to call a service supplied by a management program such as an OS.

The service is a trusted program and the address table to branch to the service is also trusted. Therefore, peripheral device protection is not applied to the peripheral device access by the SYSCALL instruction even if the PSW.PP bit is set (1).

Consequently, the PPI exception is never detected while the SYSCALL instruction is executed.

Because the exception is not detected, peripheral device protection and the SYSCALL instruction can also be used, for example, when test code is placed in the peripheral device area during debugging.

CHAPTER 8 TIMING SUPERVISION FUNCTION

Timing supervision function is performed to manage times such as program execution time and to prevent the CPU execution time from being excessively monopolized.

The timing supervision function provides the following six types of supervision functions by operating six counters each having the same function in accordance with a specific operation method.

- Runtime supervision function
- Deadline supervision function
- Resource supervision function
- Global interrupt lock supervision function
- Software interrupt lock supervision function
- Supervising number of times of interrupt arrival

Each counter has a 28-bit count width and counts in each clock cycle of the CPU. It also has a divided count function of 1 to 1024 and a count range of up to 1.34 seconds at an accuracy of the CPU clock cycle, or up to 1374 seconds (about 23 minutes) at an accuracy 1024 times the CPU clock cycle (where the CPU frequency is 200 MHz).

If violation is detected as a result of supervising the CPU status by each counter in accordance with setting, a TSI exception occurs. The TSI exception is defined as an FE level exception, and can forcibly start exception processing even while an ordinary user application is in a critical section (period during which the PSW.ID bit is 1 and interrupts are not acknowledged). For the TSI exception, refer to **8.5 Handling of TSI Exception**.

8.1 Register Set

Table 8-1 lists the peripheral device registers related to the timing supervision function.

All registers related to timing supervision function are placed in the peripheral device register area. The base address for the timing supervision function register set is defined by hardware and is FFFF5000H for the V850E2M CPU.

Table 8-1. Timing Supervision Function Register Set

Register Name	Address	Accessible Size				Initial Value
		1	8	16	32	
TSEC	00H	√	√	√	√	00000000H
TSECR	04H		√	√	√	00000000H
TSCCFGn	20H + (n*10H) + 0H		√	√	√	00000000H
TSCCNTn	20H + (n*10H) + 4H				√	Refer to **Table 8-2.**
TSCCMPn	20H + (n*10H) + 8H				√	Refer to **Table 8-2.**
TSCRLDn	20H + (n*10H) + CH				√	Refer to **Table 8-2.**

Remark n = 0 to 5

The names, addresses, and initial values of the timing supervision function counter registers for the V850E2M CPU (the TSCCFGn, TSCCNTn, TSCCMPn, and TSCRLDn registers) are shown in Table 8-2.

Table 8-2. Timing Supervision Function Cunter Register of V850E2M CPU

Register Name	Address	Initial Value	TSU counter
TSCCFG0	FFFF5020H	00000000H	TSU counter 0
TSCCNT0	FFFF5024H	00000000H	
TSCCMP0	FFFF5028H	00000000H	
TSCRLD0	FFFF502CH	00000000H	
TSCCFG1	FFFF5030H	00000000H	TSU counter 1
TSCCNT1	FFFF5034H	00000000H	
TSCCMP1	FFFF5038H	00000000H	
TSCRLD1	FFFF503CH	00000000H	
TSCCFG2	FFFF5040H	00000000H	TSU counter 2
TSCCNT2	FFFF5044H	00000000H	
TSCCMP2	FFFF5048H	00000000H	
TSCRLD2	FFFF504CH	00000000H	
TSCCFG3	FFFF5050H	00000000H	TSU counter 3
TSCCNT3	FFFF5054H	00000000H	
TSCCMP3	FFFF5058H	00000000H	
TSCRLD3	FFFF505CH	00000000H	
TSCCFG4	FFFF5060H	00000000H	TSU counter 4
TSCCNT4	FFFF5064H	00000000H	
TSCCMP4	FFFF5068H	00000000H	
TSCRLD4	FFFF506CH	00000000H	
TSCCFG5	FFFF5070H	00000000H	TSU counter 5
TSCCNT5	FFFF5074H	00000000H	
TSCCMP5	FFFF5078H	00000000H	
TSCRLD5	FFFF507CH	00000000H	

8.1.1 TSEC – Controlling timing supervision

This register has function bits that control the timing supervision function. It can be accessed in units of 32, 16, 8, or 1 bit.

Be sure to set bits 31 to 1 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
TSEC	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	TSECESUP	Initial value 00000000H

Bit Position	Bit Name	Description
0	TSECESUP	While the TSECESUP bit is set, the timing supervision counter is kept from making a request. The TSECESUP bit is automatically set (1) as soon as the TSI exception has been acknowledged, and report of the TSI exception is controlled. As a result, starting another TSI exception is prevented while the first TSI exception is being processed. A TSI exception that occurs while the TSECESUP bit is held pending. After the TSECESUP bit is cleared, the TSI exception is not lost but requested again. Be sure to clear (0) the TSECESUP bit before completion of TSI exception processing.

8.1.2 TSECR – Timing supervision exception cause

This is an exception cause register of timing supervision exception (TSI exception). It can be accessed in units of 32, 16, or 8 bits.

Be sure to set bits 31 to 6 to “0”.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
TSECR	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	TSECRES5	TSECRES4	TSECRES3	TSECRES2	TSECRES1	TSECRES0	Initial value 00000000H

Bit Position	Bit Name	Description
5	TSECRES5	This is the exception cause bit of timing supervision counter 5. It is a mapping bit of the TSCCFG5.TSCCFG5ES bit.
4	TSECRES4	This is the exception cause bit of timing supervision counter 4. It is a mapping bit of the TSCCFG4. TSCCFG4ES bit.
3	TSECRES3	This is the exception cause bit of timing supervision counter 3. It is a mapping bit of the TSCCFG3. TSCCFG3ES bit.
2	TSECRES2	This is the exception cause bit of timing supervision counter 2. It is a mapping bit of the TSCCFG2. TSCCFG2ES bit.
1	TSECRES1	This is the exception cause bit of timing supervision counter 1. It is a mapping bit of the TSCCFG1. TSCCFG1ES bit.
0	TSECRES0	This is the exception cause bit of timing supervision counter 0. It is a mapping bit of the TSCCFG0. TSCCFG0ES bit.

8.1.3 TSCCFGn – Setting of timing supervision function counter n (n = 0 to 5)

This register is used to set counter n for timing supervision (n = 0 to 5). It can be accessed in units of 32, 16, or 8 bits. Be sure to set 0 to bits 31 to 27, 23, 22, 19, 15 to 11, and 7 to 4.

The TSCCFGn register allocates each of its fields at a 1-byte boundary. Therefore, to access any of the fields, a partial operation can be performed by accessing a byte (for example, to clear only a status, write a byte to an address of offset + 1H).

(1/3)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TSCCFGn	0	0	0	0	0	TSCCFGnRES			0	0	TSCCFGnCTM		0	TSCCFGnARM	TSCCFGnEXM	TSCCFGnUDM
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	TSCCFGnOS	TSCCFGnES	TSCCFGnVS	0	0	0	0	TSCCFGnRLD	TSCCFGnFE	TSCCFGnEIACT	TSCCFGnACT

Initial value 00000000H

Bit Position	Bit Name	Description
26 to 24	TSCCFGnRES	These bits select the resolution of the timing supervision counter. 000: Counts every 1 clock cycle. 001: Counts every 4 clock cycles. 010: Counts every 16 cock cycles. 011: Counts every 64 clock cycles. 100: Counts every 256 clock cycles. 101: Counts every 1024 clock cycles. 110: RFU 111: RFU
21, 20	TSCCFGnCTM	These bits select an operation mode. 00: Normal mode 01: Global interrupt lock supervision mode 10: Runtime supervision mode 11: RFU • **Normal mode** The timing supervision counter operates as an ordinary counter in this mode. The counter is not automatically started or stopped. • **Global interrupt lock supervision mode** In this mode, the value of the TSCRLDn.TSCRLDnVAL bit is automatically transferred to the TSCCNTn.TSCCNTnVAL bit when the PSW.ID bit changes from 0 to 1. After that, TSCCFGnACT is updated to 1, and the counter starts counting. TSCCFGnACT is automatically updated to 0 and the counter is stopped when the PSW.ID bit changes from 1 to 0. • **Runtime supervision mode** In this mode, the value of TSCCFGnACT is automatically transferred to TSCCFGnEIACT/ TSCCFGnFEACT when an EI level or FE level exception is acknowledged. TSCCFGnACT is updated to 0 and the counter is stopped. When the EIRET/FERET instruction is executed, the value of TSCCFGnEIACT/ TSCCFGnFEACT bit is automatically transferred to TSCCFGnACT. If TSCCFGnACT is set to 1 as a result, counting is resumed.

(2/3)

Bit Position	Bit Name	Description
18	TSCCFGnARM	Enables or disables auto reloading. This bit selects whether the value of the TSCRLDn. TSCRLDnVAL bit is to be transferred to the TSCCNTn.TSCCNTnVAL bit or not if violation is detected (if TSCCNTn.TSCCNTnVAL and TSCCMPn.TSCCMPnVAL bits match). 0: Does not execute auto reloading (disabled). 1: Executes auto reloading (enabled).
17	TSCCFGnEXM	This bit selects an exception mode. It selects whether the TSI exception is reported if violation is detected (if TSCCNTn.VAL and TSCCMPn.VAL bits match). 0: Mode not to report TSI exception. TSI exception is not reported. 1: Mode to report TSI exception. TSI exception is reported.
16	TSCCFGnUDM	This bit selects a counting direction of the counter. 0: Counter operates as an up counter (addition). 1: Counter operates as a down counter (subtraction).
10	TSCCFGnOS[Note]	This is the overflow bit of the counter. It is set if the TSCCNTn.VAL bit overflows or underflows. 0: This counter does not overflow or underflow. 1: This counter overflows or underflows.
9	TSCCFGnES[Note]	This is an exception cause bit. It is set if the TSI exception is acknowledged when the exception mode is the “mode to report TSI exception (TSCCFGnEXM = 1)”. Be sure to clear (0) this bit before completion of the TSI exception processing. 0: This counter is not the cause of the TSI exception currently under processing. 1: This counter is the cause of the TSI exception currently under processing.
8	TSCCFGnVS[Note]	This bit detects violation. 0: Violation is not detected. 1: Violation is detected. This bit is set if violation is detected (if TSCCNTn.VAL and TSCCMPn.VAL match). The TSI exception is requested if the exception mode is the “mode to report TSI exception (TSCCFGnEXM = 1)” and this bit is set. This bit is cleared if the TSI exception is acknowledged while the counter is requesting the TSI exception. If this bit is cleared by software processing, the request for the TSI exception by the counter can be canceled.
3	TSCCFGnRLD	This is a reload command bit. If this bit is set (1), the TSCTSCCFGnRLDn.VAL bit is transferred to the TSCCNTn.VAL bit. This bit is always 0 when read.

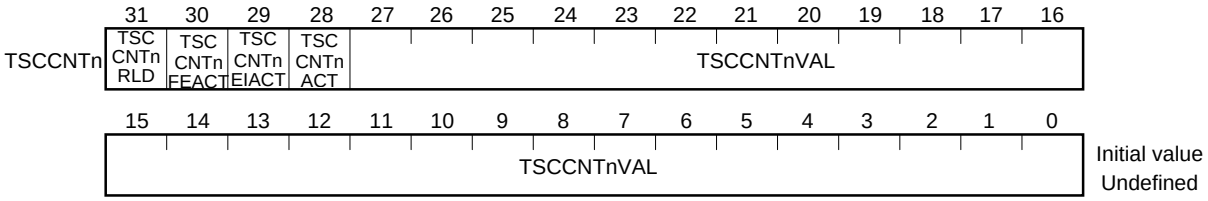
Note The TSCCFGnOS, TSCCFGnVS, and TSCCFGnES bits can only be cleared (0) by a write operation. They cannot be set (1).

(3/3)

Bit Position	Bit Name	Description
2	TSCCFGnFEACT	This bit saves the content of the TSCCFGnACT bit when an FE level exception is acknowledged. • **In runtime supervision mode** This bit transfers the content of the TSCCFGnACT bit to the TSCCFGnFEACT bit and clears (0) the TSCCFGnACT bit when an FE level exception is acknowledged. It transfers the content of the TSCCFGnFEACT bit to the TSCCFGnACT bit when the FERET instruction is executed. • **In mode other than runtime supervision mode** The content of the TSCCFGnFEACT bit is not transferred to the TSCCFGnACT bit even when the FERET instruction is executed.
1	TSCCFGnEIACT	This bit saves the content of the TSCCFGnACT bit when an EI level exception is acknowledged. • **In runtime supervision mode** This bit transfers the content of the TSCCFGnACT bit to the TSCCFGnEIACT bit and clears (0) the TSCCFGnACT bit when an EI level exception is acknowledged. It transfers the content of the TSCCFGnEIACT bit to the TSCCFGnACT bit when the EIRET instruction is executed. • **In mode other than runtime supervision mode** The content of the TSCCFGnEIACT bit is not transferred to the TSCCFGnACT bit even when the EIRET instruction is executed.
0	TSCCFGnACT	This bit indicates that the counter is operating or stopped. By setting (1) this bit, the counter operation can be started. By clearing (0) it, the counter can be stopped. 0: Stopped 1: Operating The value of this bit automatically changes in the runtime supervision mode and global interrupt lock supervision mode. For details, refer to the description of the TSCCFGnCTM bit. This bit is fixed to 0 if the MPM.MPE bit is 0. As a result, the timing supervision counter does not operate, and does not detect violation and request the TSI exception.

8.1.4 TSCCNTn – Count value of timing supervision counter n (n = 0 to 5)

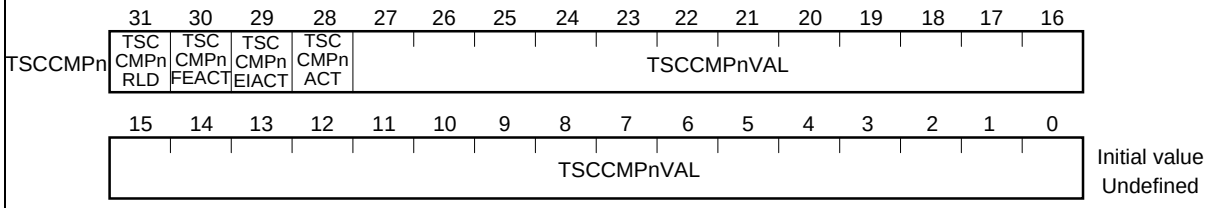
This is the count register of timing supervision counter n (n = 0 to 5). It can be accessed in 32-bit units.



Bit Position	Bit Name	Description
31	TSCCNTn RLD	This is a reload command bit. It is a mapping bit of the TSCCFGn.TSCCFGnRLD bit.
30	TSCCNTn FEACT	This bit saves the content of the TSCCFGnACT bit when an FE level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnFEACT bit.
29	TSCCNTn EIACT	This bit saves the content of the TSCCFGnACT bit when an EI level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnEIACT bit.
28	TSCCNTn ACT	This bit indicates whether the counter is operating or stopped. It is a mapping bit of the TSCCFGn.TSCCFGnACT bit.
27 to 0	TSCCNTn VAL	These bits indicate the current value of the counter. Violation is detected if TSCCNTn.TSCCNTnVAL and TSCCMPn. TSCCMPnVAL match.

8.1.5 TSCCMPn – Comparison value of timing supervision counter n (n = 0 to 5)

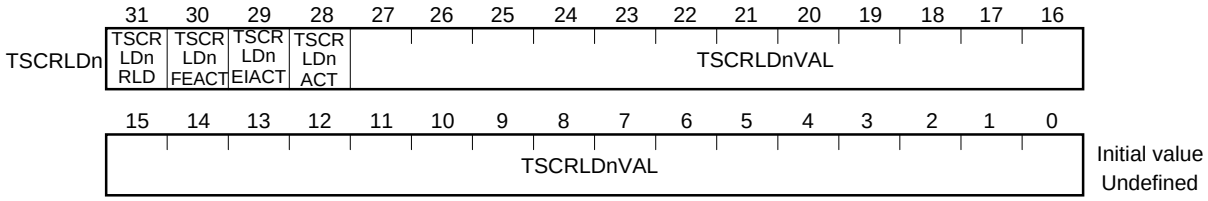
This is a compare register of the timing supervision counter (n = 0 to 5). It can be accessed in 32-bit units.



Bit Position	Bit Name	Description
31	TSCCMPn RLD	This is a reload command bit. It is a mapping bit of the TSCCFGn.TSCCFGnRLD bit.
30	TSCCMPn FEACT	This bit saves the content of the TSCCFGnACT bit when an FE level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnFEACT bit.
29	TSCCMPn EIACT	This bit saves the content of the TSCCFGnACT bit when an EI level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnEIACT bit.
28	TSCCMPn ACT	This bit indicates whether the counter is operating or stopped. It is a mapping bit of the TSCCFGn.TSCCFGnACT bit.
27 to 0	TSCCMPn VAL	These bits indicate the comparison value of the counter. Violation is detected if TSCCNTn.TSCCNTnVAL and TSCCMPn.TSCCMPnVAL match.

8.1.6 TSCRLDn – Reload value of timing supervision counter n (n = 0 to 5)

This is a reload register of timing supervision counter n (n = 0 to 5). It can be accessed in 32-bit units.



Bit Position	Bit Name	Description
31	TSCRLDnRLD	This is a reload command bit. It is a mapping bit of the TSCCFGn.TSCCFGnRLD bit.
30	TSCRLDnFEACT	This bit saves the content of the TSCCFGnACT bit when an FE level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnFEACT bit.
29	TSCRLDnEIACT	This bit saves the content of the TSCCFGnACT bit when an EI level exception is acknowledged. It is a mapping bit of the TSCCFGn.TSCCFGnEIACT bit.
28	TSCRLDnACT	This bit indicates whether the counter is operating or stopped. It is a mapping bit of the TSCCFGn.TSCCFGnACT bit.
27 to 0	TSCRLDnVAL	These bits indicate the value to be transferred from the TSCRLDn.TSCRLDnVAL bit to the TSCCNTn.TSCCNTnVAL bit during reloading. Reloading is performed in the following cases. • When the TSCCFGn.TSCCFGnRLD bit is set (1) • If violation is detected when the TSCCFGn.TSCCFGnARM bit is set (1) • If the PSW.ID bit changes from 0 to 1 when the TSCCFGn.TSCCFGnCTM bits are 01 (global interrupt lock supervision mode)

8.2 Counter Functions

Six identical counters are provided for monitoring the timing. Each counter has a 28-bit count width, is incremented on a CPU clock cycle basis, and can be set to one of six resolutions in the range from 1 to 1,024. During timing supervision function, violations are detected when a counter value matches a pre-specified comparison value. When a timing supervision function violation is detected, a TSI exception request is reported to the CPU in accordance with settings.

8.2.1 Resolution

The resolution of each one count of the counter is set by the TSCCFGn.TSCCFGnRES bit. The resolution indicates the number of clock cycles for one counter (TSCCNTn.TSCCNTnVAL). For example, if the resolution is 1, the count value 1 indicates 1 clock. If the resolution is 1024, the count value 1 indicates 1024 clocks.

Each counter can separately select the resolutions in Table 8-2.

Table 8-2. Resolutions

Value of TSCCFGn.TSCCFGnRES Bit	Operation
000	Counts every 1 clock.
001	Counts every 4 clocks.
010	Counts every 16 clocks.
011	Counts every 64 clocks.
100	Counts every 256 clocks.
101	Counts every 1024 clocks.
Others	RFU

Caution If the resolution is other than 1, a count error of the number of clocks up to two times higher than the resolution occurs. Note this error when using a low resolution.

8.2.2 Counting direction

An up counter mode or a down counter mode can be selected by using the TSCCFG.TSCCFGnUDM bit. The count value can be incremented or decremented in 1 unit of the count operation.

(1) Up counter mode

Increments the count value (TSCCNTn.TSCCNTnVAL) each time the counter counts.

(2) Down counter mode

Decrements the count value (TSCCNTn.TSCCNTnVAL) each time the counter counts.

8.2.3 Exception mode

The TSCCFGn.TSCCFGnEXM bit can be used to select whether the TSI exception is to be reported or not if the count value (TSCCNTn.TSCCNTnVAL) matches a specified comparison value (TSCCMPn.TSCCMPnVAL) and violation is detected. A mode to report the TSI exception is used for the normal timing supervision usage. However, when the counter is used for other usage, a mode not to report the TSI exception may be used depending on the purpose.

(1) Mode not to report TSI exception

This mode does not report the TSI exception if the counter detects violation (if the count value matches the comparison value).

(2) Mode to report TSI exception

This mode reports the TSI exception if the counter detects violation (if the count value matches the comparison value).

8.2.4 Auto reloading

The TSCCFGn.TSCCFGnARM bit can be used to select whether a reloading value (TSCRLDn.TSCCNTnVAL) is to be automatically transferred to the counter if the count value (TSCCNTn.TSCCNTnVAL) matches a specified comparison value (TSCCMPn.TSCCMPnVAL) and violation is detected. Enable auto reloading to immediately start the next counting as soon as violation has been detected, such as when periodically measuring time.

(1) When auto reloading is disabled

A value is not automatically reloaded to the counter when the counter detects violation (when the count value matches the comparison value). The counter continues counting from the current value.

(2) When auto reloading is enabled

A value is automatically reloaded to the counter when the counter detects violation (when the count value matches the comparison value). The counter continues counting from the reloaded value.

8.2.5 Counter mode

The counter has special counter modes that are used for specific purposes. Global interrupt lock supervision mode and runtime supervision mode, as well as the normal counter mode, can be selected by using the TSCCFGn.TSCCFGnCTM bit.

(1) Normal mode

The counter operates as an ordinary counter in this mode. It is not automatically started or stopped, and is controlled by software's manipulation of the TSCCFGnACT bit. Be sure to set this mode when global interrupt lock supervision or runtime supervision is not performed.

Caution	**To stop the counter operating in the global interrupt lock supervision mode or runtime supervision mode, be sure to change the mode to normal mode. If the counter remains in the global interrupt lock supervision mode or runtime supervision mode, its operation may automatically be resumed by execution of another program.**

(2) Global interrupt lock supervision mode

Set this mode to perform global interrupt lock supervision. Global interrupt lock supervision supervises the user application so that it does not illegally extend a critical section and hinder the operations of the other programs of the CPU. As a global interrupt lock period, time during which the PSW.ID bit is set (1) is supervised, and violation is detected if a specified time is exceeded.

Because the user application can start or end a critical section by using the DI or EI instruction without the need of an extra procedure, the status of the PSW.ID bit is supervised and the counter is automatically started, stopped, or reloaded.

The operations in the global interrupt lock supervision mode are listed below.

Table 8-3. Operations in Global Interrupt Lock Supervision Mode

Bit	When PSW.ID Changes from 0 to 1	When PSW.ID Changes from 1 to 0
TSCCFGnACT	Updated to 1	Updated to 0
TSCCNTn.TSCCNTnVAL	Value of TSCRLDn.VAL is transferred.	Not updated

(3) Runtime supervision mode

Set this mode to perform runtime supervision. In the runtime supervision mode, the counter is automatically stopped when other task is started because of generation of an exception, in order to strictly manage the time budget of the program currently under execution. At this time, the content of the bit indicating the operation status of the counter (TSCCFGn.TSCCFGnACT bit) is saved for each exception level (TSCCFGn.TSCCFGnEIACT/TSCCFGnFEACT). When the return instruction (EIRET/FERET) is executed, the saved content is restored to the TSCCFGnACT bit, so that the counting operation is automatically resumed immediately after execution has returned from the exception processing. Appropriately save and restore the counter value at the beginning and end of each exception.

The operations in the runtime supervision mode are shown in Table 8-4.

Table 8-4. Operations in Runtime Supervision Mode

Bit	When EI level Exception Is Acknowledged	When EIRET Instruction Is Executed	When FE level Exception Is Acknowledged	When FERET Instruction Is Executed
TSCCFGnACT	Updated to 0	Value of TSCCFGnEIACT is transferred	Updated to 0	Value of TSCCFGnFEACT is transferred
TSCCFGnEIACT	Value of TSCCFGnACT is transferred	Not updated	Not updated	Not updated
TSCCFGnFEACT	Not updated	Not updated	Value of TSCCFGnACT is transferred	Not updated
TSCCNTn. TSCCNTnVAL	Not updated	Not updated	Not updated	Not updated

Caution None of the above bits is updated when the CALLT or CTRET instruction is executed.

8.3 Operation Modes of Counter and CPU

Each counter of the timing supervision function performs the following operations depending on the operation status of each CPU.

Table 8-5. Operation Modes of Counter and CPU

V850E2M Operation Mode	System Clock	MPM.MPE	Program under Execution	TSCCFGnACT Bit	Operation Status of TSU Counter
Normal/HALT status[Note1]	Supplied	0	Don't care	Fixed to 0	Counter stop when TSCCFGnACT = 0
		1	Normal (task)/ EI level exception/ FE level exception	0	Counter stop when TSCCFGnACT = 0
				1	Counter opereiton when TSCCFGnACT = 0
Standby mode	Stopped	Previous value retained	Don't care	Previous value retained[Note 2]	Stops because supply of CPU clock is stopped.

Notes 1. The HALT status is released as a result of occurrence of the TSI exception, and the exception is acknowledged in accordance with the acknowledgment condition.

2. Stop the timing supervision function by clearing (0) the TSCCFGnACT bit in advance before stopping the clock.

8.4 Detecting Violation

Violation related to the timing supervision function is detected when the count value (TSCCNTn.TSCCMPnVAL) matches the comparison value (TSCCMPn.TSCCMPnVAL). When violation has been detected, the violation detection bit (TSCCFGn.TSCCFGnVS) of the corresponding counter is set (1).

8.4.1 Identifying violation cause

Each counter of the timing supervision function does not have information that indicates the cause of violation. Manage the contents supervised by each counter by software and perform appropriate violation processing.

8.5 Handling of TSI Exception

The timing supervision function reports the TSI exception in accordance with setting if it detects a violation.

8.5.1 Reporting TSI exception

If the exception mode is the “mode to report the TSI exception (TSCCFGn.TSCCFGnEXM = 1)” when a violation is detected, the TSI exception is immediately reported to the CPU. The TSI exception reported is acknowledged immediately if PSW.NP is cleared (0).

As soon as the TSI exception has been acknowledged, the violation detection bit (TSCCFGn.TSCCFGnVS) corresponding to the counter reporting the TSI exception is saved to the exception cause bit (TSCCFGn.TSCCFGnES). Therefore, the TSCCFGnES bit of that counter is set (1) and TSCCFGnVS bit is cleared (0).

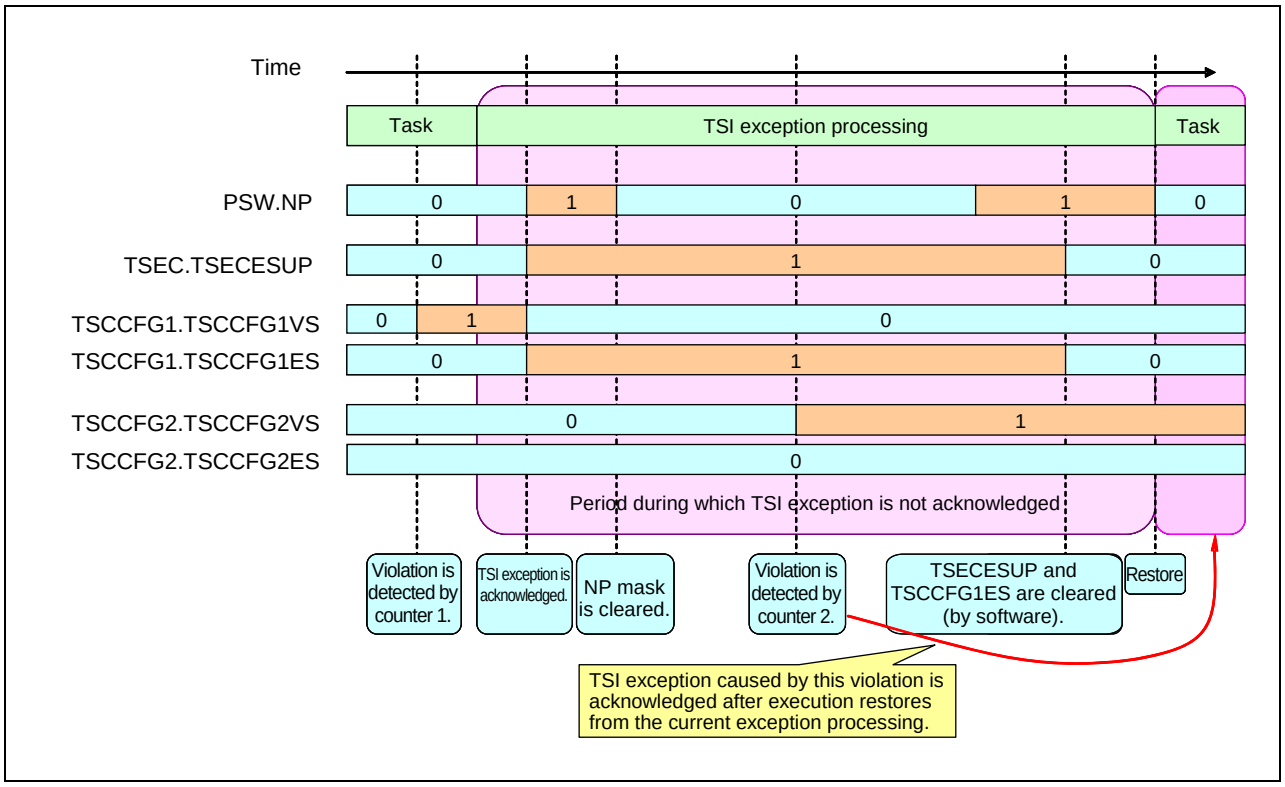
Because the content of the TSCCFGnVS bit of the counter that reports the TSI exception is saved to the TSCCFGnES bit as soon as the TSI exception has been acknowledged, new timing supervision violation that is detected while the timing supervision counter continues operating during TSI exception processing and the violation that has caused the ongoing TSI exception processing can be distinguished from each other.

Caution	**The TSCCFGnVS bit of the counter operating in the ”mode not to report the TSI exception” is not changed when the TSI exception is acknowledged.**

As soon as the TSI exception has been acknowledged, the TSEC.TSECESUP bit is also set (1), suppressing reporting of the TSI exception under processing. As a result, occurrence of multiple exceptions which takes place if the TSI exception is reported again when the other counter detects violation while the first TSI exception is being processed is prevented.

Caution	**To acknowledge the next TSI exception, be sure to clear the TSCCFGn.TSCCFGnES and TSEC.TSECESUP bits of counters for which violation processing was performed to 0 before the TSI exception processing finishes. If the bits are not cleared before the system returns from this exception processing, it will not be possible to acknowledge the next TSI exception.**

Figure 8-1. Report of TSI Exception



8.5.2 Identifying exception cause

Each counter of the timing supervision function does not have information that indicates the cause of exception. Manage the contents supervised by each counter by software and perform appropriate exception processing.

8.5.3 Canceling TSI exception

If a program that is subject to measurement by a counter is terminated as a result of processing other exception, and if the TSI exception is reported by that counter and has not been acknowledged yet, cancel the TSI exception in the following procedure. For the situation where this processing is necessary, refer to **6.3 Exception Management** in **PART 2**.

<1> Set (1) the PSW.NP bit.
<2> Clear (0) the TSCCFGn.TSCCFGnACT bit of the counter in question.
<3> Change the mode of the counter to the normal mode by using the TSCCFGn.TSCCFGnCTM bit.
<4> Clear (0) the TSCCFGn.TSCCFGnVS bit of the counter.

Remark The TSCCFGn register can be accessed in 32-bit units, and <2> to <4> can be performed at the same time.

8.6 Setting Counter Corresponding to Each Supervision Function

Set the operation of the counter in accordance with each supervision content. This section shows a recommended example of setting the register corresponding to each supervision function. Appropriately adjust the set value in accordance with each OS (and programs similar to it) or purpose for the actual operation.

8.6.1 Global interrupt lock supervision

Global interrupt lock supervision supervises the PSW.ID bit so that the critical section of the user application does not continue, exceeding specified time. If the ID bit remains 1 longer than specified time, exception processing is started. Consequently, the system can be prevented from being deadlocked due to a design error of the user application.

Table 8-6. Global Interrupt Lock Supervision

Register	Bit	Example of Recommended Setting
TSCCFGn	TSCCFGnUDM	Don't care
	TSCCFGnEXM	1 (TSI exception occurs.)
	TSCCFGnARM	0 (Auto reloading is not performed.)
	TSCCFGnCTM	1 (Global interrupt lock supervision mode)
	TSCCFGnRES	Don't care
TSCCNTn	–	Don't care
TSCCMPn	–	Maximum permissible lock time for up counter 0 for down counter
TSCRLDn	–	0 for up counter Maximum permissible lock time for down counter

8.6.2 Runtime supervision

Runtime supervision is to supervise the execution time budget given to a task under execution and perform exception processing when the budget is used up. If a preemption occurs due to interrupt, the counter is automatically stopped or resumed when an interrupt is acknowledged or execution restores from the interrupt, so that the time of the CPU clock cycle is accurately and delicately supervised.

To supervise part of time of the task under execution, appropriately select one of the set values listed in Table 8-7 in accordance with the purpose.

To measure time during which a task acquires a resource, for example, make the setting in Table 8-7, start the task by software when the resource is acquired, and stop the task by software when the resource is released. In this way, the execution time during which the task acquires the resource can be accurately supervised even if a preemption due to an interrupt occurs while the task is acquiring the resource.

Table 8-7. Runtime Supervision

Register	Bit	Example of Recommended Setting
TSCCFGn	TSCCFGnUDM	Don't care
	TSCCFGnEXM	1 (TSI exception occurs.)
	TSCCFGnARM	0 (Auto reloading is not performed.)
	TSCCFGnCTM	2 (Runtime supervision mode)
	TSCCFGnRES	Don't care
TSCCNTn	–	0 for up counter Execution time budget for down counter
TSCCMPn	–	Execution time budget for up counter 0 for down counter
TSCRLDn	–	Not used

8.6.3 Supervising number of times of interrupt reaches a specific value

Supervising the number of times of interrupt arrival is to supervise the number of times a specific interrupt occurs in a specific frame time. If the interrupt has occurred more than the specified number of times, it is disabled from being acknowledged and the processing time of the CPU is released to other programs. The disabled interrupt is enabled to be acknowledged again each time a specific period has elapsed. As a trigger of this periodic operation, a counter of the timing supervision function is used. Because the counter operates as a periodic counter, the counter that is used to supervise the number of times the interrupt reaches a specific value uses the auto reloading feature, and when the count value reaches a specific value, an exception is requested and the counter starts counting the next time frame.

To start a supervising program at specific intervals, change the set value shown in Table 8-8 appropriately in accordance with the purpose like when supervising the number of times interrupt reaches a specific value.

Table 8-8. Supervising Number of Times of Interrupt Arrival

Register	Bit	Example of Recommended Setting
TSCCFGn	TSCCFGnUDM	Don't care
	TSCCFGnEXM	1 (TSI exception occurs.)
	TSCCFGnARM	1 (Auto reloading is performed.)
	TSCCFGnCTM	0 (Normal mode)
	TSCCFGnRES	Don't care
TSCCNTn	–	Don't care (TSCCFGnACT bit is set as soon as reloading is executed to start counting.)
TSCCMPn	–	Unit frame time to be supervised for up counter 0 for down counter
TSCRLDn	–	0 for up counter Unit frame time to be supervised for down counter

8.6.4 Supervising time lapse

To supervise a certain time from occurrence of an event that serves as a reference, basically set the counter as shown in Table 8-9. By making this setting, basic supervision of the counter can be performed by starting and stopping the counter by software.

For example, to supervise deadline, acquisition time of a specific resource, or the time during which a specific interrupt source is masked, appropriately change the set value in Table 8-9.

Table 8-9. Supervising Time Lapse

Register	Bit	Example of Recommended Setting
TSCCFGn	TSCCFGnUDM	Don't care
	TSCCFGnEXM	1 (TSI exception occurs.)
	TSCCFGnARM	0 (Auto reloading is not performed.)
	TSCCFGnCTM	0 (Normal mode)
	TSCCFGnRES	Don't care
TSCCNTn	–	0 for up counter Target time for down counter
TSCCMPn	–	Target time for up counter 0 for down counter
TSCRLDn	–	Not used

CHAPTER 9 PROCESSOR PROTECTION EXCEPTION

This chapter describes different types of processor protection violations and exceptions. For details about processing for each exception, see **CHAPTER 6 EXCEPTIONS**, which is in PART 2.

9.1 Types of Violations

The V850E2M CPU detects violation in accordance with the setting of each protection function and, as necessary, generates an exception defined by each protection function. This section explains in detail the relationship between violations and exceptions.

The following five types of violations are detected in accordance with the setting defined by the processor protection function.

- System register protection violation
- Execution protection violation
- Data protection violation
- Peripheral device protection violation
- Timing supervision violation

9.1.1 System register protection violation

This violation is detected if a system register is illegally accessed. No exception occurs even when this violation is detected. For the processing to be performed if this violation is detected, refer to **5.5 Operation Method**.

9.1.2 Execution protection violation

This violation may be detected when an instruction is executed. The execution protection violation is detected if an attempt is made to execute an instruction allocated in an area of the program area where execution is not enabled.

If the execution protection violation is detected, the MIP exception is always generated.

9.1.3 Data protection violation

This violation may be detected when an instruction accesses data. It is detected if a memory access instruction reads or writes data from or to an area of the data area that is not enabled to be accessed.

If the data protection violation is detected, the MDP exception always occurs.

9.1.4 Peripheral device protection violation

This violation may be detected when an instruction accesses a data. It is detected if a memory access instruction is enabled by the memory protection function and an operation not permitted by an access control based on the peripheral device protection setting defined for each system is performed.

If the peripheral device protection violation is detected, the PPI exception occurs in accordance with the setting.

9.1.5 Timing supervision violation

This violation is detected by the timing supervision function. It is detected when each counter of the timing supervision function counts in accordance with the operation setting and satisfies a specified status.

If the timing supervision violation is detected, the TSI exception occurs in accordance with the setting.

9.2 Types of Exceptions

The V850E2M CPU generates four types of exceptions defined by the processor protection function. If an exception occurs, execution branches to the exception handler (00000030H) and violation information defined for each source is stored in a register.

9.2.1 MIP exception

This exception occurs when an execution protection violation has been detected. This exception is a precise exception that occurs if execution of an instruction allocated at an address that is not permitted to be accessed is attempted. It can also be resumed and restored because the original processing can be correctly continued from the instruction that has generated this exception.

9.2.2 MDP exception

This exception occurs if a data protection violation is detected. This exception is a precise exception that occurs if data allocated at an address not permitted to be accessed is read or written. It can also be resumed and restored because the original processing can be correctly continued from the instruction that has caused this exception.

9.2.3 PPI exception

This exception occurs if peripheral device protection violation is detected. It is an imprecise exception that may occur lagging behind the event that has caused the violation, and that is held pending when the PSW.NP bit is set (1). This exception can be resumed but cannot be restored, which makes it difficult to continue execution from the instruction that has caused the violation, because the exception processing is performed after the instruction that has caused the violation has already been completed and the subsequent instructions have been executed.

A PPI exception is synchronized with exceptions when they are acknowledged or returned from, and the synchronization is performed by the exception synchronization instruction SYNCE. For details, see **6.3.1 Synchronizing exception when exception is acknowledged or when execution returns** and **6.3.2 Exception synchronization instruction, which are in PART 2.**

9.2.4 TSI exception

This exception occurs if a timing supervision violation is detected. The timing supervision violation is caused by a counter that operates with the clock input of the CPU. Therefore, it occurs at an unspecific timing, like interrupts. For this reason, occurrence of the exception is held pending if the PSW.NP bit is set (1) in order to prevent the exception from being acknowledged when it is impossible to acknowledge exceptions. A TSI exception is synchronized with exceptions when they are acknowledged or returned from. For details, see **6.3.1 Synchronizing exception when exception is acknowledged, which is in PART 2**.

9.3 Identifying Violation Cause

If protection violation is detected, an exception cause that indicates which exception, MIP, MDP, PPI, or TSI, has caused a branch to the exception handler is stored in the system register FEIC of the CPU bank. Because the exception handler address is shared with the other exceptions, branching processing by the exception cause is necessary. As shown in Table 9-1, auxiliary information indicating the cause of each exception is stored in a specific system register of the MPV bank.

Table 9-1. Identifying Violation Cause

	FEIC	Exception Type	Violation	Violation Information Register
Violation related to processor protection	00000430H	MIP exception	Execution protection violation	VMECR, VMADR, VMTID
	00000431H	MDP exception	Data protection violation	VMECR, VMADR, VMTID
	00000432H	PPI exception	Peripheral device protection violation	VPNECR, VPNADR, VPNTID
	00000433H	TSI exception	Timing supervision violation	TSECR
Other exceptions	–	–	–	–

9.3.1 MIP exception

00000430H is stored in the FEIC register. The contents of the TID register when this exception occurs are stored in the VMTID register, and the PC of the instruction that has caused the exception is stored in the VMADR register. The VMX bit of the VMECR register is set (1), and the other bits are cleared (0).

Because the MIP and MDP exceptions never occur at the same time, the MIP exception share the violation information registers (VMECR, VMTID, and VMADR registers) with the MDP exception.

Table 9-2. VMECR Set Value When MIP Exception Occurs

Register	VMECR						
Bit number	6	5	4	3	2	1	0
Bit name	VMMS	VMRMW	VMS	VMW	VMR	VMX	-
All instructions	0	0	0	0	0	1	0

9.3.2 MDP exception

00000431H is stored in the FEIC register. The contents of the TID register when this exception occurs are stored in the VMTID register, and the address of a memory access that has caused the exception is stored in the VMADR register.

In accordance with the contents of the violation detected, the VMR, VMW, and VMS bits of the VMECR register are set (1) and the VMX bit is cleared (0).

Because the MIP and MDP exceptions never occur at the same time, the MDP exception shares the violation information registers (VMECR, VMTID, and VMADR registers) with the MIP exception.

Table 9-3. VMECR Set Value When MDP Exception Occurs

Register	VMECR						
Bit number	6	5	4	3	2	1	0
Bit name	VMMS	VMRMW	VMS	VMW	VMR	VMX	–
Read instruction (aligned)Note 1	0	0	0/1Note 5	0	1	0	0
Write instruction (aligned)Note 2	0	0	0/1Note 5	1	0	0	0
Read instruction (misaligned)Note 3	1	0	0/1Note 5	0	1	0	0
Write instruction (misaligned)Note 2	1	0	0/1Note 5	1	0	0	0
CAXI/SET1/NOT1/CLR1 instructionNote 4	0	1	0/1Note 5	0	1	0	0
PREPARE instruction	0	0	1	1	0	0	0
DISPOSE instruction	0	0	1	0	1	0	0

Notes 1. LD/SLD/CALLT/SWITCH/TST1 instruction

2. ST/SST instruction

3. LD/SLD instruction

4. When an instruction that performs a read-modify-write operation is executed, violation occurs only during the read operation because enabling the write operation is checked in the read cycle.

5. 1 if sp indirect access is executed in accordance with the operand specification of an instruction; otherwise, 0.

9.3.3 PPI exception

00000432H is stored in the FEIC register. The task ID when the violation has been detected is stored in the VPNTID register and the address of the memory access that has caused the violation is stored in the VPNADR register. In the VPNECR register, information on the access that has caused the exception and on the peripheral device accessed is stored.

Cautions 1. The PPI exception does not occur when special peripheral device protection violation is detected in the T state. The violation information in that case is stored in the VPTECR, VPTTID, and VPTADR registers, rather than the above VPNECR, VPNTID, and VPNADR registers. For details, refer to 7.4 Peripheral Device Protection Violation in T State.

2. The address saved in the VPNADR register is an address dependent upon the bus system of each product. For correspondence between the address stored in the VPNADR register and the logical address of the architecture, refer to the manual of each product.

9.3.4 TSI exception

00000433H is stored in the FEIC register. The exception cause bit (each bit of the TSECR register or TSCCFGn.TSCCFGnES bit) corresponding to the counter that has detected violation when the TSI exception is acknowledged is set (1).

CHAPTER 10 MEMORY PROTECTION SETTING CHECK FUNCTION

A memory protection settings check is performed in advance to provide service protection by checking whether the data areas for which operations are requested by user applications running on the OS (and programs similar to it) are permitted by the access permissions of the application that requested the service. The OS (and programs similar to it) can check the validity of the parameters for a system service supplied by the user. This check processing can be completed in a shorter time than repeating a reading and comparing area settings by software.

10.1 Register Set

The registers used for the memory protection setting check function are listed below.

Table 10-1. System Register Bank

Group	Processor Protection Function (10H)				
Bank	Processor protection violation (00H)		Processor protection setting (01H)		Software paging (10H)
Bank label	MPV, PROT00		MPU, PROT01		PROT10
Register No.	Name	Function	Name	Function	Name
0	**VSECR**	**System register protection violation cause**	MPM	Setting processor protection operation mode	MPM
1	**VSTID**	**System register protection violation task identifier**	MPC	Specifying processor protection command	MPC
2	**VSADR**	**System register protection violation address**	TID	Task identifier	TID
3	Reserved for function expansion		Reserved for function expansion		VMECR
4	VMECR	Memory protection violation cause			VMTID
5	VMTID	Memory protection violation task identifier			VMADR
6	VMADR	Memory protection violation address	**IPA0L**	**Instruction/constant protection area 0 lower-limit address**	IPA0L
7	Reserved for function expansion		**IPA0U**	**Instruction/constant protection area 0 upper-limit address**	IPA0U
8			**IPA1L**	**Instruction/constant protection area 1 lower-limit address**	IPA1L
9			**IPA1U**	**Instruction/constant protection area 1 upper-limit address**	IPA1U
10			**IPA2L**	**Instruction/constant protection area 2 lower-limit address**	IPA2L
11			**IPA2U**	**Instruction/constant protection area 2 upper-limit address**	IPA2U
12			**IPA3L**	**Instruction/constant protection area 3 lower-limit address**	IPA3L
13			**IPA3U**	**Instruction/constant protection area 3 upper-limit address**	IPA3U
14			**IPA4L**	**Instruction/constant protection area 4 lower-limit address**	IPA4L
15			**IPA4U**	**Instruction/constant protection area 4 upper-limit address**	IPA4U
16			**DPA0L**	**Data protection area 0 lower-limit address (for stack)**	DPA0L
17			**DPA0U**	**Data protection area 0 upper-limit address (for stack)**	DPA0U
18			**DPA1L**	**Data protection area 1 lower-limit address**	DPA1L
19			**DPA1U**	**Data protection area 1 upper-limit address**	DPA1U
20			**DPA2L**	**Data protection area 2 lower-limit address**	DPA2L
21			**DPA2U**	**Data protection area 2 upper-limit address**	DPA2U
22			**DPA3L**	**Data protection area 3 lower-limit address**	DPA3L
23			**DPA3U**	**Data protection area 3 upper-limit address**	DPA3U
24	**MCA**	**Memory protection setting check address**	**DPA4L**	**Data protection area 4 lower-limit address**	DPA4L
25	**MCS**	**Memory protection setting check size**	**DPA4U**	**Data protection area 4 upper-limit address**	DPA4U
26	**MCC**	**Memory protection setting check command**	**DPA5L**	**Data protection area 5 lower-limit address (2^n specification only)**	DPA5L
27	**MCR**	**Memory protection setting check result**	**DPA5U**	**Data protection area 5 upper-limit address (2^n specification only)**	DPA5U

10.1.1 MCA – Memory protection setting check address

This register specifies the base address of an area where memory protection setting is to be checked.

MCA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCA31	MCA30	MCA29	MCA28	MCA27	MCA26	MCA25	MCA24	MCA23	MCA22	MCA21	MCA20	MCA19	MCA18	MCA17	MCA16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MCA15	MCA14	MCA13	MCA12	MCA11	MCA10	MCA9	MCA8	MCA7	MCA6	MCA5	MCA4	MCA3	MCA2	MCA1	MCA0	Initial value Undefined

Bit Position	Bit Name	Description
31 to 0	MCA31 to MCA0	These bits specify in bytes the start address of the memory area where memory protection setting is to be checked.

10.1.2 MCS – Memory protection setting check size

This register specifies the size of an area where memory protection setting is to be checked.

MCS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCS31	MCS30	MCS29	MCS28	MCS27	MCS26	MCS25	MCS24	MCS23	MCS22	MCS21	MCS20	MCS19	MCS18	MCS17	MCS16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MCS15	MCS14	MCS13	MCS12	MCS11	MCS10	MCS9	MCS8	MCS7	MCS6	MCS5	MCS4	MCS3	MCS2	MCS1	MCS0	Initial value Undefined

Bit Position	Bit Name	Description
31 to 0	MCS31 to MCS0	These bits specify in bytes the size of the target area for which the size of the memory area where memory protection setting is to be checked is specified. Because the specified size is treated as an unsigned integer, the area cannot be checked in the direction in which the address value is decremented from the value of the MCA register. Do not set the MCS register to 00000000H.

10.1.3 MCC – Memory protection setting check command

This is a command register to start checking memory protection setting.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
MCC	MCC31	MCC30	MCC29	MCC28	MCC27	MCC26	MCC25	MCC24	MCC23	MCC22	MCC21	MCC20	MCC19	MCC18	MCC17	MCC16	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	MCC15	MCC14	MCC13	MCC12	MCC11	MCC10	MCC9	MCC8	MCC7	MCC6	MCC5	MCC4	MCC3	MCC2	MCC1	MCC0	Initial value 00000000H

Bit Position	Bit Name	Description
31 to 0	MCC31 to MCC0	When any value is written to the MCC register, checking memory protection settings is started. By setting the MCA/MCS register in advance and by writing this register, the result is stored in MCR. Because checking is started with an arbitrarily written value, checking can be started without additional registers, by using r0 as the source register. Checking reflects the result in accordance with the setting of each area, regardless of the status of the PSW.DMP and IMP bits. The value read from the MCC register is always 00000000H.

10.1.4 MCR – Memory protection setting check result

This register stores the result of checking memory protection setting.

Be sure to set bits 31 to 9, 7 to 5, and 3 to “0”.

	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	0	SE	0	WE	RE	XE	Initial value 00000000H

Bit Position	Bit Name	Description
8	OV	1 is stored in this bit when the specified area extends to 00000000H or 7FFFFFFFH. Otherwise, 0 is stored.
4	SE	1 is stored in this bit if the specified area fits in one of the protection areas and if the stack protection setting of the protection area is enabled. Otherwise, 0 is stored.
2	WE	1 is stored in this bit if the specified area fits in one of the protection areas and if that protection area is enabled to be written. Otherwise, 0 is stored.
1	RE	1 is stored in this bit if the specified area fits in one of the protection areas and if that protection area is enabled to be read. Otherwise, 0 is stored.
0	XE	1 is stored in this bit if the specified area fits in one of the protection areas and if that protection area is executable. Otherwise, 0 is stored.

10.2 Sample Code

An example of how to check the memory protection settings is shown below. If the area to be checked extends over 00000000H, it is judged that the specified area is incorrect, and the MCR.OV bit is set (1). To reference the check result, therefore, be sure to check the MCR.OV bit, confirm that the result is not illegal (OV = 0), and then use the other check results.

```
_service_protection:
   …
   ori     0x1000, r0, r12
   ldsr    r12, BSEL                  // Selecting MPV/PROT00 bank
   …
   mov     ADDRESS, r10               // Storing in r10 the start address of area to be checked
   mov     SIZE, r11                  // Storing in r11 the size of area to be checked
   di
   ldsr    r10, MCA // Setting address
   ldsr    r11, MCS // Setting size
   ldsr    r0, MCC  // Starting check
   stsr    MCR, r12 // Acquiring result
   ei
   andi    0x0100, r12, r0
   be      _overflow                  // Processed with input value assumed to be illegal if OV = 1
   br      _result_check              // Otherwise, judging the result.
```

CHAPTER 11 SPECIFAL FUNCTON

This chapter explains the special function related to the processor protection function.

11.1 Clearing Memory Protection Setting All at Once

By setting (1) the MPC.ALDS bit, the following memory protection setting bits are cleared (0) all at once in the cycle next to the one in which the MPC.ALDS bit is set.

- IPAnL.E bit (n = 0 to 4)
- DPAnL.E bit (n = 0 to 5)

PART 4 FLOATING-POINT OPERATION FUNCTION

CHAPTER 1 OVERVIEW

The floating-point unit (FPU) of the V850E2M CPU conforms to the V850E2v3 architecture and operates as a CPU coprocessor that executes floating-point operation 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 V850E2M CPU FPU conforms to ANSI/IEEE standard 754-1985 (IEEE binary floating-point operation standard).

1.1 Features

(1) Floating-point instructions

- Conform to ANSI/IEEE standard 754-1985 (IEEE binary floating-point operation standard).
- Supports single-precision (32-bit) and double-precision (64-bit).
- Supports the basic addition, subtraction, multiplication, and division instruction, multiply-add instruction, Maximum/Minimum instruction, and square root instruction.
- 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 transfer 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 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 register: Uses general-purpose registers
  (not special-purpose register for floating-point operations)
- Floating-point system register: 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

1.2 Implementing Floating-point Operation Function

The V850E2M CPU assumes the floating-point operation function as a coprocessor that can be implemented or not as follows. It depends on the products. Refer to Hardware User's Manual of each product.

(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 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 **CHAPTER 2 REGISTER SET**.

CHAPTER 2 REGISTER SET

2.1 Floating-point Operation 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:
  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. Since r0 is a zero register (always holds “0”), in principle {r1, r0} cannot be used by a double-precision floating-point instruction.

2.2 Floating-point System Registers

The FPU status bank (bank with group #20 and bank number 00H) of the system register bank includes 28 control registers. The FPU status bank is selected when the LDSR instruction sets 0x2000 to the BSEL register.

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, denormalized number flash 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 indicates the same information as the RM and XE bits of the FPSR register.
- FPCC: This register indicates the same information as the CC(7:0) bits of the FPSR register.
- FPCFG: This register indicates the same information as the RM and FS bits of the FPSR register.
- FPEC: Controls checking and canceling the pending status of the FPI exception.

System registers in the FPU bank other than the above are reserved for future expansion. These registers are write-prohibited. Also, their values are undefined when read. System registers can be accessed by the LDSR, STSR, or TRFSR instruction.

Table 2-1 shows the system register bank configuration. System register numbers 28 to 31 are used by all banks, and the CPU function bank’s EIWR, FEWR, DBWR, BSEL registers can be referenced regardless of the settings in the BSEL register.

Table 2-1. System Register Bank

System Register No.	System Register Name	Able to Specify Operands?		System Register Protection
		LDSR Instruction	STSR Instruction	
0 to 5	Reserved for future function expansion. (Operation is not guaranteed if these registers are accessed.)	×	×	√
6	FPSR – Floating-point configuration/status	√	√	√
7	FPEPC – Floating-point exception program counter	√	√	√
8	FPST – Floating point status	√	√	×
9	FPCC – Floating-point comparison result	√	√	×
10	FPCFG – Floating-point configuration	√	√	×
11	FPEC – Floating-point exception control	√	√	√
12 to 26	Reserved for future function expansion. (Operation is not guaranteed if these registers are accessed.)	×	×	√
28	EIWR – Working register for EI level exception	√	√	√
29	FEWR – Working register for FE level exception	√	√	√
30	DBWR – Working register for DB level exception	√	√	√
31	BSEL – Selection of register bank	√	√	√

Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: Indicates in the column of “Able to Specify Operands?” that the register cannot be specified. In the column of “System Register Protection”, this symbol indicates that the register is not protected.

2.2.1 FPSR – Floating-point configuration/status

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

For exception, see **CHAPTER 6 EXCEPTIONS** in **PART 2**.

Bits 23 and 22 are reserved for future function expansion, and writing of non-zero values is prohibited. Operation is undefined when a non-zero value is written. Also, values are undefined when read.

(1/2)

	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	0	0	DEM	SEM	RM		FS	PR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	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

Initial value **Note**

Bit position	Bit name	Description
31 to 24	CC(7:0)	These are the CC (condition) bits. They store the results of floating-point comparison instructions. The CC(7:0) bits are not affected by any instructions except the comparison instruction and LDSR instruction. The initial values are undefined. 0: Comparison result is false 1: Comparison result is true
21	DEM	This bit indicates double-precision operation exception mode. When the DEM bit is 1, any exception that occurs during execution of a double-precision (Double) instruction is handled as a precise exception. For description of double-precision instructions, see **4.2 Overview of Floating-point Instruction**. The initial value is 0.
20	SEM	This bit indicates single-precision operation exception mode. When the SEM bit = 1, any exception that occurs during execution of a single-precision (Single) instruction is handled as a precise exception. For description of single-precision instructions, see **4.2 Overview of Floating-point Instruction**. The initial value is 0.
19, 18	RM	These are the rounding mode control bits. The RM bits define the rounding mode that the FPU uses for all floating-point instructions. The initial value is 0. Be sure to set these bits to “00”.

RM bit		Mnemonic	Description
Bit 19	Bit 18		
0	0	RN	Rounds the result to the nearest representable value. If the value is exactly in-between the two nearest representable values, the result is rounded toward the value whose least significant bit is 0.
Other than above		Setting prohibited	

Note See the description of individual bits.

(2/2)

<table>
<tr><td>17</td><td>FS</td><td>The FS bit enables flushing of values that cannot be normalized (denormalized numbers). If this bit is set, the result of a denormalized number does not cause an unimplemented operation exception (E) but is flushed instead. Whether the denormalized number that has been flushed is 0 or the minimum normalized value depends on the rounding mode. The initial value is 1.
Be sure to set FS bit to “1”.
<table>
<tr><th>Result of denormalized number</th><th>Rounding mode of flushed result (RN)</th></tr>
<tr><td>Positive</td><td>+0</td></tr>
<tr><td>Negative</td><td>−0</td></tr>
</table></td></tr>
<tr><td>16</td><td>PR</td><td>If the exception mode of the instruction that has caused the floating-point operation exception is imprecise exception mode, this bit is cleared to 0. If it is precise exception mode, this bit is set to 1. The initial value is undefined.</td></tr>
<tr><td>15 to 10</td><td>XC
(E, V, Z, O, U, I)</td><td>These are the cause bits. The initial values are undefined. For details, see 2.2.1 (1) Cause bits (XC).</td></tr>
<tr><td>9 to 5</td><td>XE
(V, Z, O, U, I)</td><td>These are the enable bits. The initial values are 0. For details, see 2.2.1 (2) Enable bits (XE).</td></tr>
<tr><td>4 to 0</td><td>XP
(V, Z, O, U, I)</td><td>These are the preservation bits. The initial values are undefined. For details, see 2.2.1 (3) Preservation bits (XP).</td></tr>
</table>

(1) Cause bits (XC)

Bits 15 to 10 of 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 (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 (1).

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

The cause bits are rewritten by a floating-point operation instruction (except the TRFSR instruction) where the floating-point operation exception occurred. The E bit is set (1) when software emulation is required, otherwise it is cleared (0). Other bits are set (1) or cleared (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 (1) by an LDSR instruction, a floating-point operation exception does not occur.

(2) Enable bits (XE)

Bits 9 to 5 of 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 (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 (1), no exception occurs and the default result defined by IEEE754 is stored.

(3) Preservation bits (XP)

Bits 4 to 0 of the FPSR register are preservation bits. These bits store and indicate the detected exception after reset. An exception defined by IEEE754 occurs, and if the corresponding enable bit is not set (1), the preservation bit is set (1), otherwise it does not change. The preservation bits are not cleared (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.

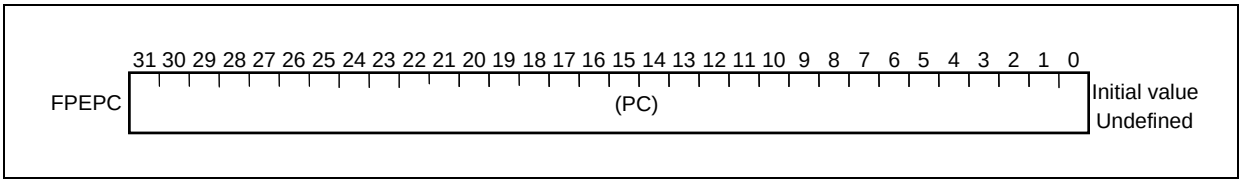
There are no preservation bits corresponding to unimplemented operation exceptions (E). An unimplemented operation exception (E) always occurs as a floating-point operation exception.

Remark For description of the exception types and how they relate to particular bits, see **Figure 5-1 Cause, Enable, and Preservation Bits of FPSR Register**.

2.2.2 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.

For bits 31 to 29 and 0, writing of non-zero values is prohibited. Operation is undefined when a non-zero value is written.



Caution	**For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign-extension of bit 28 is automatically set to bits 31 to 29 of FPEPC.**

2.2.3 FPST – Floating-point operation status

These bits indicate the same information as the PR, XC, and XP bits of the FPSR register.

For bits 31 to 16, 14, and 7 to 5, writing of non-zero values is prohibited. Operation is undefined when a non-zero value is written.

	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	
	PR	0	Cause bits (XC)						0	0	0	Preservation bits (XP)					Initial value
			E	V	Z	O	U	I				V	Z	O	U	I	Undefined

Bit position	Bit name	Description
15	PR	This bit indicates the exception mode of the instruction where the floating-point operation exception occurred. The initial values are undefined. The value written to this bit is reflected in PR bit of FPSR. 0: Imprecise exception 1: Precise exception
13 to 8	XC (E, V, Z, O, U, I)	These are cause bits. The initial values are undefined. For details, see **2.2.1 (1) Cause bits (XC)**. Values written to these bits are reflected in XC bits of FPSR.
4 to 0	XP (V, Z, O, U, I)	These are preservation bits. The initial values are undefined. For details, see **2.2.1 (3) Preservation bits (XP)**. Values written to these bits are reflected in XP bits of FPSR.

2.2.4 FPCC - Floating-point operation comparison result

These bits indicate the same information as the CC(7:0) bits of the FPSR register.

For bits 31 to 8, writing of non-zero values is prohibited. Operation is undefined when a non-zero value is written.

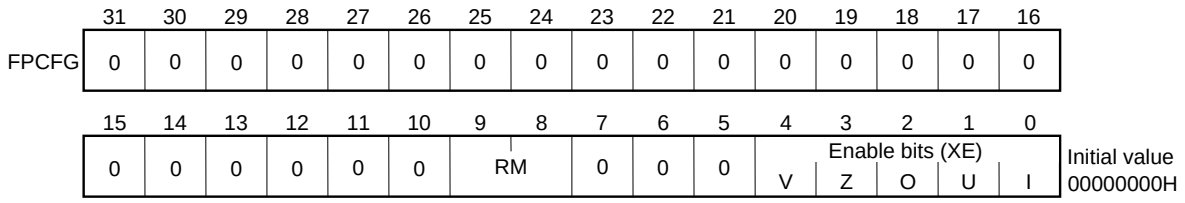
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
FPCC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	Initial value Undefined

Bit position	Bit name	Description
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. The initial values are undefined. Values written to these bits are reflected in CC(7:0) bits of FPSR. 0: Comparison result is false 1: Comparison result is true

2.2.5 FPCFG – Floating-point operation configuration

These bits indicate the same information as the RM and XE bits of the FPSR register.

For bits 31 to 10 and 7 to 5, writing of non-zero values is prohibited. Operation is undefined when a non-zero value is written.



Bit position	Bit name	Description
9, 8	RM	These are rounding mode control bits. The RM bits define the rounding mode that the FPU uses for all floating-point instructions. The initial value is 0. Be sure to set these bits to “00”. Values set to these bits are reflected in RM bits of FPSR.
4 to 0	XE (V, Z, O, U, I)	These are enable bits. The initial value is 0. For details, see **2.2.1 (2) Enable bits (XE)**. Also, values written to these bits are reflected in XE bits of FPSR.

RM bit		Mnemonic	Description
Bit 9	Bit 8		
0	0	RN	Rounds the result to the nearest representable value. If the value is exactly in-between the two representable values, the result is rounded toward the value whose least significant bit is 0.
Other than above		Setting prohibited	

2.2.6 FPEC – Floating-point exception control

This register controls the floating-point operation exception.

Writing a value other than 0 to bits 31 to 1 is prohibited. If a value other than 0 is written to these bits, the operation is undefined.

Caution For how to handle the FPEC register, refer to 6.3 Exception Management in PART 2.

	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	FPIVD	Initial value 00000000H

Bit Position	Bit Name	Meaning
0	FPIVD[Note]	This bit indicates the status of reporting the FPI exception. If this bit is set (1), the FPI exception is reported to the CPU but is not acknowledged. It is automatically cleared (0) when the CPU acknowledges the FPI exception. While this bit is set (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 (1). When report of the LDSR instruction is canceled, the CPU does not acknowledge the FPI exception. 0: FPI exception is reported. 1: FPI exception is reported.

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

CHAPTER 3 DATA TYPES

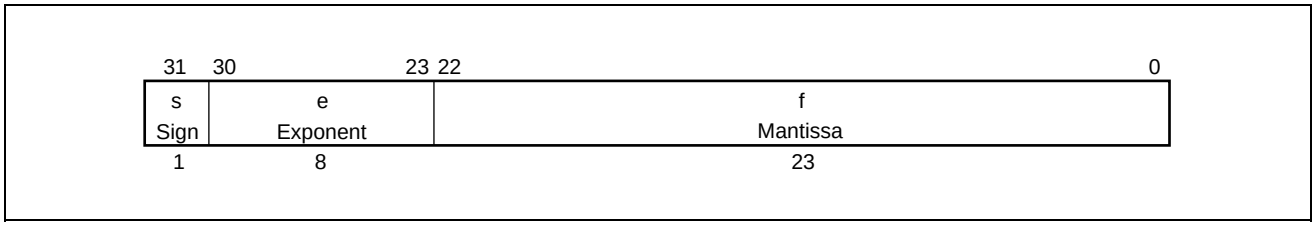
3.1 Data Formats

3.1.1 Floating-point format

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

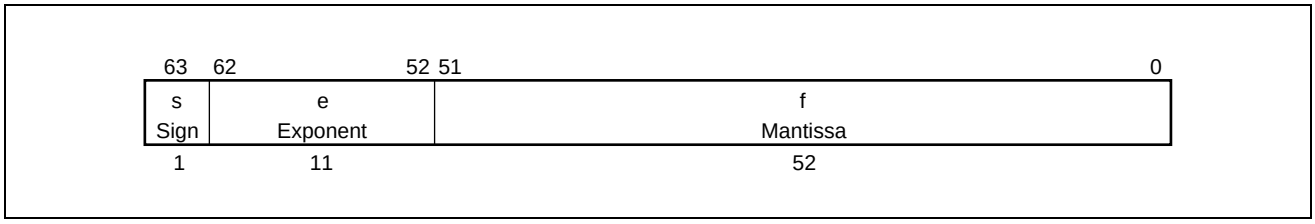
The single-precision floating-point format consists of a 24-bit signed mantissa (s + f) and an 8-bit exponent (e), as shown in Figure 3-1.

Figure 3-1. Single-precision Floating-point Format



The double-precision floating-point format consists of a 53-bit signed mantissa (s + f) and an 11-bit exponent (e), as shown in Figure 3-2.

Figure 3-2. 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
- Mantissa: 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 denormalized 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 3-1.

Table 3-1. 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 $v = (-1)^s 2^E (1.f)$
Denormalized number	If $E = E_{min} - 1$ and $f \neq 0$	then $v = (-1)^s 2^{Emin} (0.f)$
±0 (zero)	If $E = E_{min} - 1$ and $f = 0$	then $v = (-1)^s 0$

- 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 may 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 3-2 lists the value of each parameter defined in floating-point formats.

Table 3-2. Floating-point Formats and Parameter Values

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

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

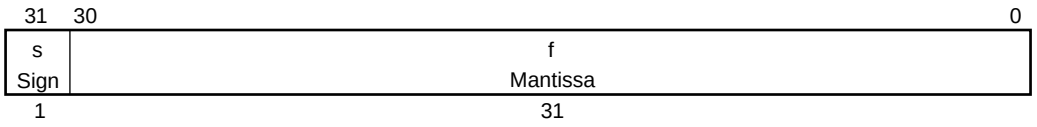
Table 3-3. Floating-point Minimum and Maximum Values

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

3.1.2 Fixed-point formats

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

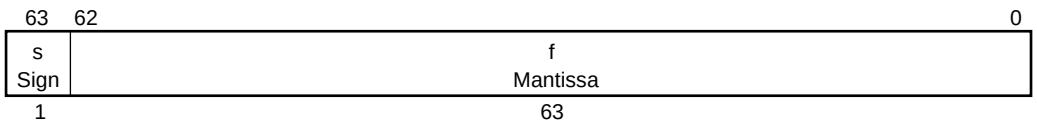
Figure 3-3. 32-bit Fixed-Point Format



s: Sign bit

i: Integer value (2's complement)

Figure 3-4. 64-bit Fixed-Point Format



s: Sign bit

i: Integer value (2's complement)

CHAPTER 4 INSTRUCTIONS

4.1 Instruction Formats

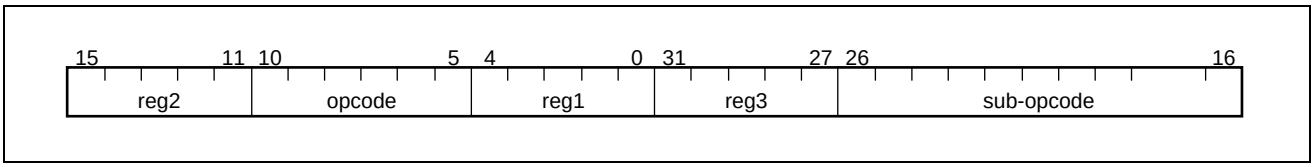
All floating-point instructions are in 32-bit format.

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) → Higher 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.



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

- MADDF.S: Floating-point Multiply-Add (Single)
- MSUBF.S: Floating-point Multiply-Subtract (Single)
- NMADDF.S: Floating-point Negate Multiply-Add (Single)
- NMSUBF.S: Floating-point Negate Multiply-Subtract (Single)

(3) Conversion instructions

- CEILF.DL: Floating-point Truncate to Long Fixed-point Format, rounded toward +∞ (Double)
- CEILF.DW: Floating-point Truncate to Single Fixed-point Format, rounded toward +∞ (Double)
- CEILF.SL: Floating-point Truncate to Long Fixed-point Format, rounded toward +∞ (Single)
- CEILF.SW: Floating-point Truncate to Single Fixed-point Format, rounded toward +∞ (Single)
- CEILF.DUL: Floating-point Truncate to Unsigned Long, rounded toward +∞ (Double)
- CEILF.DUW: Floating-point Truncate to Unsigned Word, rounded toward +∞ (Double)
- CEILF.SUL: Floating-point Truncate to Unsigned Long, rounded toward +∞ (Single)
- CEILF.SUW: Floating-point Truncate to Unsigned Word, rounded toward +∞ (Single)
- CVTF.DL: Floating-point Convert to Long Fixed-point Format (Double)
- CVTF.DS: Floating-point Convert to Single Floating-point Format (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 to Single Fixed-point Format (Double)
- CVTF.LD: Floating-point Convert to Single Floating-point Format (Double)
- CVTF.LS: Floating-point Convert to Single Floating-point Format (Single)
- CVTF.SD: Floating-point Convert to Double Floating-point Format (Double)
- CVTF.SL: Floating-point Convert to Long Fixed-point Format (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 to Single Fixed-point Format (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 to Single Floating-point Format (Double)
- CVTF.WS: Floating-point Convert to Single Floating-point Format (Single)
- FLOORF.DL: Floating-point Truncate to Long Fixed-point Format, rounded toward -∞ (Double)
- FLOORF.DW: Floating-point Truncate to Single Fixed-point Format, rounded toward -∞ (Double)
- FLOORF.SL: Floating-point Truncate to Long Fixed-point Format, rounded toward -∞ (Single)
- FLOORF.SW: Floating-point Truncate to Single Fixed-point Format, rounded toward -∞ (Single)
- FLOORF.DUL: Floating-point Truncate to Unsigned Long, rounded toward -∞ (Double)
- FLOORF.DUW: Floating-point Truncate to Unsigned Word, rounded toward -∞ (Double)
- FLOORF.SUL: Floating-point Truncate to Unsigned Long, rounded toward -∞ (Single)
- FLOORF.SUW: Floating-point Truncate to Unsigned Word, rounded toward -∞ (Single)
- TRNCF.DL: Floating-point Truncate to Long Fixed-point Format, rounded to zero (Double)
- TRNCF.DUL: Floating-point Truncate Double to Unsigned-Long (Double)
- TRNCF.DUW: Floating-point Truncate Double to Unsigned-Word (Double)
- TRNCF.DW: Floating-point Truncate to Single Fixed-point Format, rounded to zero (Double)
- TRNCF.SL: Floating-point Truncate to Long Fixed-point Format, rounded to zero (Single)

- TRNCF.SUL: Floating-point Truncate Single to Unsigned-Long (Single)
- TRNCF.SUW: Floating-point Truncate Single to Unsigned-Word (Single)
- TRNCF.SW: Floating-point Truncate to Single Fixed-point Format, rounded to zero (Single)

(4) Comparison instructions

- CMPF.S: Compares floating-point values (Single)
- CMPF.D: Compares floating-point values (Double)

(5) Conditional transfer 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)

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 4-1 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 4-2 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 4-1. 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 4-2. Definitions of Condition Code Bits and Their Logical Inversions

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

4.4 Instruction Set

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

- Instruction format: Describes the instruction coding method and operands used (symbols are listed in Table 4-3).
- Operation: Indicates instruction function (symbols are listed in Table 4-4).
- Format: Indicates the instruction format (see **4.1 Instruction Formats**).
- opcode: Indicates the instruction opcode in bit fields (symbols are listed in Table 4-5).
- Description: Describes the instruction operations.
- Supplement: Provides a supplementary description of the instruction.

Table 4-3. 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, refer to **4.3 Conditions for Comparison Instructions**).

Table 4-4. Operations

Symbol	Explanation
←	Assignment (input for)
GR [ ]	General-purpose register
SR [ ]	System register
result	Result is reflected in flag.
+	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

Table 4-5. 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>

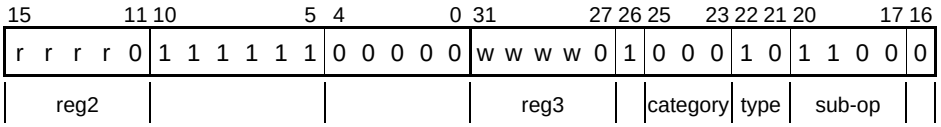
ABSF.D	Floating-point Absolute Value (Double) Floating-point absolute value (double precision)

[Instruction format] ABSF.D reg2, reg3

[Operation] reg3 ← abs (reg2)

[Format] Format F:I

[opcode]



[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.

The absolute value operation is performed arithmetically. In other words, when the operand is S-NaN, an IEEE754-defined invalid operation exception (V) is detected.

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

Inexact exception (I)

[Operation result] When FPSR.FS = 1

reg2	+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
Operation result [exception]	+Normal		+0		+∞		Q-NaN	Q-NaN [V]

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

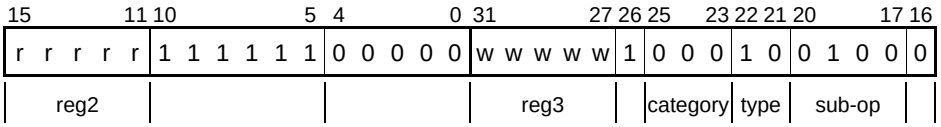
ABSF.S	Floating-point Absolute Value (Single) Floating-point absolute value (single precision)

[Instruction format] ABSF.S reg2, reg3

[Operation] reg3 ← abs (reg2)

[Format] Format F:I

[opcode]



[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. The absolute value operation is performed arithmetically. In other words, when the operand is S-NaN, an IEEE754-defined invalid operation exception (V) is detected.

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

[Operation result] When FPSR.FS = 1

reg2	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
Operation result [exception]	+Normal		+0		+∞		Q-NaN	Q-NaN [V]

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

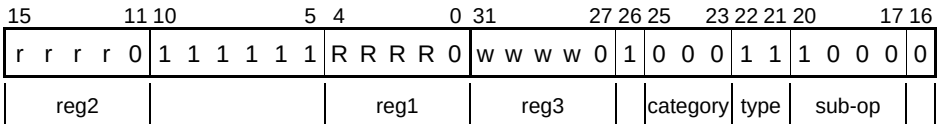
ADDF.D	Floating-point Add (Double) Floating-point add (double precision)

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

[Operation] reg3 ← reg2 + reg1

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg1 (A) \ reg2 (B)	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
+Normal	A+B					–∞		
–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. Denormalized numbers are flushed and handled as “0”.

<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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg1 (A) \ reg2 (B)	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
+Normal	A+B				+∞	–∞	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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

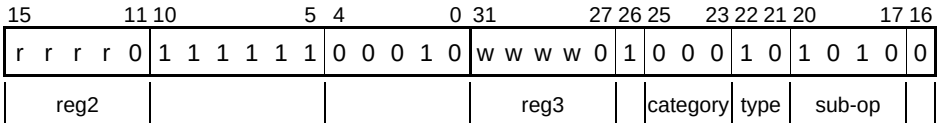
CEILF.DL	Floating-point Ceiling to Long Fixed-point Format (Double) Conversion to fixed-point format (double precision)

[Instruction format] CEILF.DL reg2, reg3

[Operation] reg3 ← ceil reg2 (double → long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

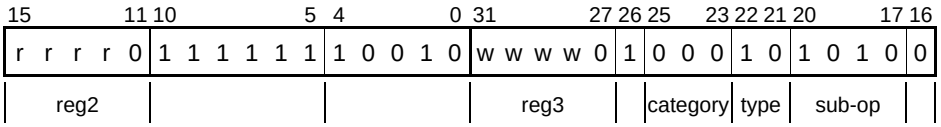
CEILF.DUL	Floating-point Ceiling to Unsigned Long Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CEILF.DUL reg2, reg3

[Operation] reg3 ← ceil reg2 (double → unsigned long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

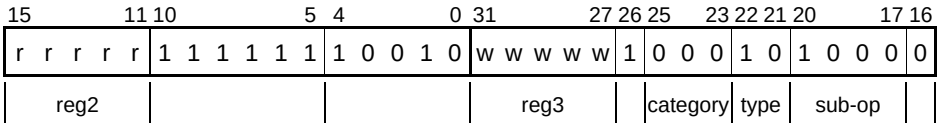
CEILF.DUW	Floating-point Ceiling to Unsigned Single Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CEILF.DUW reg2, reg3

[Operation] reg3 ← ceil reg2 (double → unsigned word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

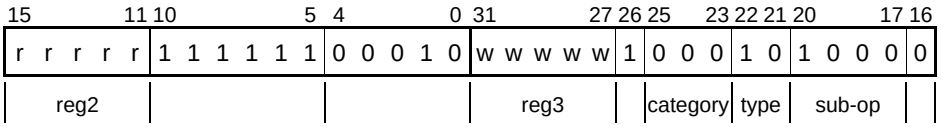
CEILF.DW	Floating-point Ceiling to Single Fixed-point Format (Double) Conversion to fixed-point format (double precision)

[Instruction format] CEILF.DW reg2, reg3

[Operation] reg3 ← ceil reg2 (double → word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

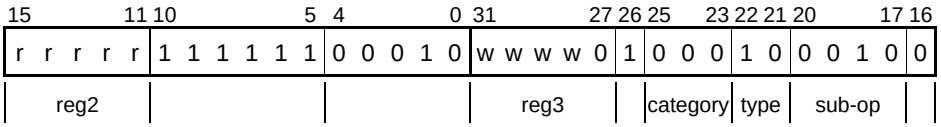
CEILF.SL	Floating-point Ceiling to Long Fixed-point Format (Single) Conversion to fixed-point format (single precision)

[Instruction format] CEILF.SL reg2, reg3

[Operation] reg3 ← ceil reg2 (single → long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

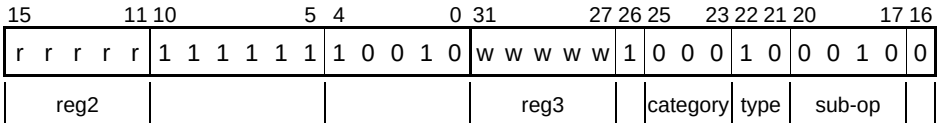
CEILF.SUL	Floating-point Ceiling to Unsigned Long Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CEILF.SUL reg2, reg3

[Operation] reg3 ← ceil reg2 (single → unsigned long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

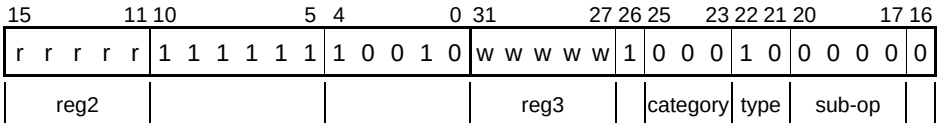
CEILF.SUW	Floating-point Ceiling to Unsigned Single Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CEILF.SUW reg2, reg3

[Operation] reg3 ← ceil reg2 (single → unsigned word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

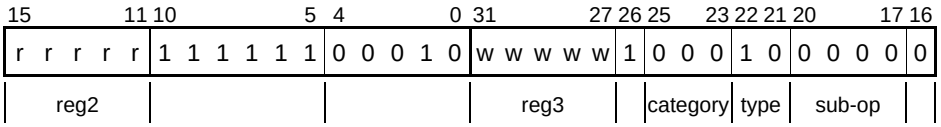
CEILF.SW	Floating-point Ceiling to Single Fixed-point Format (Single) Conversion to fixed-point format (single precision)

[Instruction format] CEILF.SW reg2, reg3

[Operation] reg3 ← ceil reg2 (single → word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CMOVF.D	Floating-point Conditional Move (Double) Conditional transfer (double precision)

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

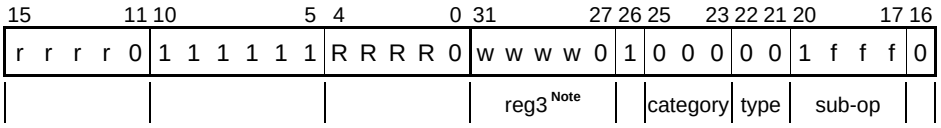
[Operation]

```
if FPSR.CCn == 1 then
    reg3 ← reg1
else
    reg3 ← reg2
endif
```

Remark n = fcbit

[Format] Format F:I

[opcode]



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

Caution Do not set reg3 to r0.

<Floating-point condition instruction >

CMOVF.S	Floating-point Conditional Move (Single) Conditional transfer (single precision)

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

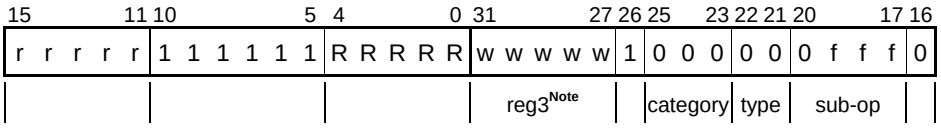
[Operation]

```
if FPSR.CCn == 1 then
    reg3 ← reg1
else
    reg3 ← reg2
endif
```

Remark n = fcbit

[Format] Format F:I

[opcode]



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

Caution Do not set reg3 to r0.

<Floating-point instruction>

CMPF.D	Floating-point Compare (Double) Floating-point comparison (double precision)

[Instruction format] CMPF.D fcond, reg2, reg1, fcbit

CMPF.D fcond, reg2, reg1

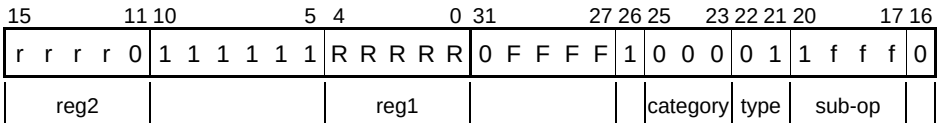
[Operation]

```
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]



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 4-6 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 operation 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 processing 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 4-6. Comparison Conditions

Comparison Conditions		Definition	Description	Invalid operation exception occurs when 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.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)

[Operation result] When FPSR.FS = 1

[Condition code (fcond) = 0 to 7]

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

[Condition code (fcond) = 8 to 15]

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

<Floating-point instruction>

CMPF.S	Floating-point Compare (Single) Floating-point comparison (single precision)

[Instruction format] CMPF.S fcond, reg2, reg1, fcbit

CMPF.S fcond, reg2, reg1

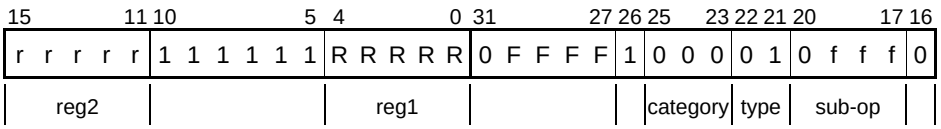
[Operation]

```
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]



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 4-7 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 operation 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 processing 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 4-7. Comparison Conditions

Comparison Conditions		Definition	Description	Invalid operation exception occurs when 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)

[Operation result] When FPSR.FS = 1

[Condition code (fcond) = 0 to 7]

reg1 (B) / reg2 (A)	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
±Normal	Stores result of comparison (true or false) under comparison condition (fcond) in FPSR.CCn bit (n = fcbit)							
±0								
±∞								
Q-NaN	Unordered							
S-NaN								

[Condition code (fcond) = 8 to 15]

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

<Floating-point instruction>

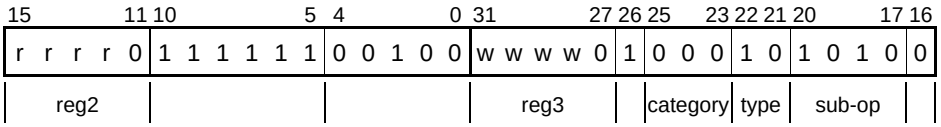
CVTF.DL	Floating-point Convert to Long Fixed-point Format (Double) Conversion to fixed-point format (double precision)

[Instruction format] CVTF.DL reg2, reg3

[Operation] reg3 ← cvt reg2 (double → long-word)

[Format] Format F:I

[opcode]



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

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to -2^{63}, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.DS	Floating-point Convert to Single Floating-point Format (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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

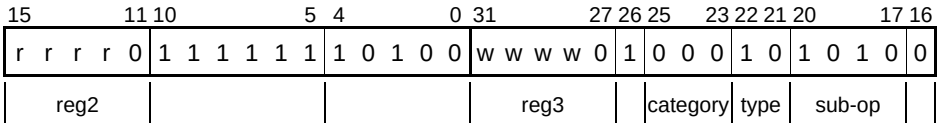
CVTF.DUL	Floating-point Convert to Unsigned Long Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.DUL reg2, reg3

[Operation] reg3 ← cvt reg2 (double → unsigned long-word)

[Format] Format F:I

[opcode]



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

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{64} - 1$ to 0, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

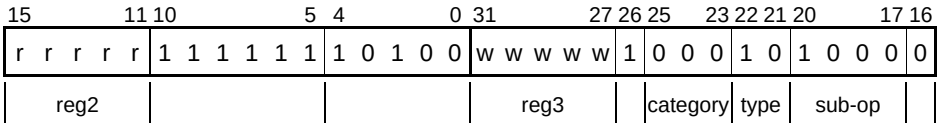
CVTF.DUW	Floating-point Convert to Unsigned Single Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.DUW reg2, reg3

[Operation] reg3 ← cvt reg2 (double → word)

[Format] Format F:I

[opcode]



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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.DW	Floating-point Convert to Single Fixed-point Format (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 r	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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.LD	Floating-point Convert to Double Floating-point Format (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	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

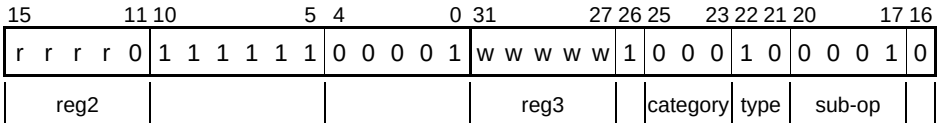
CVTF.LS	Floating-point Convert to Single Floating-point Format (Single) Conversion to floating-point format (single precision)

[Instruction format] CVTF.LS reg2, reg3

[Operation] reg3 ← cvt reg2 (long-word → single)

[Format] Format F:I

[opcode]



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.SD	Floating-point Convert to Double Floating-point Format (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	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

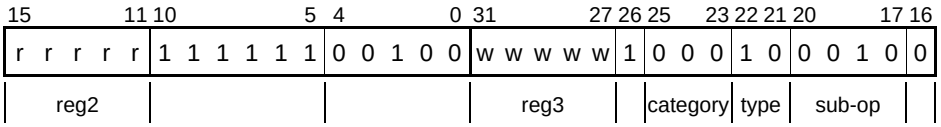
CVTF.SL	Floating-point Convert to Long Fixed-point Format(Single) Conversion to fixed-point format (single precision)

[Instruction format] CVTF.SL reg2, reg3

[Operation] reg3 ← cvt reg2 (single → long-word)

[Format] Format F:I

[opcode]



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

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to -2^{63}, an IEEE754-defined invalid operation exception is detected.

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

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

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

Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

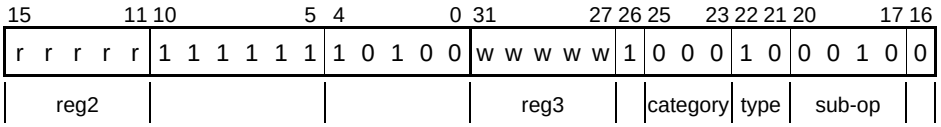
CVTF.SUL	Floating-point Convert to Unsigned Long Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.SUL reg2, reg3

[Operation] reg3 ← cvt reg2 (single → unsigned long-word)

[Format] Format F:I

[opcode]



[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 $+\infty$: $2^{64} - 1$ is returned.
- Source is a negative number, not-a-number, or $-\infty$: 0 is returned.

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

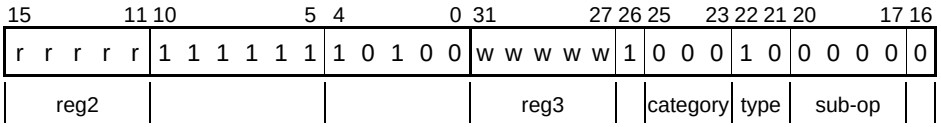
CVTF.SUW	Floating-point Convert to Unsigned Single Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.SUW reg2, reg3

[Operation] reg3 ← cvt reg2 (single → unsigned word)

[Format] Format F:I

[opcode]



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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.SW	Floating-point Convert to Single Fixed-point Format (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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

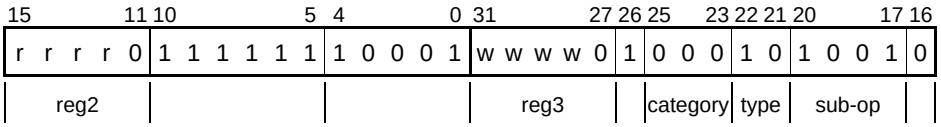
CVTF.ULD	Floating-point Convert to Double Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.ULD reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned long-word → double)

[Format] Format F:I

[opcode]



[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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

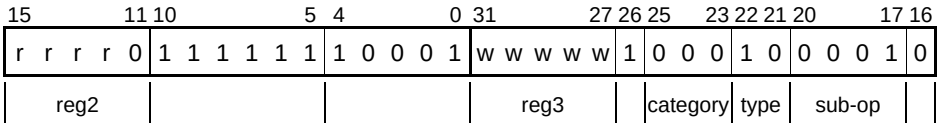
CVTF.ULS	Floating-point Convert to Single Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.ULS reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned long-word → single)

[Format] Format F:I

[opcode]



[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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

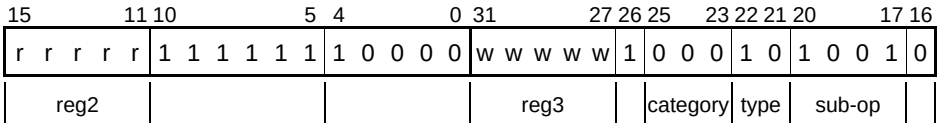
CVTF.UWD	Floating-point Convert to Double Fixed-point Format (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] CVTF.UWD reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned word → double)

[Format] Format F:I

[opcode]



[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] Inexact exception (I)

[Operation result] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

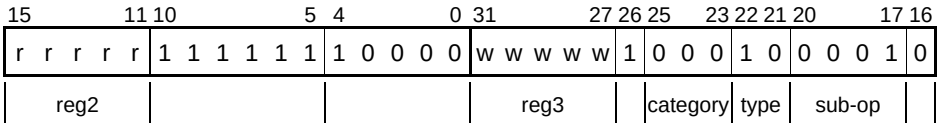
CVTF.UWS	Floating-point Convert to Single Fixed-point Format (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] CVTF.UWS reg2, reg3

[Operation] reg3 ← cvt reg2 (unsigned word → single)

[Format] Format F:I

[opcode]



[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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.WD	Floating-point Convert to Double Floating-point Format (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	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	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	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] Inexact exception (I)

[Operation result] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

CVTF.WS	Floating-point Convert to Single Floating-point Format (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] When FPSR.FS = 1

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

Remark Denormalized numbers are flushed and handled as “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] Invalid operation exception (V)
Inexact exception (I)
Division-by-zero exception (Z)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

<table>
<tr><th>reg1 (A) \ reg2 (B)</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="4" rowspan="2">B÷A</td><td>+∞</td><td>–∞</td><td rowspan="6" colspan="2"></td></tr>
<tr><td>–Normal</td><td>–∞</td><td>+∞</td></tr>
<tr><td>+0</td><td colspan="2" rowspan="2">±∞ [Z]</td><td colspan="2" rowspan="2">Q-NaN [V]</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–0</td><td>–∞</td><td>+∞</td></tr>
<tr><td>+∞</td><td>+0</td><td>–0</td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Q-NaN [V]</td></tr>
<tr><td>–∞</td><td>–0</td><td>+0</td><td>–0</td><td>+0</td></tr>
<tr><td>Q-NaN</td><td colspan="6"></td><td>Q-NaN</td><td rowspan="2"></td></tr>
<tr><td>S-NaN</td><td colspan="7"></td><td>Q-NaN [V]</td></tr>
</table>

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

2. Denormalized numbers are flushed and handled as “0”.

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

[Operation result] When FPSR.FS = 1

<table>
<tr><th>reg1 (A) \ reg2 (B)</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="4" rowspan="2">B÷A</td><td>+∞</td><td>–∞</td><td rowspan="6"></td><td rowspan="6"></td></tr>
<tr><td>–Normal</td><td>–∞</td><td>+∞</td></tr>
<tr><td>+0</td><td colspan="2" rowspan="2">±∞ [Z]</td><td colspan="2" rowspan="2">Q-NaN [V]</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–0</td><td>–∞</td><td>+∞</td></tr>
<tr><td>+∞</td><td>+0</td><td>–0</td><td>+0</td><td>–0</td><td colspan="2" rowspan="2">Q-NaN [V]</td></tr>
<tr><td>–∞</td><td>–0</td><td>+0</td><td>–0</td><td>+0</td></tr>
<tr><td>Q-NaN</td><td colspan="7"></td><td>Q-NaN</td><td></td></tr>
<tr><td>S-NaN</td><td colspan="8"></td><td>Q-NaN [V]</td></tr>
</table>

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

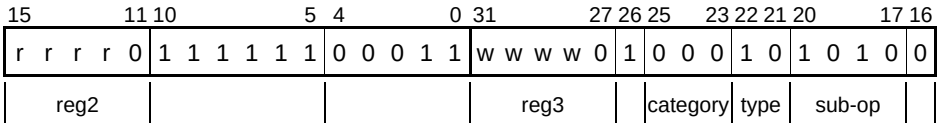
FLOORF.DL	Floating-point Truncate, rounded toward the direction of –∞ (Double) Conversion to fixed-point format (double precision)

[Instruction format] FLOORF.DL reg2, reg3

[Operation] reg3 ← floor reg2 (double → long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

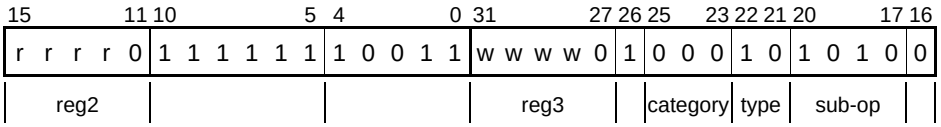
FLOORF.DUL	Floating-point Truncate to Unsigned, rounded toward the direction of –∞ (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] FLOORF.DUL reg2, reg3

[Operation] reg3 ← floor reg2 (double → unsigned long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

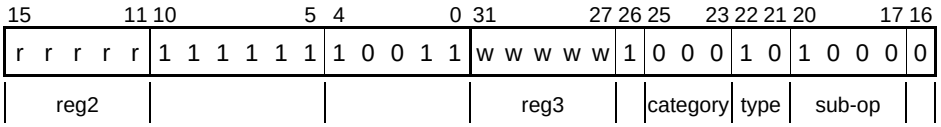
FLOORF.DUW	Floating-point Truncate to Unsigned, rounded toward the direction of –∞ (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] FLOORF.DUW reg2, reg3

[Operation] reg3 ← floor reg2 (double → unsigned word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

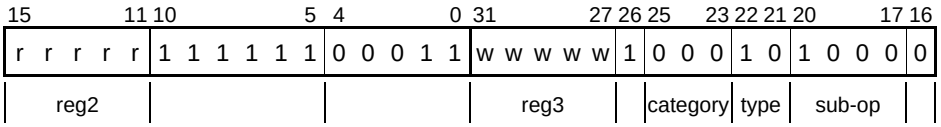
FLOORF.DW	Floating-point Truncate, rounded toward the direction of –∞ (Double) Conversion to fixed-point format (double precision)

[Instruction format] FLOORF.DW reg2, reg3

[Operation] reg3 ← floor reg2 (double → word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

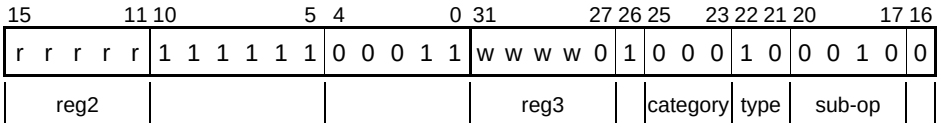
FLOORF.SL	Floating-point Truncate, rounded toward the direction of –∞ (Single) Conversion to fixed-point format (single precision)

[Instruction format] FLOORF.SL reg2, reg3

[Operation] reg3 ← floor reg2 (single → long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

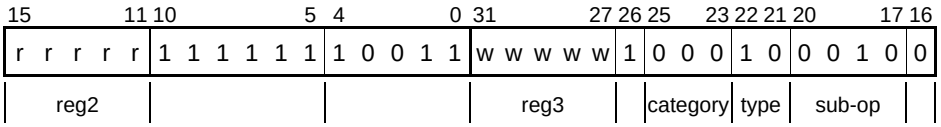
FLOORF.SUL	Floating-point Truncate to Unsigned, rounded toward the direction of –∞ (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] FLOORF.SUL reg2, reg3

[Operation] reg3 ← floor reg2 (single → unsigned long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

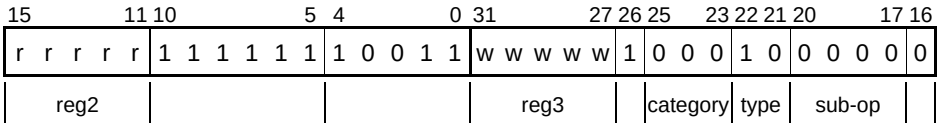
FLOORF.SUW	Floating-point Truncate to Unsigned, rounded toward the direction of -∞ (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] FLOORF.SUW reg2, reg3

[Operation] reg3 ← floor reg2 (single → unsigned word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

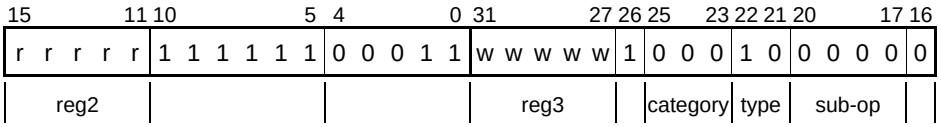
FLOORF.SW	Floating-point Truncate, rounded toward the direction of -∞ (Single) Conversion to fixed-point format (single precision)

[Instruction format] FLOORF.SW reg2, reg3

[Operation] reg3 ← floor reg2 (single → word)

[Format] Format F:I

[opcode]



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

The result is rounded in the $-\infty$ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{31} - 1$ to -2^{31}, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

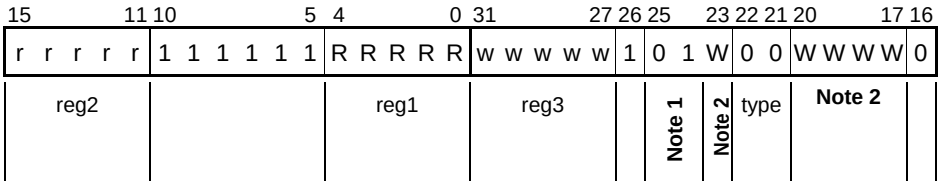
MADDF.S	Floating-point Multiply-add (Single) Floating-point fused-multiply-add operation (single precision)

[Instruction format] MADDF.S reg1, reg2, reg3, reg4

[Operation] reg4 ← reg2 × reg1 + reg3

[Format] Format F:I

[opcode]



Notes 1. category

2. reg4 (The least significant bit of reg4 is bit 23.)

[Description] This instruction multiples the contents of general-purpose register reg2 by the contents of general-purpose register reg1, then adds the result with the contents of floating-point register reg3 and stores the result in general-purpose register reg4. 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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg3(C) \ reg2 (B) / reg1 (A)		+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
±Normal	+Normal	MADD(A, B, C)				+∞	–∞		
	–Normal					–∞	+∞		
	±0					Q-NaN [V]			
	+∞	+∞	–∞	Q-NaN [V]		+∞	–∞		
	–∞	–∞	+∞			–∞	+∞		
±0	+Normal	MADD(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. Denormalized numbers are flushed and handled as "0".

[Supplement] An exception occurs based on the multiplication result in the following case.

- When the multiplication operation results in an overflow and the addition operation does not cause an invalid operation exception.

<Floating-point instruction>

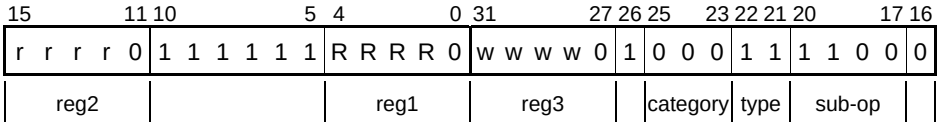
MAXF.D	Floating-point Maximum (Double) Floating-point maximum value (double precision)

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

[Operation] reg3 ← max (reg2, reg1)

[Format] Format F:I

[opcode]



[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.
When the FS bit of the FPSR register has been set, both reg1 and reg2 contain denormalized numbers, and the maximum value is either +0 or −0, +0 or −0 is stored in reg3.

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

[Supplement] When the FS bit of the FPSR register has been set, both reg1 and reg2 contain denormalized numbers, and the maximum value is either +0 or −0, it is undefined whether+0 or −0 is stored in reg3.

[Operation result] When FPSR.FS = 1

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

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

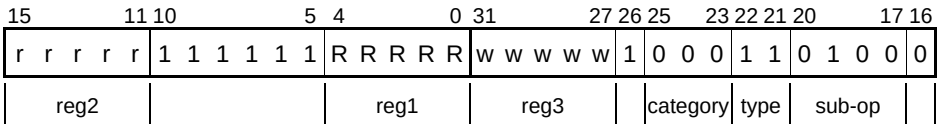
MAXF.S	Floating-point Maximum (Single) Floating-point maximum value (single precision)

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

[Operation] reg3 ← max (reg2, reg1)

[Format] Format F:I

[opcode]



[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.

When the FS bit of the FPSR register has been set, and both reg1 and reg2 contain denormalized numbers, the maximum value is either +0 or −0, +0 or −0 is stored in reg3.

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

[Supplement] When the FS bit of the FPSR register has been set, and both reg1 and reg2 contain denormalized numbers, the maximum value is either +0 or −0, it is undefined whether +0 or −0 is stored in reg3.

[Operation result] When FPSR.FS = 1

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

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

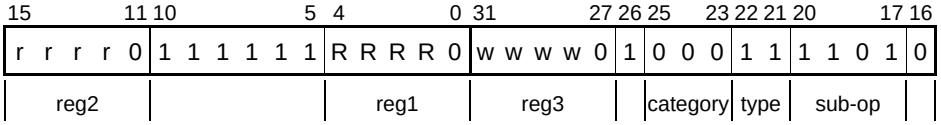
MINF.D	Floating-point Minimum (Double) Floating-point minimum value (double precision)

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

[Operation] reg3 ← min (reg2, reg1)

[Format] Format F:I

[opcode]



[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.
When the FS bit of the FPSR register has been set, and both reg1 and reg2 contain denormalized numbers, and the minimum value is either +0 or –0, +0 or –0 is stored in reg3.

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

[Supplement] When the FS bit of the FPSR register has been set, and both reg1 and reg2 contain denormalized numbers, and the minimum value is either +0 or –0, whether +0 or –0 is stored in reg3 is undefined.

[Operation result] When FPSR.FS = 1

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

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

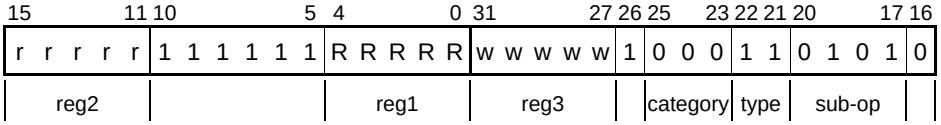
MINF.S	Floating-point Minimum (Single) Floating-point minimum value (single precision)

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

[Operation] reg3 ← min (reg2, reg1)

[Format] Format F:I

[opcode]



[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.

When the FS bit of the FPSR register has been set, both reg1 and reg2 contain denormalized numbers, and the minimum value is either +0 or −0, +0 or −0 is stored in reg3.

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

[Supplement] When the FS bit of the FPSR register has been set, both reg1 and reg2 contain denormalized numbers, and the minimum value is either +0 or −0, whether +0 or −0 is stored in reg3 is undefined.

[Operation result] When FPSR.FS = 1

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

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

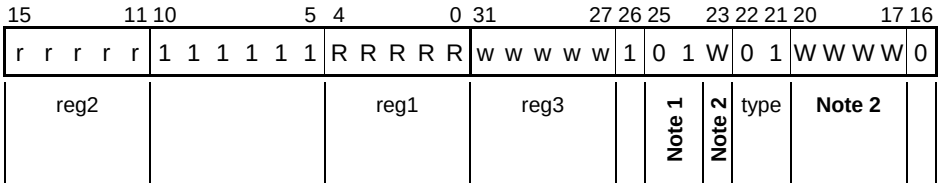
MSUBF.S	Floating-point Multiply-subtract (Single) Floating-point fused-multiply-add operation (single precision)

[Instruction format] MSUBF.S reg1, reg2, reg3, reg4

[Operation] reg4 ← reg2 × reg1 − reg3

[Format] Format F:I

[opcode]



Notes 1. category

2. reg4 (The least significant bit of reg4 is bit 23.)

[Description] This instruction multiples 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, then subtracts the single-precision floating-point format contents of general-purpose register reg3, and stores the result in general-purpose register reg4. 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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg3(C) \ reg1 (A) \ reg2 (B)		+Normal	−Normal	+0	−0	+∞	−∞	Q-NaN	S-NaN
±Normal	+Normal	MUSB(A, B, C)				+∞	−∞		
	−Normal					−∞	+∞		
	±0					Q-NaN [V]			
	+∞	+∞	−∞	Q-NaN [V]		+∞	−∞		
	−∞	−∞	+∞			−∞	+∞		
±0	+Normal	MUSB(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. Denormalized numbers are flushed and handled as "0".

[Supplement] An exception occurs based on the multiplication result in the following case.

- When the multiplication operation results in an overflow and the addition operation does not cause an invalid operation exception.

<Floating-point instruction>

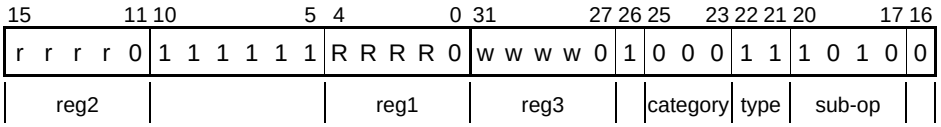
MULF.D	Floating-point Multiply (Double) Floating-point multiplication (double precision)

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

[Operation] reg3 ← reg2 × reg1

[Format] Format F:I

[opcode]



[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 general-purpose register reg3.

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

[Operation result] When FPSR.FS = 1

reg1 (A) \ reg2 (B)	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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

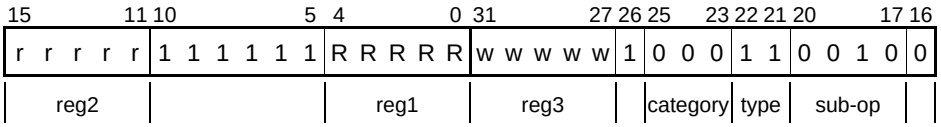
MULF.S	Floating-point Multiply (Single) Floating-point multiplication (single precision)

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

[Operation] reg3 ← reg2 × reg1

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

<table>
<tr><th>reg1 (A) \ reg2 (B)</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="4" rowspan="4">A×B</td><td>+∞</td><td>–∞</td><td rowspan="6">Q-NaN</td><td rowspan="7">Q-NaN [V]</td></tr>
<tr><td>–Normal</td><td>–∞</td><td>+∞</td></tr>
<tr><td>+0</td><td colspan="2" rowspan="2">Q-NaN [V]</td></tr>
<tr><td>–0</td></tr>
<tr><td>+∞</td><td>+∞</td><td>–∞</td><td colspan="2" rowspan="2">Q-NaN [V]</td><td>+∞</td><td>–∞</td></tr>
<tr><td>–∞</td><td>–∞</td><td>+∞</td><td>–∞</td><td>+∞</td></tr>
<tr><td>Q-NaN</td><td colspan="6">Q-NaN</td></tr>
<tr><td>S-NaN</td><td colspan="7">Q-NaN [V]</td></tr>
</table>

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

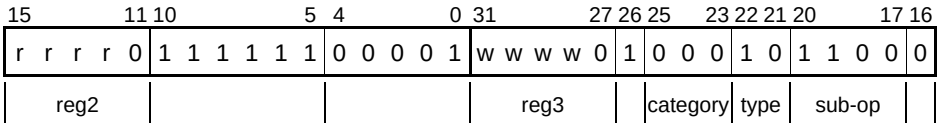
NEGF.D	Floating-point Negate (Double) Floating-point sign inversion (double precision)

[Instruction format] NEGF.D reg2, reg3

[Operation] reg3 ← neg reg2

[Format] Format F:I

[opcode]



[Description] This instruction inverts the sign of double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in general-purpose register reg3.

Sign inversion is performed arithmetically. This means that when the operand is S-NaN, an IEEE754-defined invalid operation exception is detected.

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

Inexact exception (I)

[Operation result] When FPSR.FS = 1

reg2	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
Operation result [exception]	–Normal	+Normal	–0	+0	–∞	+∞	Q-NaN	Q-NaN [V]

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

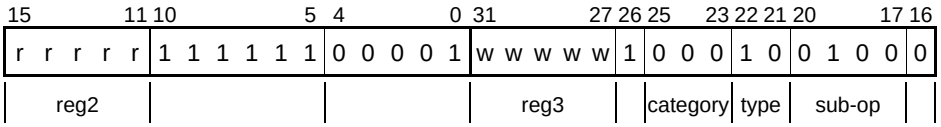
NEGF.S	Floating-point Negate (Single) Floating-point sign inversion (single precision)

[Instruction format] NEGF.S reg2, reg3

[Operation] reg3 ← neg reg2

[Format] Format F:I

[opcode]



[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. Sign inversion is performed arithmetically. This means that when the operand is S-NaN, an IEEE754-defined invalid operation exception is detected.

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

[Operation result] When FPSR.FS = 1

reg2	+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
Operation result [exception]	–Normal	+Normal	–0	+0	–∞	+∞	Q-NaN	Q-NaN [V]

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

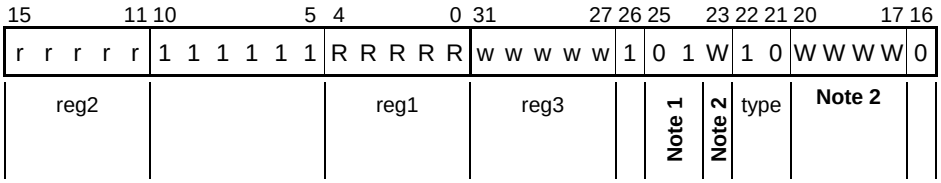
NMADDF.S	Floating-point Negate Multiply-add (Single) Floating-point fused-multiply-add operation (single precision)

[Instruction format] NMADDF.S reg1, reg2, reg3, reg4

[Operation] reg4 ← neg (reg2 × reg1 + reg3)

[Format] Format F:I

[opcode]



Notes 1. category

2. reg4 (The least significant bit of reg4 is bit 23.)

[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, then adds the result to the single-precision floating-point format contents of general-purpose register reg3. Next, it inverts the sign of the result and stores it in general-purpose register reg4. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

Sign inversion is performed arithmetically. This means that when the operand is S-NaN, an IEEE754-defined invalid operation exception is detected.

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

[Operation result] When FPSR.FS = 1

reg3(C) \ reg1 (A) \ reg2 (B)		+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
±Normal	+Normal	NMADD (A, B, C)				–∞	+∞		
	–Normal					+∞	–∞		
	±0					Q-NaN [V]			
	+∞	–∞	+∞	Q-NaN [V]		–∞	+∞		
	–∞	+∞	–∞			+∞	–∞		
±0	+Normal	NMADD (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. Denormalized numbers are flushed and handled as "0".

[Supplement] An exception occurs based on the multiplication result in the following case.

- When the multiplication operation results in an overflow and the addition operation does not cause an invalid operation exception.

<Floating-point instruction>

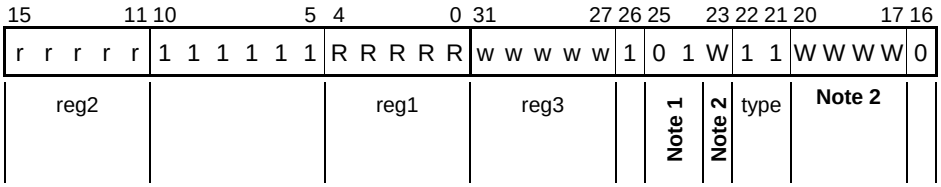
NMSUBF.S	Floating-point Negate Multiply-subtract (Single) Floating-point fused-multiply-add operation (single precision)

[Instruction format] NMSUBF.S reg1, reg2, reg3, reg4

[Operation] reg4 ← neg (reg2 × reg1-reg3)

[Format] Format F:I

[opcode]



Notes 1. category

2. reg4 (The least significant bit of reg4 is bit 23.)

[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, then subtracts from this result the single-precision floating-point format contents of general-purpose register reg3. Next, it inverts the sign of the result and stores it in general-purpose register reg4. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

Sign inversion is performed arithmetically. This means that when the operand is S-NaN, an IEEE754-defined invalid operation exception is detected.

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

Inexact exception (I)

Overflow exception (O)

Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg3(C) \ reg2 (B) / reg1 (A)		+Normal	–Normal	+0	–0	+∞	–∞	Q-NaN	S-NaN
±Normal	+Normal	NMSUB(A, B, C)				–∞	+∞		
	–Normal					+∞	–∞		
	±0					Q-NaN [V]			
	+∞	–∞	+∞	Q-NaN [V]		–∞	+∞		
	–∞	+∞	–∞			+∞	–∞		
±0	+Normal	NMSUB(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. Denormalized numbers are flushed and handled as "0".

[Supplement] An exception occurs based on the multiplication result in the following case.

- When the multiplication operation results in an overflow and the addition operation does not cause an invalid operation exception.

<Floating-point instruction>

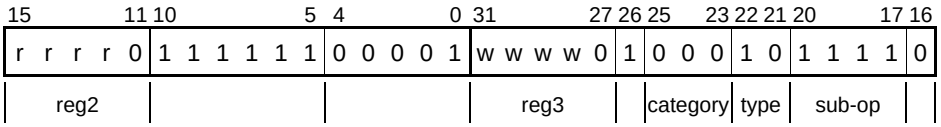
RECIPF.D	Reciprocal of a Floating-point Value (Double) Reciprocal (double precision)

[Instruction format] RECIPF.D reg2, reg3

[Operation] reg3 ← 1 ÷ reg2

[Format] Format F:I

[opcode]



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

[Supplement] When underflow exceptions are prohibited, the initial value is determined according to the sign of the intermediate value in the approximation calculation, so whether or not the sign will be the same as the source is not guaranteed.

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

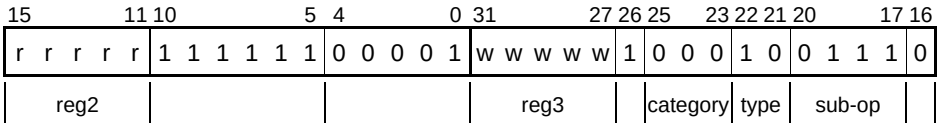
RECIPF.S	Reciprocal of a Floating-point Value (Single) Reciprocal (single precision)

[Instruction format] RECIPF.S reg2, reg3

[Operation] reg3 ← 1 ÷ reg2

[Format] Format F:I

[opcode]



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

[Supplement] When underflow exceptions are prohibited, the default is determined according to the sign of the intermediate value in the approximation calculation, so whether or not the sign will be the same as the source is not guaranteed.

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

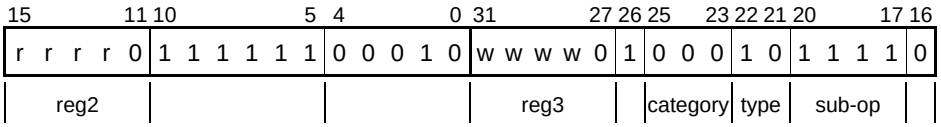
RSQRTF.D	Reciprocal of the Square Root of a Floating-point Value (Double) Reciprocal of square root (double precision)

[Instruction format] RSQRTF.D reg2, reg3

[Operation] reg3 ← 1 ÷ (sqrt reg2)

[Format] Format F:I

[opcode]



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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

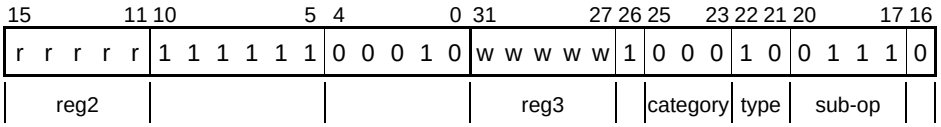
RSQRTF.S	Reciprocal of the Square Root of a Floating-point Value (Single) Reciprocal of square root (single precision)

[Instruction format] RSQRTF.S reg2, reg3

[Operation] reg3 ← 1 ÷ (sqrt reg2)

[Format] Format F:I

[opcode]



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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

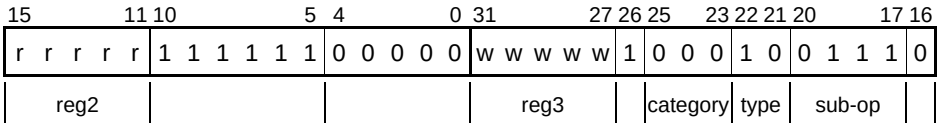
SQRTF.S	Floating-point Square Root (Single) Square root (single precision)

[Instruction format] SQRTF.S reg2, reg3

[Operation] reg3 ← sqrt reg2

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

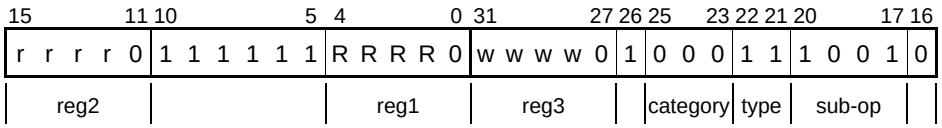
SUBF.D	Floating-point Subtract (Double) Floating-point subtraction (double precision)

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

[Operation] reg3 ← reg2 – reg1

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

<table>
<tr><th>reg1 (A) \ reg2 (B)</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="4" rowspan="4">B – A</td><td rowspan="4">+∞</td><td rowspan="5">–∞</td><td rowspan="7">Q-NaN</td><td rowspan="6"></td></tr>
<tr><td>–Normal</td></tr>
<tr><td>+0</td></tr>
<tr><td>–0</td></tr>
<tr><td>+∞</td><td colspan="4">–∞</td><td>Q-NaN [V]</td></tr>
<tr><td>–∞</td><td colspan="5">+∞</td><td>Q-NaN [V]</td></tr>
<tr><td>Q-NaN</td><td colspan="7">Q-NaN</td></tr>
<tr><td>S-NaN</td><td colspan="8">Q-NaN [V]</td></tr>
</table>

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

2. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

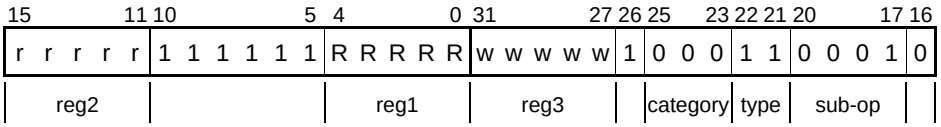
SUBF.S	Floating-point Subtract (Single) Floating-point subtraction (single precision)

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

[Operation] reg3 ← reg2 – reg1

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)
Overflow exception (O)
Underflow exception (U)

[Operation result] When FPSR.FS = 1

reg1 (A) \ reg2 (B)	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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

TRFSR	Transfer Floating Flags Flag transfer

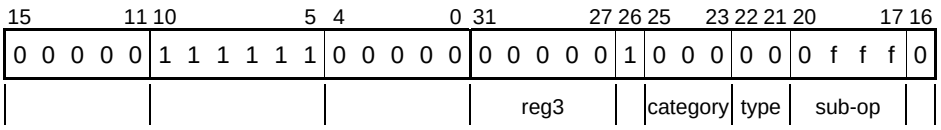
[Instruction format] TRFSR fcbit

TRFSR

[Operation] PSW.Z ← fcbit

[Format] Format F:I

[opcode]



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>

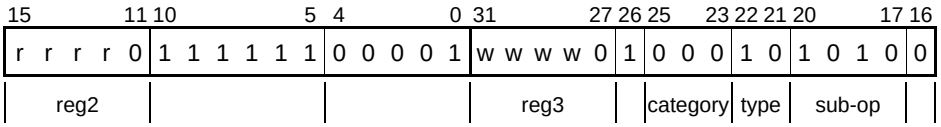
TRNCF.DL	Floating-point Truncate to Long Fixed-point Format, rounded to zero (Double) Conversion to fixed-point format (double precision)

[Instruction format] TRNCF.DL reg2, reg3

[Operation] reg3 ← trunc reg2 (double → long-word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

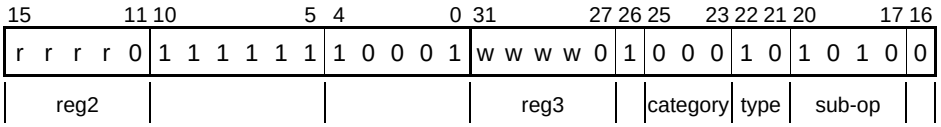
TRNCF.DUL	Floating-point Truncate to Unsigned Long Fixed-point Format, rounded to zero (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] TRNCF.DUL reg2, reg3

[Operation] reg3 ← trunc reg2 (double → unsigned long-word)

[Format] Format F:I

[opcode]



[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 $+\infty$: $2^{64} - 1$ is returned.
- Source is a negative number, not-a-number, or $-\infty$: 0 is returned.

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

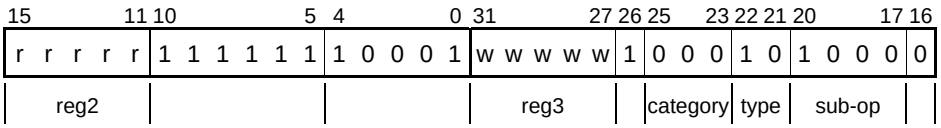
TRNCF.DUW	Floating-point Truncate to Unsigned Single Fixed-point Format, rounded to zero (Double) Conversion to unsigned fixed-point format (double precision)

[Instruction format] TRNCF.DUW reg2, reg3

[Operation] reg3 ← trunc reg2 (double → unsigned word)

[Format] Format F:I

[opcode]



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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

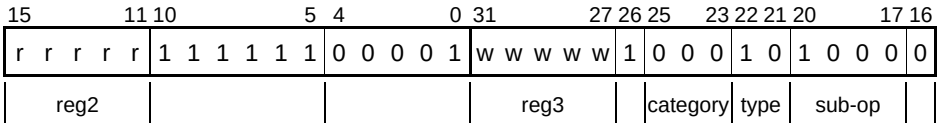
TRNCF.DW	Floating-point Truncate to Single Fixed-point Format, rounded to zero (Double) Conversion to fixed-point format (double precision)

[Instruction format] TRNCF.DW reg2, reg3

[Operation] reg3 ← trunc reg2 (double → word)

[Format] Format F:I

[opcode]



[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] Invalid operation exception (V)
Inexact exception (I)

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

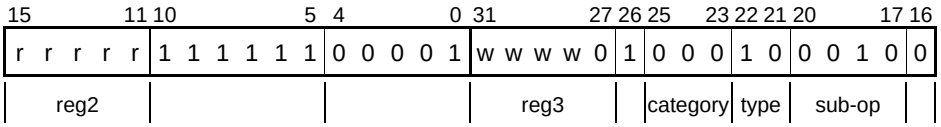
TRNCF.SL	Floating-point Truncate to Long Fixed-point Format, rounded to zero (Single) Conversion to fixed-point format (single precision)

[Instruction format] TRNCF.SL reg2, reg3

[Operation] reg3 ← trunc reg2 (single → long-word)

[Format] Format F:I

[opcode]



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

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

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to -2^{63}, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

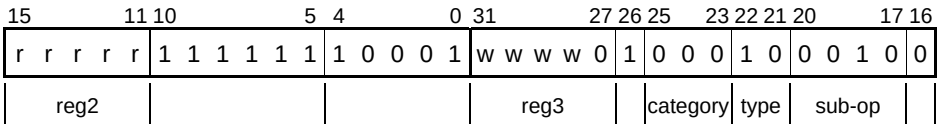
TRNCF.SUL	Floating-point Truncate to Unsigned Long Fixed-point Format, rounded to zero (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] TRNCF.SUL reg2, reg3

[Operation] reg3 ← trunc reg2 (single → unsigned long-word)

[Format] Format F:I

[opcode]



[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 $+\infty$: $2^{64} - 1$ is returned.
- Source is a negative number, not-a-number, or $-\infty$: 0 is returned.

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

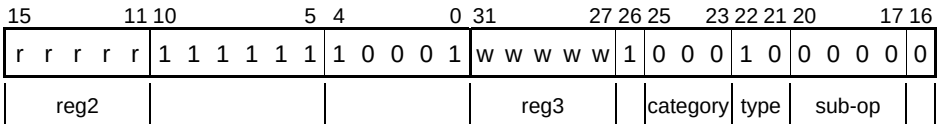
TRNCF.SUW	Floating-point Truncate to Unsigned Single Fixed-point Format, rounded to zero (Single) Conversion to unsigned fixed-point format (single precision)

[Instruction format] TRNCF.SUW reg2, reg3

[Operation] reg3 ← trunc reg2 (single → unsigned word)

[Format] Format F:I

[opcode]



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

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

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of $2^{32} - 1$ to 0, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

<Floating-point instruction>

TRNCF.SW	Floating-point Truncate to Single Fixed-point Format, rounded to 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 number format contents of general-purpose register reg2 to 32-bit fixed-point format, and stores the result in general-purpose register reg3.

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

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{31} - 1$ to -2^{31}, an IEEE754-defined invalid operation exception is detected.

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

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

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

[Operation result] When FPSR.FS = 1

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. Denormalized numbers are flushed and handled as “0”.

CHAPTER 5 FLOATING-POINT OPERATION EXCEPTIONS

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

5.1 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.

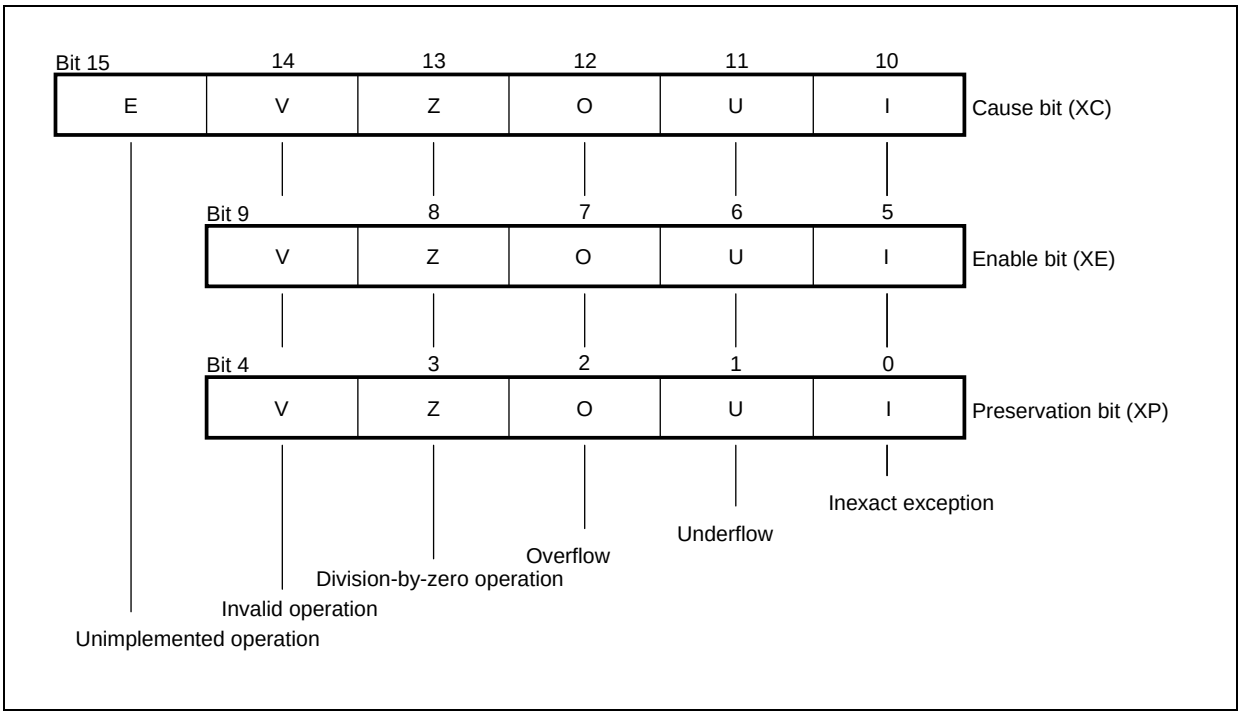
The FPU uses cause bits, enable bits, and preservation bits (status flags) to support the following five types of IEEE754-defined exception causes.

- 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 5-1 shows the FPSR register bits that are used to support exceptions.

Figure 5-1. 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.

5.2 Exception Processing

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

5.2.1 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 a 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 5-1 lists the initial values provided for each of the FPU IEEE754-defined exceptions.

Table 5-1. FPU Initial Values for IEEE754-defined Exceptions

Area	Description	Rounding mode	Initial value
V	Invalid operation	–	Uses quiet not-a-number (Q-NaN)
Z	Division-by-zero	–	Uses correctly signed ∞
O	Overflow	RN	∞ with sign of intermediate result
U	Underflow	RN	0 with sign of intermediate result
I	Inexact operation	–	Uses rounded result

5.3 Exception Details

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

5.3.1 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 denormalized 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

(1) 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.

(2) 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.

5.3.2 Invalid operation exception (V)

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

- Addition/subtraction or fused-multiply-add operation[Note]:
  Addition or subtraction of infinite values (+∞) + (–∞) or (–∞) – (–∞)
- Multiplication or fused-multiply-add operation: ±0 × ±∞
- Division: ±0 ÷ ±0 or ±∞ ÷ ±∞
- Comparison: With condition codes 8 15, if the operand is unordered (see **Table 4-2 Definitions of Condition Code Bits and Their Logical Inversions**)
- Arithmetic operation with S-Nan included in operands. The conditional transfer instruction (cmov) is not handled as an arithmetic operation, but the absolute value (ABS), arithmetic negation (NEG), minimum value (MIN), and maximum value (MAX) operations are handled as arithmetic operations.
- Comparison when operand includes S-NaN and conversion to floating-point format.
- Conversion to integer when source is outside of integer range.
- Square root: When operand is less than 0

Note When the multiplication result is rounded to infinity or when adding or subtracting between infinities, i.e., (+∞) + (–∞) or (–∞) – (–∞)

(1) 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.

(2) 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, refer to the operation result description of each instruction for the value to be stored in the destination register.

5.3.3 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.

(1) 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.

(2) If exception is not enabled

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

5.3.4 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.

(1) 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.

(2) 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 5-1 FPU Initial Values for IEEE754-defined Exceptions**).

5.3.5 Underflow exception (U)

An underflow exception is detected in the following two cases.

- If the operation result is -2^{Emin} to $+2^{Emin}$ (but not zero)
- If precision is dropped due to an operation between small numbers that are not normalized

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.

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

This FPU detects an underflow before rounding.

The following two methods can be used to detect a drop in precision.

- Denormalized loss (when a given result differs from the result calculated when the exponent range is infinite)
- Inexact result (when a given result differs from the result calculated when the exponent range and precision are infinite)

This FPU detects a drop in precision as an inexact result.

(1) 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.

(2) 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 sign is stored in the destination register (see **Table 5-1 FPU Initial Values for IEEE754-defined Exceptions**).

5.3.6 Unimplemented operation exception (E)

The E bit is set and an unimplemented operation exception occurs when an attempt is made to execute an instruction using an operation code that is reserved for future expansion. The operand and destination register contents do not change. Usually, the unimplemented instruction is emulated by software. If an IEEE754-defined exception occurs from an emulated operation, that exception should be emulated.

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

5.4 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 for both single-precision instructions and double-precision instructions. Precise exceptions can be specified to occur by setting the SEM and DEM bits of the FPSR register. The SEM bit specifies a single-precision instruction exception mode and the DEM bit specifies a double-precision instruction exception mode. For a description of single-precision instructions and double-precision instructions, see **4.2 Overview of Floating-point Instructions**.

5.4.1 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.

5.4.2 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 PSW.NP bit and ID bit as “0”, the pending exception is acknowledged.

5.5 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 0x00000071 for a precise exception and 0x00000072 for an imprecise exception. For the sake of compatibility, the lower 16 bits of the ECR register is also used to store the lower 16 bits of the exception code.

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, ECR, 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 override if another floating-point operation exception occurred. 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.

With the floating-point operation exception handler, the PR bit of the FPSR register can be checked to determine whether a precise exception or imprecise exception has occurred.

The cause bits and PR bit 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.

When accessing the FPSR or FPEPC register, note with caution that the FPU function system register bank must be selected via the BSEL register. If an exception occurs during exception processing, the exception processing may overwrite the BSEL register. Therefore, it is recommended that the BSEL register be saved as a program context prior to exception processing.

An example of the floating-point operation exception handler code is shown below.

```
000     .offset   0x70          -- FPU exception
001  jr _fpu_exception_handler
002  ...
003
004  _fpu_exception_handler:
005  -- Save basic context
006     addi      -100, sp, sp
007     st.w      r31, 96[sp]
008     st.w      r1, 92[sp]
009     stsr      31, r1        -- Save BSEL
010     st.w      r1, 8[sp]
011     ldsr      zero, 31      -- Select CPU bank
012     stsr      13, r1        -- Save EIIC
013     st.w      r1, 28[sp]
014     stsr      1, r1         -- Save EIPSW
015     st.w      r1, 24[sp]
016     stsr      0, r1         -- Save EIPC
017     st.w      r1, 20[sp]
018     ei                      -- EI level maskable exception enable
019     stsr      17, r1        -- Save CTPSW
020     st.w      r1, 16[sp]
021     stsr      16, r1        -- Save CTPC
```

```
022     st.w     r1, 12[sp]
023     movea    0x2000, r0, r1 -- Select FPU status bank
024     ldsr     r1, 31
025     stsr     6, r1          -- Save FPSR
026     st.w     r1, 4[sp]
027     stsr     7, r1          -- Save FPEPC
028     st.w     r1, 0[sp]
029
030  --  Determination of precise exception or imprecise exception
031     ld.w     28[sp], r1
032     addi     -0x72, r1, r0  -- Is EIIC exception cause code 0x72?
033     be       _imprecise_fpe
034
035  --  Floating-point operation exception processing
036
037  -- Basic context restoration
038     movea    0x2000, r0, r1 -- Select FPU status bank
039     ldsr     r1, 31
040     ld.w     0[sp], r31
041     ld.w     4[sp], r1
042     ldsr     r31, 7         -- FPEPC restoration
043     ldsr     r1, 6          -- FPSR restoration
044     ldsr     zero, 31       -- Select CPU bank
045     ld.w     12[sp], r31
046     ld.w     16[sp], r1
047     ldsr     r31, 16        -- CTPC restoration
048     ldsr     r1, 17         -- CTPSW restoration
049     di                      -- EI level maskable exception prohibited
050     ld.w     24[sp], r31
051     ld.w     28[sp], r1
052     ldsr     r31, 1         -- EIPSW restoration
053     ldsr     r1, 1          -- EIIC restoration
054  -- Set restoration address (for precise exception)
055     ld.w     20[sp], r1     -- Load EIPC to be saved
056     add      4, r1          -- Add 4 to r1 (all FPU instructions are four bytes)
057     ldsr     r1, 0          -- Set r1 (next instruction PC) to EIPC
058     ld.w     8[sp], r1      -- BSEL restoration
059     ldsr     r1, 31
060     ld.w     92[sp], r1
061     ld.w     96[sp], r31
062     addi     100, sp, sp
063     eiret
064
065  _imprecise_fpe:
066  -- Basic context restoration
067          ... Omitted ...
068
069  -- Store restoration point address to EIPC
070     ld.w     8[sp], r1      -- BSEL restoration
071     ldsr     r1, 31
072     ld.w     92[sp], r1
073     addi     100, sp, sp
074     eiret
```

5.6 Selection of Floating-point Operation Model

5.6.1 When accurate operations are required

With this FPU, when an attempt is made to execute an operation code that is reserved for future expansion or an instruction with invalid format code, the E bit is set and an unimplemented operation exception (E) occurs. The operand and destination register do not change.

When strictly accurate operations are required, any instruction where an exception occurs is emulated by software. If IEEE754-defined exceptions occur during an emulated operation, these exceptions should be emulated. If the program must be resumed after emulation, set the SEM and DEM bits of the FPSR register and specify the precise exception in advance.

IEEE754 recommends an exception handler that can store calculation results in the destination register regardless of which of the five standard exceptions occurs.

The exception handler can search instructions using the FPEPC register in order to determine the following.

- Instruction being executed
- Format of destination

To obtain the correctly rounded result when an overflow exception, underflow exception (except conversion instruction), or inexact exception occurs, the exception handler must have software that checks the source register and emulates instructions.

If an invalid operation exception or division-by-zero exception occurs or if an overflow exception or underflow exception occurs during floating-point conversion, the exception handler should have software that can obtain the value of the operand by checking the source register of the instruction.

IEEE754 recommends that, if possible, the overflow and underflow exceptions should have a priority higher than inexact exceptions. This priority is set by software. The hardware sets the bits of both the overflow and the underflow exceptions, as well as the inexact exception.

5.6.2 When operation performance is emphasized

For this FPU, one operation model emphasizes processing performance by preventing as much as possible the occurrence of exceptions due to execution of floating-point operations. For applications that require real-time processing or cases where operations do not have to be strictly precise, this model can be used to eliminate the emulation processing overhead related to exceptions that occur.

The following settings are recommended when operation performance is emphasized.

- Clear (0) the FPSR register enable bit to prohibit exceptions.
- Use single-precision floating-point format for processing that does not require high precision.
- Use the SEM and DEM bits of the FPSR register to specify imprecise exception mode.

During processing of operations, prohibit exceptions when floating-point operation exceptions can be ignored, so that the operation can continue using initial values. Single-precision instructions generally require a small number of execution clocks (low latency).

For this FPU, an unimplemented operation exception (E) does not occur when the FS bit of the FPSR register is set (1) and flushing of denormalized numbers is enabled. When flushing is enabled, flushing occurs before storage even when the operation result has underflowed, so the results are not held as denormalized results.

APPENDIX A LIST OF INSTRUCTIONS

A.1 Basic Instructions

Table A-1 shows an alphabetized list of basic instruction functions.

Table A-1. Basic Instruction Function List (Alphabetic Order) (1/4)

Mnemonic	Operand	Format	Flag					Function of Instruction
			CY	OV	S	Z	SAT	
ADD	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Add
ADD	imm5, reg2	II	0/1	0/1	0/1	0/1	–	Add
ADDI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	–	Add
ADF	cccc, reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	–	Conditional add
AND	reg1, reg2	I	–	0	0/1	0/1	–	AND
ANDI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	AND
Bcond	disp9	III	–	–	–	–	–	Conditional branch
BSH	reg2, reg3	XII	0/1	0	0/1	0/1	–	Byte swap of halfword data
BSW	reg2, reg3	XII	0/1	0	0/1	0/1	–	Byte swap of word data
CALLT	imm6	II	–	–	–	–	–	Subroutine call with table look up
CAXI	[reg1], reg2, reg3	IX	0/1	0/1	0/1	0/1	–	Comparison and swap
CLR1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Bit clear
CLR1	reg2, [reg1]	IX	–	–	–	0/1	–	Bit clear
CMOV	cccc, reg1, reg2, reg3	XI	–	–	–	–	–	Conditional transfer
CMOV	cccc, imm5, reg2, reg3	XII	–	–	–	–	–	Conditional transfer
CMP	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Comparison
CMP	imm5, reg2	II	0/1	0/1	0/1	0/1	–	Comparison
CTRET	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from subroutine call
DI	(None)	X	–	–	–	–	–	Disable EI level maskable exception
DISPOSE	imm5, list12	XIII	–	–	–	–	–	Stack frame deletion
DISPOSE	imm5, list12, [reg1]	XIII	–	–	–	–	–	Stack frame deletion
DIV	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (signed) word data
DIVH	reg1, reg2	I	–	0/1	0/1	0/1	–	Division of (signed) halfword data
DIVH	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (signed) halfword data.
DIVHU	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (unsigned) halfword data
DIVQ	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (signed) word data (variable steps)
DIVQU	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (unsigned) word data (variable steps)
DIVU	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Division of (unsigned) word data

Table A-1. Basic Instruction Function List (Alphabetic Order) (2/4)

Mnemonic	Operand	Format	Flag					Function of Instruction
			CY	OV	S	Z	SAT	
EI	(None)	X	–	–	–	–	–	Enable EI level maskable exception
EIRET	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from EI level exception
FERET	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from FE level exception
FETRAP	vector	I	–	–	–	–	–	FE level software exception instruction
HALT	(None)	X	–	–	–	–	–	Halt
HSH	reg2, reg3	XII	0/1	0	0/1	0/1	–	Halfword swap of halfword data
HSW	reg2, reg3	XII	0/1	0	0/1	0/1	–	Halfword swap of word data
JARL	disp22, reg2	V	–	–	–	–	–	Branch and register link
JARL	disp32, reg1	VI	–	–	–	–	–	Branch and register link
JMP	[reg1]	I	–	–	–	–	–	Unconditional branch (register relative)
JMP	disp32 [reg1]	VI	–	–	–	–	–	Unconditional branch (register relative)
JR	disp22	V	–	–	–	–	–	Unconditional branch (PC relative)
JR	disp32	VI	–	–	–	–	–	Unconditional branch (PC relative)
LD.B	disp16 [reg1], reg2	VII	–	–	–	–	–	Load of (signed) byte data
LD.B	disp23 [reg1], reg3	XIV	–	–	–	–	–	Load of (signed) byte data
LD.BU	disp16 [reg1], reg2	VII	–	–	–	–	–	Load of (unsigned) byte data
LD.BU	disp23 [reg1], reg3	XIV	–	–	–	–	–	Load of (unsigned) byte data
LD.H	disp16 [reg1], reg2	VII	–	–	–	–	–	Load of (signed) halfword data
LD.H	disp23 [reg1], reg3	XIV	–	–	–	–	–	Load of (signed) halfword data
LD.HU	disp16 [reg1], reg2	VII	–	–	–	–	–	Load of (unsigned) halfword data
LD.HU	disp23 [reg1], reg3	XIV	–	–	–	–	–	Load of (unsigned) halfword data
LD.W	disp16 [reg1], reg2	VII	–	–	–	–	–	Load of word data
LD.W	disp23 [reg1], reg3	XIV	–	–	–	–		Load of word data
LDSR	reg2, regID	IX	–	–	–	–	–	Load to system register
MAC	reg1, reg2, reg3, reg4	XI	–	–	–	–	–	Multiply-accumulate for (signed) word data
MACU	reg1, reg2, reg3, reg4	XI	–	–	–	–	–	Multiply-accumulate for (unsigned) word data
MOV	reg1, reg2	I	–	–	–	–	–	Data transfer
MOV	imm5, reg2	II	–	–	–	–	–	Data transfer
MOV	imm32, reg1	VI	–	–	–	–	–	Data transfer
MOVEA	imm16, reg1, reg2	VI	–	–	–	–	–	Effective address transfer
MOVHI	imm16, reg1, reg2	VI	–	–	–	–	–	Higher halfword transfer
MUL	reg1, reg2, reg3	XI	–	–	–	–	–	Multiplication of (signed) word data
MUL	imm9, reg2, reg3	XII	–	–	–	–	–	Multiplication of (signed) word data
MULH	reg1, reg2	I	–	–	–	–	–	Multiplication of (signed) halfword data
MULH	imm5, reg2	II	–	–	–	–	–	Multiplication of (signed) halfword data
MULHI	imm16, reg1, reg2	VI	–	–	–	–	–	Multiplication of (signed) halfword immediate data

Table A-1. Basic Instruction Function List (Alphabetic Order) (3/4)

Mnemonic	Operand	Format	Flag					Function of Instruction
			CY	OV	S	Z	SAT	
MULU	reg1, reg2, reg3	XI	–	–	–	–	–	Multiplication of (unsigned) word data
MULU	imm9, reg2, reg3	XII	–	–	–	–	–	Multiplication of (unsigned) word data
NOP	(None)	I	–	–	–	–	–	Nothing else is done.
NOT	reg1, reg2	I	–	0	0/1	0/1	–	Logical negation (1's complement)
NOT1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	NOT bit
NOT1	reg2, [reg1]	IX	–	–	–	0/1	–	NOT bit
OR	reg1, reg2	I	–	0	0/1	0/1	–	OR
ORI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	OR immediate
PREPARE	list12, imm5	XIII	–	–	–	–	–	Create stack frame
PREPARE	list12, imm5, sp/imm	XIII	–	–	–	–	–	Create stack frame
RETI	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from EI level software exception or interrupt
RIE	(None)	I/X	–	–	–	–	–	Reserved instruction exception
SAR	reg1, reg2	IX	0/1	0	0/1	0/1	–	Arithmetic right shift
SAR	imm5, reg2	II	0/1	0	0/1	0/1	–	Arithmetic right shift
SAR	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	–	Arithmetic right shift
SASF	cccc, reg2	IX	–	–	–	–	–	Shift and flag condition setting
SATADD	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated addition
SATADD	imm5, reg2	II	0/1	0/1	0/1	0/1	0/1	Saturated addition
SATADD	reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	0/1	Saturated addition
SATSUB	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated subtraction
SATSUB	reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	0/1	Saturated subtraction
SATSUBI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	0/1	Saturated subtraction
SATSUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated reverse subtraction
SBF	cccc, reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	–	Conditional subtraction
SCH0L	reg2, reg3	IX	0/1	0	0	0/1	–	Bit (0) search from MSB side
SCH0R	reg2, reg3	IX	0/1	0	0	0/1	–	Bit (0) search from LSB side
SCH1L	reg2, reg3	IX	0/1	0	0	0/1	–	Bit (1) search from MSB side
SCH1R	reg2, reg3	IX	0/1	0	0	0/1	–	Bit (1) search from LSB side
SET1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Bit setting
SET1	reg2, [reg1]	IX	–	–	–	0/1	–	Bit setting
SETF	cccc, reg2	IX	–	–	–	–	–	Flag condition setting
SHL	reg1, reg2	IX	0/1	0	0/1	0/1	–	Logical left shift
SHL	imm5, reg2	II	0/1	0	0/1	0/1	–	Logical left shift
SHL	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	–	Logical left shift

Table A-1. Basic Instruction Function List (Alphabetic Order) (4/4)

Mnemonic	Operand	Format	Flag					Function of Instruction
			CY	OV	S	Z	SAT	
SHR	reg1, reg2	IX	0/1	0	0/1	0/1	–	Logical right shift
SHR	imm5, reg2	II	0/1	0	0/1	0/1	–	Logical right shift
SHR	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	–	Logical right shift
SLD.B	disp7 [ep], reg2	IV	–	–	–	–	–	Load of (signed) byte data
SLD.BU	disp4 [ep], reg2	IV	–	–	–	–	–	Load of (unsigned) byte data
SLD.H	disp8 [ep], reg2	IV	–	–	–	–	–	Load of (signed) halfword data
SLD.HU	disp5 [ep], reg2	IV	–	–	–	–	–	Load of (unsigned) halfword data
SLD.W	disp8 [ep], reg2	IV	–	–	–	–	–	Load of word data
SST.B	reg2, disp7 [ep]	IV	–	–	–	–	–	Storage of byte data
SST.H	reg2, disp8 [ep]	IV	–	–	–	–	–	Storage of halfword data
SST.W	reg2, disp8 [ep]	IV	–	–	–	–	–	Storage of word data
ST.B	reg2, disp16 [reg1]	VII	–	–	–	–	–	Storage of byte data
ST.B	reg3, disp23 [reg1]	XIV	–	–	–	–	–	Storage of byte data
ST.H	reg2, disp16 [reg1]	VII	–	–	–	–	–	Storage of halfword data
ST.H	reg3, disp23 [reg1]	XIV	–	–	–	–	–	Storage of halfword data
ST.W	reg2, disp16 [reg1]	VII	–	–	–	–	–	Storage of word data
ST.W	reg3, disp23 [reg1]	XIV	–	–	–	–	–	Storage of word data
STSR	regID, reg2	IX	–	–	–	–	–	Storage of contents of system register
SUB	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Subtraction
SUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Reverse subtraction
SWITCH	reg1	I	–	–	–	–	–	Jump with table look up
SXB	reg1	I	–	–	–	–	–	Sign-extension of byte data
SXH	reg1	I	–	–	–	–	–	Sign-extension of halfword data
SYNCE	(None)	I	–	–	–	–	–	Exception synchronize instruction
SYNCM	(None)	I	–	–	–	–	–	Memory synchronize instruction
SYNCP	(None)	I	–	–	–	–	–	Pipeline synchronize instruction
SYSCALL	vector8	X	–	–	–	–	–	System call exception
TRAP	vector5	X	–	–	–	–	–	Software exception
TST	reg1, reg2	I	–	0	0/1	0/1	–	Test
TST1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Bit test
TST1	reg2, [reg1]	IX	–	–	–	0/1	–	Bit test
XOR	reg1, reg2	I	–	0	0/1	0/1	–	Exclusive OR
XORI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	Exclusive OR immediate
ZXB	reg1	I	–	–	–	–	–	Zero-extension of byte data
ZXH	reg1	I	–	–	–	–	–	Zero-extension of halfword data

A.2 Floating-point Instructions

Table A-2 shows a list of floating-point instruction functions.

Table A-2. Floating-point Instruction Function List (1/3)

Mnemonic	Operand	Format	Function of Instruction
ABSF.D	reg2, reg3	Double precision	Floating-point absolute value
ABSF.S	reg2, reg3	Single precision	Floating-point absolute value
ADDF.D	reg1, reg2, reg3	Double precision	Floating-point add
ADDF.S	reg1, reg2, reg3	Single precision	Floating-point add
CEILF.DL	reg2, reg3	Double precision	Conversion to fixed-point format
CEILF.DUL	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CEILF.DUW	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CEILF.DW	reg2, reg3	Double precision	Conversion to fixed-point format
CEILF.SL	reg2, reg3	Single precision	Conversion to fixed-point format
CEILF.SUL	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CEILF.SUW	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CEILF.SW	reg2, reg3	Single precision	Conversion to fixed-point format
CMOVF.D	cc, reg1, reg2, reg3	Double precision	Conditional transfer
CMOVF.S	cc, reg1, reg2, reg3	Single precision	Conditional transfer
CMPF.D	cond, reg2, reg1, fcbit cond, reg1, reg2	Double precision	Floating-point comparison
CMPF.S	cond, reg2, reg1, fcbit cond, reg2, reg1	Single precision	Floating-point comparison
CVTF.DL	reg2, reg3	Double precision	Conversion to fixed-point format
CVTF.DS	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.DUL	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CVTF.DUW	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
CVTF.DW	reg2, reg3	Double precision	Conversion to fixed-point format
CVTF.LD	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.LS	reg2, reg3	Single precision	Conversion to floating-point format
CVTF.SD	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.SL	reg2, reg3	Single precision	Conversion to fixed-point format
CVTF.SUL	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
CVTF.SUW	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
CVTF.SW	reg2, reg3	Single precision	Conversion to fixed-point format
CVTF.ULD	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.ULS	reg2, reg3	Single precision	Conversion to floating-point format
CVTF.UWD	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.UWS	reg2, reg3	Single precision	Conversion to floating-point format
CVTF.WD	reg2, reg3	Double precision	Conversion to floating-point format
CVTF.WS	reg2, reg3	Single precision	Conversion to floating-point format

Table A-2. Floating-point Instruction Function List (2/3)

Mnemonic	Operand	Format	Function of Instruction
DIVF.D	reg1, reg2, reg3	Double precision	Floating-point division
DIVF.S	reg1, reg2, reg3	Single precision	Floating-point division
FLOORF.DL	reg2, reg3	Double precision	Conversion to fixed-point format
FLOORF.DUL	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
FLOORF.DUW	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
FLOORF.DW	reg2, reg3	Double precision	Conversion to fixed-point format
FLOORF.SL	reg2, reg3	Single precision	Conversion to fixed-point format
FLOORF.SUL	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
FLOORF.SUW	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
FLOORF.SW	reg2, reg3	Single precision	Conversion to fixed-point format
MADDF.S	reg1, reg2, reg3, reg4	Single precision	Floating-point fused-multiply-add operation
MAXF.D	reg1, reg2, reg3	Double precision	Floating-point maximum value
MAXF.S	reg1, reg2, reg3	Single precision	Floating-point maximum value
MINF.D	reg1, reg2, reg3	Double precision	Floating-point minimum value
MINF.S	reg1, reg2, reg3	Single precision	Floating-point minimum value
MSUBF.S	reg1, reg2, reg3, reg4	Single precision	Floating-point fused-multiply-add operation
MULF.D	reg1, reg2, reg3	Double precision	Floating-point multiplication
MULF.S	reg1, reg2, reg3	Single precision	Floating-point multiplication
NEGF.D	reg2, reg3	Double precision	Floating-point sign inversion
NEGF.S	reg2, reg3	Single precision	Floating-point sign inversion
NMADDF.S	reg1, reg2, reg3, reg4	Single precision	Floating-point fused-multiply-add operation
NMSUBF.S	reg1, reg2, reg3, reg4	Single precision	Floating-point fused-multiply-add operation
RECIPF.D	reg2, reg3	Double precision	Reciprocal
RECIPF.S	reg2, reg3	Single precision	Reciprocal
RSQRTF.D	reg2, reg3	Double precision	Reciprocal of square root
RSQRTF.S	reg2, reg3	Single precision	Reciprocal of square root
SQRTF.D	reg2, reg3	Double precision	Square root
SQRTF.S	reg2, reg3	Single precision	Square root
SUBF.D	reg1, reg2, reg3	Double precision	Floating-point subtraction
SUBF.S	reg1, reg2, reg3	Single precision	Floating-point subtraction
TRFSR	cc#3	Single precision	Flag transfer
TRNCF.DL	reg2, reg3	Double precision	Conversion to fixed-point format
TRNCF.DUL	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
TRNCF.DUW	reg2, reg3	Double precision	Conversion to unsigned fixed-point format
TRNCF.DW	reg2, reg3	Double precision	Conversion to fixed-point format
TRNCF.SL	reg2, reg3	Single precision	Conversion to fixed-point format
TRNCF.SUL	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
TRNCF.SUW	reg2, reg3	Single precision	Conversion to unsigned fixed-point format
TRNCF.SW	reg2, reg3	Single precision	Conversion to fixed-point format

APPENDIX B INSTRUCTION OPCODE MAP

B.1 Basic Instruction Opcode Map

The following shows opcode maps for the basic instruction code.

Table B-1. Basic Instruction Opcode Map (16-/32-bit Instruction) (1/4)

Mnemonic	Operand	Format	opcode							Remark
			15 11	10 5	4 0	31 27	26 21	20 16		
NOP		I	00000	000000	00000					
SYNCE		I	00000	000000	11101					
SYNCM		I	00000	000000	11110					
SYNCP		I	00000	000000	11111					
MOV	reg1, reg2	I	rrrrr	000000	RRRRR					rrrrr ≠ 00000
NOT	reg1, reg2	I	rrrrr	000001	RRRRR					
RIE		I	00000	0 0 0 1 0	00000					
SWITCH	reg1	I	00000	000010	RRRRR					
FETRAP	vector4	I	0iiii	000010	00000					iiii ≠ 0000
DIVH	reg1, reg2	I	rrrrr	000010	RRRRR					rrrrr ≠ 00000, RRRRR ≠ 00000
JMP	[reg1]	I	000000	000011	RRRRR					
SLD.BU	disp4 [ep], reg2	IV	rrrrr	000011	0dddd					rrrrr ≠ 00000
SLD.HU	disp5 [ep],reg2	IV	rrrrr	000011	1dddd					rrrrr ≠ 00000
ZXB	reg1	I	00000	000100	RRRRR					
SXB	reg1	I	00000	000101	RRRRR					
ZXH	reg1	I	00000	000110	RRRRR					
SXH	reg1	I	00000	000111	RRRRR					
SATSUBR	reg1, reg2	I	rrrrr	000100	RRRRR					rrrrr ≠ 00000
SATSUB	reg1, reg2	I	rrrrr	000101	RRRRR					rrrrr ≠ 00000
SATADD	reg1, reg2	I	rrrrr	000110	RRRRR					rrrrr ≠ 00000
MULH	reg1, reg2	I	rrrrr	000111	RRRRR					rrrrr ≠ 00000
OR	reg1, reg2	I	rrrrr	001000	RRRRR					
XOR	reg1, reg2	I	rrrrr	001001	RRRRR					
AND	reg1, reg2	I	rrrrr	001010	RRRRR					
TST	reg1, reg2	I	rrrrr	001011	RRRRR					
SUBR	reg1, reg2	I	rrrrr	001100	RRRRR					
SUB	reg1, reg2	I	rrrrr	001101	RRRRR					
ADD	reg1, reg2	I	rrrrr	001110	RRRRR					
CMP	reg1, reg2	I	rrrrr	001111	RRRRR					
MOV	imm5, reg2	I	rrrrr	010000	iiiii					rrrrr ≠ 00000

Table B-1. Basic Instruction Opcode Map (16-/32-bit Instruction) (2/4)

Mnemonic	Operand	Format	opcode						Remark
			15 11	10 5	4 0	31 27	26 21	20 16	
SATADD	imm5, reg2	I	r r r r r	0 1 0 0 0 1	i i i i i				rrrrr ≠ 00000
ADD	imm5, reg2	I	r r r r r	0 1 0 0 1 0	i i i i i				
CMP	imm5, reg2	I	r r r r r	0 1 0 0 1 1	i i i i i				
CALLT	imm6	II	0 0 0 0 0	0 1 0 0 0 i	i i i i i				
SHR	imm5, reg2	II	r r r r r	0 1 0 1 0 0	i i i i i				
SAR	imm5, reg2	II	r r r r r	0 1 0 1 0 1	i i i i i				
SHL	imm5, reg2	II	r r r r r	0 1 0 1 1 0	i i i i i				
MULH	imm5, reg2	II	r r r r r	0 1 0 1 1 1	i i i i i				rrrrr ≠ 00000
JR	disp32	VI	0 0 0 0 0	0 1 0 1 1 1	0 0 0 0 0	d d d d d	d d d d d d	d d d d 0	See **Table B-2**
JARL	disp32, reg1	VI	0 0 0 0 0	0 1 0 1 1 1	R R R R R	d d d d d	d d d d d d	d d d d 0	See **Table B-2** RRRRR ≠ 00000
SLD.B	disp7 [ep], reg2	IV	r r r r r	0 1 1 0 d d	d d d d d				
SST.B	reg2, disp7 [ep]	IV	r r r r r	0 1 1 1 d d	d d d d d				
SLD.H	disp8 [ep], reg2	IV	r r r r r	1 0 0 0 d d	d d d d d				
SST.H	reg2, disp8 [ep]	IV	r r r r r	1 0 0 1 d d	d d d d d				
SLD.W	disp8 [ep], reg2	IV	r r r r r	1 0 1 0 d d	d d d d 0				
SST.W	reg2, disp8 [ep]	IV	r r r r r	1 0 1 0 d d	d d d d 1				
Bcond	disp9	III	d d d d d	1 0 1 1 d d	d C C C C				
ADDI	imm16, reg1, reg2	VI	r r r r r	1 1 0 0 0 0	R R R R R	i i i i i	i i i i i i	i i i i i	
MOV	imm32, reg1	VI	0 0 0 0 0	1 1 0 0 0 1	R R R R R	I I I I I	I I I I I I	I I I I I	See **Table B-2**
MOVEA	imm16, reg1, reg2	VI	r r r r r	1 1 0 0 0 1	R R R R R	i i i i i	i i i i i i	i i i i i	rrrrr ≠ 00000
MOVHI	imm16, reg1, reg2	VI	r r r r r	1 1 0 0 1 0	R R R R R	i i i i i	i i i i i i	i i i i i	rrrrr ≠ 00000
SATSUBI	imm16, reg1, reg2	VI	r r r r r	1 1 0 0 1 1	R R R R R	i i i i i	i i i i i i	i i i i i	rrrrr ≠ 00000
DISPOSE	imm5, list12	XIII	0 0 0 0 0	1 1 0 0 1 i	i i i i L	L L L L L	L L L L L L	0 0 0 0	
DISPOSE	imm5, list12, [reg1]	XIII	0 0 0 0 0	1 1 0 0 1 i	i i i i L	L L L L L	L L L L L L	R R R R	RRRRR ≠ 00000
ORI	imm16, reg1, reg2	VI	r r r r r	1 1 0 1 0 0	R R R R R	i i i i i	i i i i i i	i i i i i	
XORI	imm16, reg1, reg2	VI	r r r r r	1 1 0 1 0 1	R R R R R	i i i i i	i i i i i i	i i i i i	
ANDI	imm16, reg1, reg2	VI	r r r r r	1 1 0 1 1 0	R R R R R	i i i i i	i i i i i i	i i i i i	
MULHI	imm16, reg1, reg2	VI	r r r r r	1 1 0 1 1 1	R R R R R	i i i i i	i i i i i i	i i i i i	rrrrr ≠ 00000
JMP	imm32 [reg1]	VI	0 0 0 0 0	1 1 0 1 1 1	R R R R R	d d d d d	d d d d d d	d d d d 0	See **Table B-2**
LD.B	disp16 [reg1], reg2	VII	r r r r r	1 1 1 0 0 0	R R R R R	d d d d d	d d d d d d	d d d d d	
LD.H	disp16 [reg1], reg2	VII	r r r r r	1 1 1 0 0 1	R R R R R	d d d d d	d d d d d d	d d d d 0	
LD.W	disp16 [reg1], reg2	VII	r r r r r	1 1 1 0 0 1	R R R R R	d d d d d	d d d d d d	d d d d 1	
ST.B	reg2, disp16 [reg1]	VII	r r r r r	1 1 1 0 1 0	R R R R R	d d d d d	d d d d d d	d d d d d	
ST.H	reg2, disp16 [reg1]	VII	r r r r r	1 1 1 0 1 1	R R R R R	d d d d d	d d d d d d	d d d d 0	
ST.W	reg2, disp16 [reg1]	VII	r r r r r	1 1 1 0 1 1	R R R R R	d d d d d	d d d d d d	d d d d 1	
PREPARE	list12, imm5	XIII	0 0 0 0 0	1 1 1 1 0 i	i i i i L	L L L L L	L L L L L L	0 0 0 1	
PREPARE	list12, imm5, sp/imm	XIII	0 0 0 0 0	1 1 1 1 0 i	i i i i i L	L L L L L	L L L L L L	f f 0 1 1	**Note**

Note See **Table B-2** when ff = 01 or 10, and see **Table B-3** when ff = 11.

Table B-1. Basic Instruction Opcode Map (16-/32-bit Instruction) (3/4)

Mnemonic	Operand	Format	opcode							Remark
			15 11	10 5	4 0	31 27	26 21	20 16		
LD.B	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R R	w w w w w	d d d d d d	d 0 1 0 1	See **Table B-2**	
LD.H	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R R	w w w w w	d d d d d d	0 0 1 1 1	See **Table B-2**	
LD.W	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R R	w w w w w	d d d d d d	0 1 0 0 1	See **Table B-2**	
ST.B	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R R	w w w w w	d d d d d d	d 1 1 0 1	See **Table B-2**	
ST.W	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R R	w w w w w	d d d d d d	0 1 1 1 1	See **Table B-2**	
LD.BU	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R R	w w w w w	d d d d d d	d 0 1 0 1	See **Table B-2**	
LD.HU	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R R	w w w w w	d d d d d d	0 0 1 1 1	See **Table B-2**	
ST.H	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R R	w w w w w	d d d d d d	0 1 1 0 1	See **Table B-2**	
JR	disp22	V	0 0 0 0 0	1 1 1 1 0 D	D D D D D	d d d d d	d d d d d d	d d d d 0		
JARL	disp22, reg2	V	r r r r r	1 1 1 1 0 D	D D D D D	d d d d d	d d d d d d	d d d d 0	r r r r r ≠ 0 0 0 0 0	
LD.BU	disp16 [reg1], reg2	VII	r r r r r	1 1 1 1 0 b	R R R R R	d d d d d	d d d d d d	d d d d 1		
SET1	bit3#, disp16 [reg1]	VIII	0 0 b b b	1 1 1 1 1 0	R R R R R	d d d d d	d d d d d d	d d d d d		
NOT1	bit#3, disp16 [reg1]	VIII	0 1 b b b	1 1 1 1 1 0	R R R R R	d d d d d	d d d d d d	d d d d d		
CLR1	bit3#, disp16 [reg1]	VIII	1 0 b b b	1 1 1 1 1 0	R R R R R	d d d d d	d d d d d d	d d d d d		
TST1	bit3#, disp16 [reg1]	VIII	1 1 b b b	1 1 1 1 1 0	R R R R R	d d d d d	d d d d d d	d d d d d		
LD.HU	disp16 [reg1], reg2	VII	r r r r r	1 1 1 1 1 1	R R R R R	d d d d d	d d d d d d	d d d d 1	r r r r r ≠ 0 0 0 0 0	
SETF	cond, reg2	IX	r r r r r	1 1 1 1 1 1	0 C C C C	0 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0		
RIE		X	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		
LDSR	reg2, regID	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 0 0 1	0 0 0 0 0		
STSR	sr1, reg2	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 0 1 0	0 0 0 0 0		
SHR	reg1, reg2	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 0 0	0 0 0 0 0		
SHR	reg1, reg2, reg3	IX	r r r r r	1 1 1 1 1 1	R R R R R	w w w w w	0 0 0 1 0 0	0 0 0 1 0		
SAR	reg1, reg2	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 0 1	0 0 0 0 0		
SAR	reg1, reg2, reg3	XI	r r r r r	1 1 1 1 1 1	R R R R R	w w w w w	0 0 0 1 0 1	0 0 0 1 0		
SHL	reg1, reg2	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 1 0	0 0 0 0 0		
SHL	reg1, reg2, reg3	XI	r r r r r	1 1 1 1 1 1	R R R R R	w w w w w	0 0 0 1 1 0	0 0 0 1 0		
SET1	reg2, [reg1]	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 1 1	0 0 0 0 0		
NOT1	reg2, [reg1]	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 1 1	0 0 0 1 0		
CLR1	reg2, [reg1]	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 1 1	0 0 1 0 0		
TST1	reg2, [reg1]	IX	r r r r r	1 1 1 1 1 1	R R R R R	0 0 0 0 0	0 0 0 1 1 1	0 0 1 1 0		
CAXI	[reg1], reg2, reg3	XI	r r r r r	1 1 1 1 1 1	R R R R R	w w w w w	0 0 0 1 1 1	0 1 1 1 0		
TRAP	imm5	X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 0	0 0 0 0 0		
HALT		X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 0 1	0 0 0 0 0		
RETI		X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 1 0	0 0 0 0 0		
CTRET		X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 1 0	0 0 1 0 0		
EIRET		X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 1 0	0 1 0 0 0		
FERET		X	0 0 0 0 0	1 1 1 1 1 1	i i i i i	0 0 0 0 0	0 0 1 0 1 0	0 1 0 1 0		
DI		X	0 0 0 0 0	1 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 1 0 1 1	0 0 0 0 0		
EI		X	1 0 0 0 0	1 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 1 0 1 1	0 0 0 0 0		

Table B-1. Basic Instruction Opcode Map (16-/32-bit Instruction) (4/4)

Mnemonic	Operand	Format	opcode						Remark
			15 11	10 5	4 0	31 27	26 21	20 16	
SYSCALL	vector8	X	11010	111111	vvvvv	00VVV	001011	00000	
SASF	cccc, reg2	IX	rrrrr	111111	0cccc	00000	010000	00000	
MUL	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010001	00000	
MULU	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010001	00010	
MUL	imm9, reg2, reg3	XII	rrrrr	111111	iiiii	wwwww	01001I	III00	
MULU	imm9, reg2, reg3	XII	rrrrr	111111	iiiii	wwwww	01001I	III10	
DIVH	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010100	00000	
DIVHU	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010100	00010	
DIV	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010110	00000	
DIVQ	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010111	11100	
DIVU	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010110	00010	
DIVQU	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	010111	11110	
CMOV	cccc, imm5, reg2, reg3	XI	rrrrr	111111	iiiii	wwwww	011000	cccc0	
CMOV	cccc, reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	011001	cccc0	
BSW	reg2, reg3	XII	rrrrr	111111	00000	wwwww	011010	00000	
BSH	reg2, reg3	XII	rrrrr	111111	00000	wwwww	011010	00010	
HSW	reg2, reg3	XII	rrrrr	111111	00000	wwwww	011010	00100	
HSH	reg2, reg3	XII	rrrrr	111111	00000	wwwww	011010	00110	
SCH0R	reg2, reg3	IX	rrrrr	111111	00000	wwwww	011011	00000	
SCH1R	reg2, reg3	IX	rrrrr	111111	00000	wwwww	011011	00010	
SCH0L	reg2, reg3	IX	rrrrr	111111	00000	wwwww	011011	00100	
SCH1L	reg2, reg3	IX	rrrrr	111111	00000	wwwww	011011	00110	
SBF	cccc, reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	011100	cccc0	cccc ≠ 1101
SATSUB	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	011100	11010	
ADF	cccc, reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	011101	cccc0	cccc ≠ 1101
SATADD	reg1, reg2, reg3	XI	rrrrr	111111	RRRRR	wwwww	011101	11010	
MAC	reg1, reg2, reg3, reg4	XI	rrrrr	111111	RRRRR	wwww0	011110	mmmm0	
MACU	reg1, reg2, reg3, reg4	XI	rrrrr	111111	RRRRR	wwww0	011111	mmmm0	

Table B-2. Basic Instruction Opcodes (48-bit Instructions)

Mnemonic	Operand	Format	opcode									
			15 11	10 5	4 0	31 27	26 21	20 16	47 43	42 37	36 32
JR	disp32	VI	00000	010111	00000	ddddd	dddddd	dddd0	DDDDD	DDDDDD	DDDDD
JARL	disp32, reg1	VI	00000	010111	RRRRR	ddddd	dddddd	dddd0	DDDDD	DDDDDD	DDDDD
MOV	imm32, reg1	VI	00000	110001	RRRRR	iiiii	iiiii	iiiii	IIIII	IIIIII	IIIII
JMP	disp32 [reg1]	VI	00000	110111	RRRRR	ddddd	dddddd	dddd0	DDDDD	DDDDDD	DDDDD
PREPARE	list12, imm5, sp/imm	XIII	00000	11110i	iiiiiL	LLLLL	LLLLLL	ff[Note]011	IIIII	IIIIII	IIIII
LD.B	disp23[reg1], reg3	XIV	00000	111100	RRRRRR	wwwww	dddddd	d0101	DDDDD	DDDDDD	DDDDD
LD.H	disp23[reg1], reg3	XIV	00000	111100	RRRRRR	wwwww	dddddd	00111	DDDDD	DDDDDD	DDDDD
LD.W	disp23[reg1], reg3	XIV	00000	111100	RRRRRR	wwwww	dddddd	01001	DDDDD	DDDDDD	DDDDD
ST.B	reg3, disp23[reg1]	XIV	00000	111100	RRRRRR	wwwww	dddddd	d1101	DDDDD	DDDDDD	DDDDD
ST.W	reg3, disp23[reg1]	XIV	00000	111100	RRRRRR	wwwww	dddddd	01111	DDDDD	DDDDDD	DDDDD
LD.BU	disp23[reg1], reg3	XIV	00000	111101	RRRRRR	wwwww	dddddd	d0101	DDDDD	DDDDDD	DDDDD
LD.HU	disp23[reg1], reg3	XIV	00000	111101	RRRRRR	wwwww	dddddd	00111	DDDDD	DDDDDD	DDDDD
ST.H	reg3, disp23[reg1]	XIV	00000	111101	RRRRRR	wwwww	dddddd	01101	DDDDD	DDDDDD	DDDDD

Note ff = 01, 10

Table B-3. Basic Instruction Opcode Map (64-bit Instruction)

Mnemonic	Operand	Format	opcode						Remark
			15 11	10 5	4 0	31 27	26 21	20 16	
PREPARE	list12, imm5, sp/imm	XIII	00000	11110i	iiiiiL	LLLLL	LLLLLL	11011	
			47 32			63 48			
			IIIII	IIIIII	IIIII	IIIII	IIIIII	IIIII	

B.2 Floating-point Instruction Opcode Map

The following is an opcode map for floating-point instruction codes.

Table B-4. Floating-point Instruction Opcodes (1/2)

Mnemonic	Operand	Format	opcode						Remark
			15 11	10 5	4 0	31 27	26 21	20 16	
ABSF.D	reg2, reg3	Double precision	rrrr0	111111	00000	wwww0	100010	11000	
ABSF.S	reg2, reg3	Single precision	rrrrr	111111	00000	wwwww	100010	01000	
ADDF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	10000	
ADDF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	00000	
CEILF.DL	reg2, reg3	Double precision	rrrr0	111111	00010	wwww0	100010	10100	
CEILF.DUL	reg2, reg3	Double precision	rrrr0	111111	10010	wwww0	100010	10100	
CEILF.DUW	reg2, reg3	Double precision	rrrr0	111111	10010	wwwww	100010	10000	
CEILF.DW	reg2, reg3	Double precision	rrrr0	111111	00010	wwwww	100010	10000	
CEILF.SL	reg2, reg3	Single precision	rrrrr	111111	00010	wwww0	100010	00100	
CEILF.SUL	reg2, reg3	Single precision	rrrrr	111111	10010	wwww0	100010	00100	
CEILF.SUW	reg2, reg3	Single precision	rrrrr	111111	10010	wwwww	100010	00000	
CEILF.SW	reg2, reg3	Single precision	rrrrr	111111	00010	wwwww	100010	00000	
CMOVF.D	cc, reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100000	1fff0	wwww ≠ 0000
CMOVF.S	cc, reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100000	0fff0	wwwww ≠ 00000
CMPF.D	cond, reg2, reg1, cc#3	Double precision	rrrr0	111111	RRRRR	0FFFF	100001	1fff0	
CMPF.S	cond, reg1, reg2, cc#3	Single precision	rrrrr	111111	RRRRR	0FFFF	100001	0fff0	
CVTF.DL	reg2, reg3	Double precision	rrrr0	111111	00100	wwww0	100010	10100	
CVTF.DS	reg2, reg3	Double precision	rrrr0	111111	00011	wwwww	100010	10010	
CVTF.DUL	reg2, reg3	Double precision	rrrr0	111111	10100	wwww0	100010	10100	
CVTF.DUW	reg2, reg3	Double precision	rrrr0	111111	10100	wwwww	100010	10000	
CVTF.DW	reg2, reg3	Double precision	rrrr0	111111	00100	wwwww	100010	10000	
CVTF.LD	reg2, reg3	Double precision	rrrr0	111111	00001	wwww0	100010	10010	
CVTF.LS	reg2, reg3	Single precision	rrrr0	111111	00001	wwwww	100010	00010	
CVTF.SD	reg2, reg3	Double precision	rrrrr	111111	00010	wwww0	100010	10010	
CVTF.SL	reg2, reg3	Single precision	rrrrr	111111	00100	wwww0	100010	00100	
CVTF.SUL	reg2, reg3	Single precision	rrrrr	111111	10100	wwww0	100010	00100	
CVTF.SUW	reg2, reg3	Single precision	rrrrr	111111	10100	wwwww	100010	00000	
CVTF.SW	reg2, reg3	Single precision	rrrrr	111111	00100	wwwww	100010	00000	
CVTF.ULD	reg2, reg3	Double precision	rrrr0	111111	10001	wwww0	100010	10010	
CVTF.ULS	reg2, reg3	Single precision	rrrr0	111111	10001	wwwww	100010	00010	
CVTF.UWD	reg2, reg3	Double precision	rrrrr	111111	10000	wwww0	100010	10010	
CVTF.UWS	reg2, reg3	Single precision	rrrrr	111111	10000	wwwww	100010	00010	
CVTF.WD	reg2, reg3	Double precision	rrrrr	111111	00000	wwww0	100010	10010	
CVTF.WS	reg2, reg3	Single precision	rrrrr	111111	00000	wwwww	100010	00010	
DIVF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	11110	
DIVF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	01110	

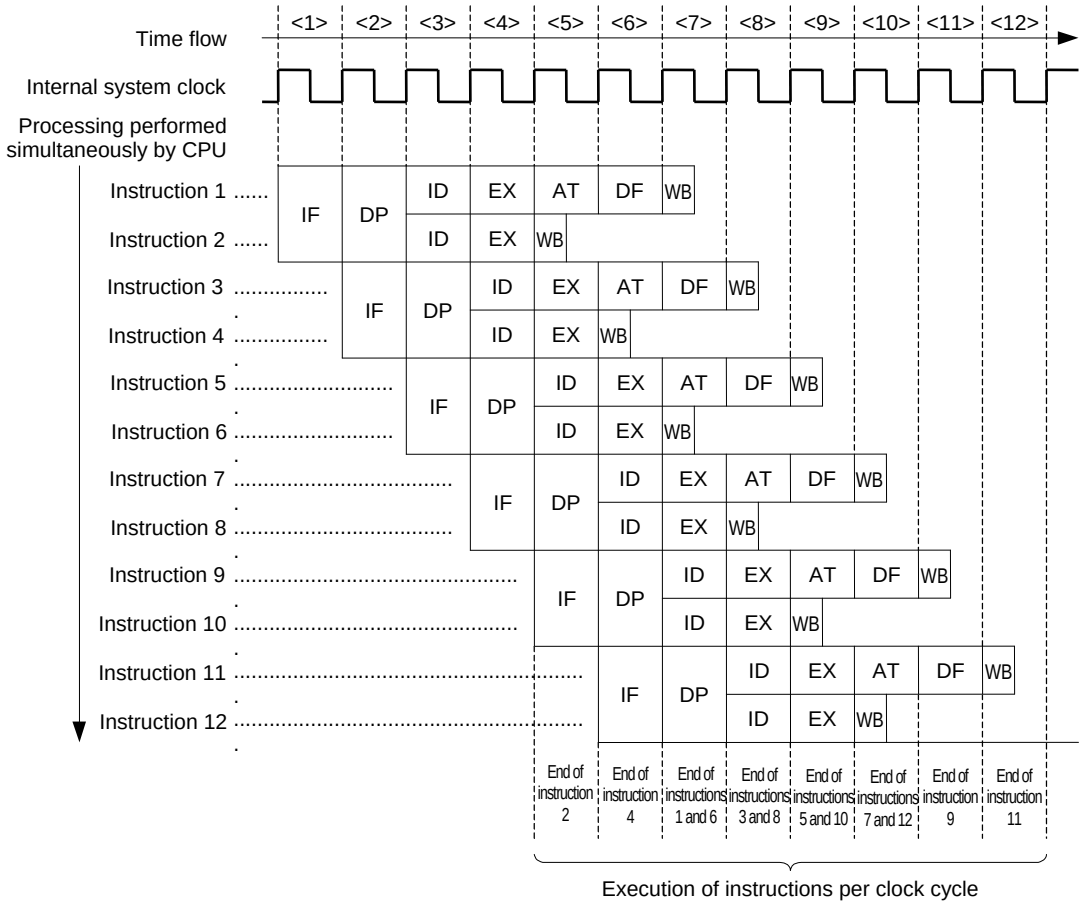
Table B-4. Floating-point Instruction Opcodes (2/2)

Mnemonic	Operand	Format	opcode						Remark
			15 11	10 5	4 0	31 27	26 21	20 16	
FLOORF.DL	reg2, reg3	Double precision	rrrr0	111111	00011	wwww0	100010	10100	
FLOORF.DUL	reg2, reg3	Double precision	rrrr0	111111	10011	wwww0	100010	10100	
FLOORF.DUW	reg2, reg3	Double precision	rrrr0	111111	10011	wwwww	100010	10000	
FLOORF.DW	reg2, reg3	Double precision	rrrr0	111111	00011	wwwww	100010	10000	
FLOORF.SL	reg2, reg3	Single precision	rrrrr	111111	00011	wwww0	100010	00100	
FLOORF.SUL	reg2, reg3	Single precision	rrrrr	111111	10011	wwww0	100010	00100	
FLOORF.SUW	reg2, reg3	Single precision	rrrrr	111111	10011	wwwww	100010	00000	
FLOORF.SW	reg2, reg3	Single precision	rrrrr	111111	00011	wwwww	100010	00000	
MADDF.S	reg1, reg2, reg3, reg4	Single precision	rrrrr	111111	RRRRR	wwwww	101W00	WWWW0	
MAXF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	11000	
MAXF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	01000	
MINF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	11010	
MINF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	01010	
MSUBF.S	reg1, reg2, reg3, reg4	Single precision	rrrrr	111111	RRRRR	wwwww	101W01	WWWW0	
MULF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	10100	
MULF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	00100	
NEGF.D	reg2, reg3	Double precision	rrrr0	111111	00001	wwww0	100010	11000	
NEGF.S	reg2, reg3	Single precision	rrrrr	111111	00001	wwwww	100010	01000	
NMADDF.S	reg1, reg2, reg3, reg4	Single precision	rrrrr	111111	RRRRR	wwwww	101W10	WWWW0	
NMSUBF.S	reg1, reg2, reg3, reg4	Single precision	rrrrr	111111	RRRRR	wwwww	101W11	WWWW0	
RECIPF.D	reg2, reg3	Double precision	rrrr0	111111	00001	wwww0	100010	11110	
RECIPF.S	reg2, reg3	Single precision	rrrrr	111111	00001	wwwww	100010	01110	
RSQRTF.D	reg2, reg3	Double precision	rrrr0	111111	00010	wwww0	100010	11110	
RSQRTF.S	reg2, reg3	Single precision	rrrrr	111111	00010	wwwww	100010	01110	
SQRTF.D	reg2, reg3	Double precision	rrrr0	111111	00000	wwww0	100010	11110	
SQRTF.S	reg2, reg3	Single precision	rrrrr	111111	00000	wwwww	100010	01110	
SUBF.D	reg1, reg2, reg3	Double precision	rrrr0	111111	RRRR0	wwww0	100011	10010	
SUBF.S	reg1, reg2, reg3	Single precision	rrrrr	111111	RRRRR	wwwww	100011	00010	
TRFSR	cc#3	Single precision	00000	111111	00000	00000	100000	0fff0	
TRNCF.DL	reg2, reg3	Double precision	rrrr0	111111	00001	wwww0	100010	10100	
TRNCF.DUL	reg2, reg3	Double precision	rrrr0	111111	10001	wwww0	100010	10100	
TRNCF.DUW	reg2, reg3	Double precision	rrrr0	111111	10001	wwwww	100010	10000	
TRNCF.DW	reg2, reg3	Double precision	rrrr0	111111	00001	wwwww	100010	10000	
TRNCF.SL	reg2, reg3	Single precision	rrrrr	111111	00001	wwww0	100010	00100	
TRNCF.SUL	reg2, reg3	Single precision	rrrrr	111111	10001	wwww0	100010	00100	
TRNCF.SUW	reg2, reg3	Single precision	rrrrr	111111	10001	wwwww	100010	00000	
TRNCF.SW	reg2, reg3	Single precision	rrrrr	111111	00001	wwwww	100010	00000	

APPENDIX C PIPELINES

The V850E2M CPU, which is based on RISC architecture, uses seven-stage pipeline control to execute almost all types of instructions in just one clock cycle. The instruction execution sequence normally includes seven stages, from instruction fetch (IF) to writeback (WB). The execution time per stage differs depending on factors such as the type of instruction and the type of memory to be accessed. As an example of pipeline operations, Figure C-1 shows processing by the CPU when 12 typical instructions are executed consecutively.

Figure C-1. Example of Consecutive Execution of 12 Typical Instructions



IF (instruction fetch):	Fetches instruction and increments fetch pointer.
DP (dispatch):	Checks instruction type and dependencies, then sends to corresponding pipeline.
ID (instruction decode):	Decodes instruction, generates immediate data, and reads registers.
EX (execution using ALU, multiplier, or barrel shifter):	Executes decoded instructions.
AT (address transfer):	Transfers address to target memory.
DF (data fetch):	Reads data from target memory.
WB (writeback):	Writes execution result to register.

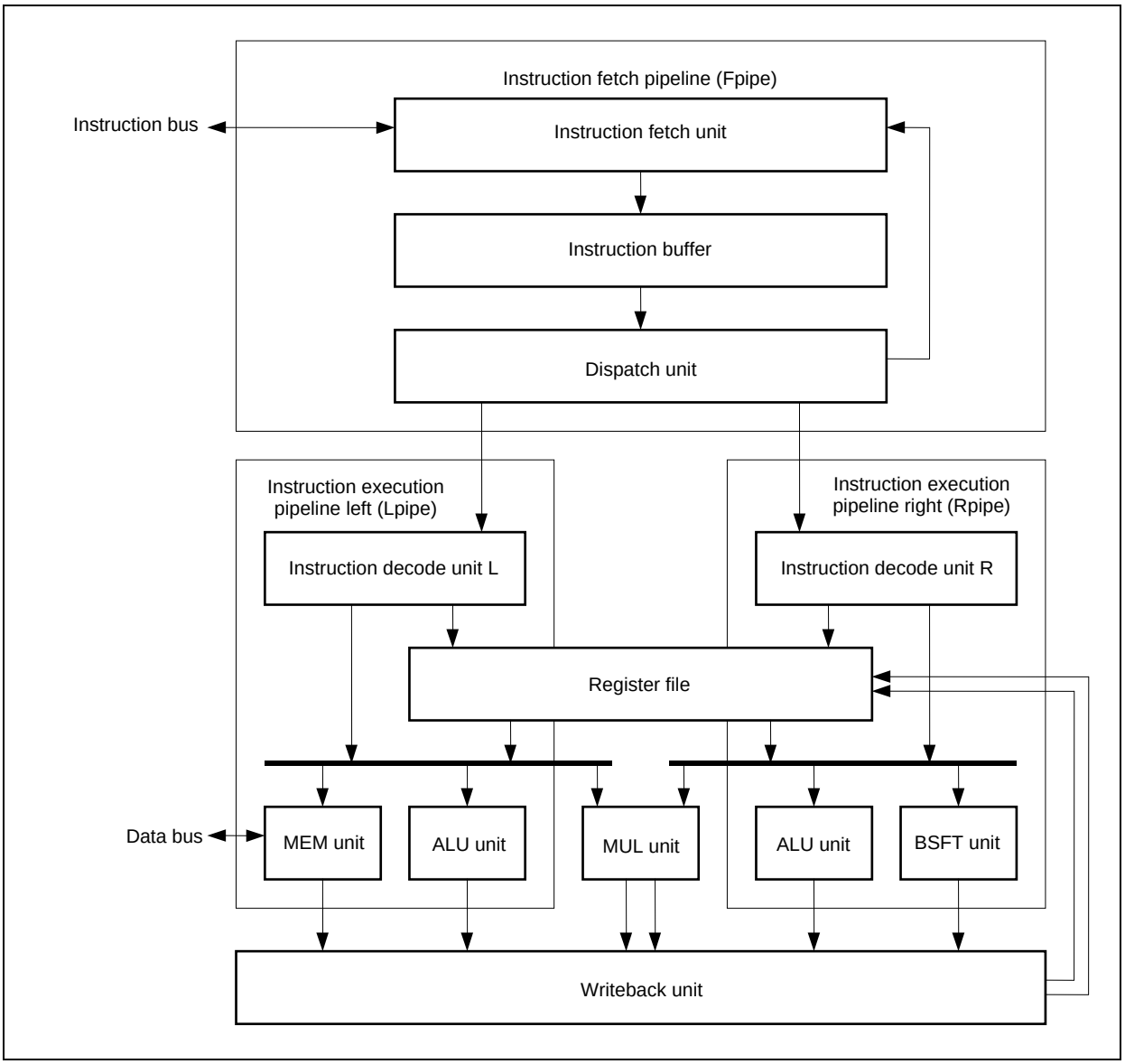
<1> to <12> are CPU states. For typical instructions, two instructions can be executed (EX) in parallel per clock cycle.

C.1 Features

Figure C-2 shows the pipeline configuration of the CPU assumed by the V850E2M CPU. The CPU has the following three independent pipelines, detects dependency relationship of instructions, and can issue up to two instructions at the same time.

- Instruction fetch pipeline (Fpipe)
- Instruction execution pipeline left (Lpipe)
- Instruction execution pipeline right (Rpipe)

Figure C-2. Pipeline Configuration



(1) Instruction fetch pipeline (Fpipe)

This pipeline includes the following three units.

(a) Instruction fetch unit

Up to eight instructions (when each instruction is 16 bits) can be fetched from a 128-bit fetch bus (iLB) during one cycle.

(b) Dispatch unit

This unit includes a 128-bit × 3-stage instruction queue, which is used to detect instruction dependencies, and which enables up to two instructions to be efficiently issued at once to the instruction execution pipeline.

(c) Instruction buffer

Fetched instructions are stored by the instruction fetch unit.

(2) Instruction execution pipeline left (Lpipe)

This pipeline includes the following three units.

(a) Instruction decode unit L

This unit decodes instructions issued from the dispatch unit.

(b) ALU unit

This unit issues instructions that perform integer operations and/or logic operations.

(c) MEM unit

This unit executes instructions (such as load and store instructions) that perform memory access.

(3) Instruction execution pipeline right (Rpipe)

This pipeline includes the following three units.

(a) Instruction decode unit R

This unit decodes instructions issued from the dispatch unit.

(b) ALU unit

This unit issues instructions that perform integer operations and/or logic operations.

(c) BSFT unit

This unit performs data manipulation instructions.

(4) MUL unit

This unit executes instructions that perform integer multiplications.

(5) Writeback unit

This unit controls writeback to register files.

C.2 Clock Requirements

C.2.1 Clock requirements for basic instructions

Table C-1 lists clock requirements for basic instructions. The clock requirements may differ according to the combination of instructions. For details, see **C.3 Pipeline for Basic Instructions**.

Table C-1. Clock Requirements for Basic Instructions (1/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks			Parallel Execution[Note 1]
				issue	repeat	latency	
Load instructions	LD.B	disp16 [reg1] , reg2	4	1	1	3[Note 2]	Yes (L)
	LD.B	disp23 [reg1] , reg3	6	1	1	3[Note 2]	No
	LD.BU	disp16 [reg1] , reg2	4	1	1	3[Note 2]	Yes (L)
	LD.BU	disp23 [reg1] , reg3	6	1	1	3[Note 2]	No
	LD.H	disp16 [reg1] , reg2	4	1	1	3[Note 2]	Yes (L)
	LD.H	disp23 [reg1] , reg3	6	1	1	3[Note 2]	No
	LD.HU	disp16 [reg1] , reg2	4	1	1	3[Note 2]	Yes (L)
	LD.HU	disp23 [reg1] , reg3	6	1	1	3[Note 2]	No
	LD.W	disp16 [reg1] , reg2	4	1	1	3[Note 2]	Yes (L)
	LD.W	disp23 [reg1] , reg3	6	1	1	3[Note 2]	No
	SLD.B	disp7 [ep] , reg2	2	1	1	3[Note 2]	Yes (L)
	SLD.BU	disp4 [ep] , reg2	2	1	1	3[Note 2]	Yes (L)
	SLD.H	disp8 [ep] , reg2	2	1	1	3[Note 2]	Yes (L)
	SLD.HU	disp5 [ep] , reg2	2	1	1	3[Note 2]	Yes (L)
	SLD.W	disp8 [ep] , reg2	2	1	1	3[Note 2]	Yes (L)
Store instructions	ST.B	reg2, disp16 [reg1]	4	1	1	1	Yes (L)
	ST.B	reg3, disp23 [reg1]	6	1	1	1	No
	ST.H	reg2, disp16 [reg1]	4	1	1	1	Yes (L)
	ST.H	reg3, disp23 [reg1]	6	1	1	1	No
	ST.W	reg2, disp16 [reg1]	4	1	1	1	Yes (L)
	ST.W	reg3, disp23 [reg1]	6	1	1	1	No
	SST.B	reg2, disp7 [ep]	2	1	1	1	Yes (L)
	SST.H	reg2, disp8 [ep]	2	1	1	1	Yes (L)
	SST.W	reg2, disp8 [ep]	2	1	1	1	Yes (L)
Multiply instructions	MUL	reg1, reg2, reg3	4	1	1	3	Yes (L)
	MUL	imm9, reg2, reg3	4	1	1	3	Yes (L)
	MULH	reg1, reg2	2	1	1	3	Yes (L)
	MULH	imm5, reg2	2	1	1	3	Yes (L)
	MULHI	imm16, reg1, reg2	4	1	1	3	Yes (L)
	MULU	reg1, reg2, reg3	4	1	1	3	Yes (L)
	MULU	imm9, reg2, reg3	4	1	1	3	Yes (L)
Multiply-accumulate instructions	MAC	reg1, reg2, reg3, reg4	4	1	1	3	No
	MACU	reg1, reg2, reg3, reg4	4	1	1	3	No

Table C-1. Clock Requirements for Basic Instructions (2/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks			Parallel Execution[Note 1]
				issue	repeat	latency	
Arithmetic operation instructions	ADD	reg1, reg2	2	1	1	1	Yes (R/L)
	ADD	imm5, reg2	2	1	1	1	Yes (R/L)
	ADDI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	CMP	reg1, reg2	2	1	1	1	Yes (R/L)
	CMP	imm5, reg2	2	1	1	1	Yes (R/L)
	MOV	reg1, reg2	2	1	1	1	Yes (R/L)
	MOV	imm5, reg2	2	1	1	1	Yes (R/L)
	MOV	imm32, reg1	6	1	1	1	No
Arithmetic operation instructions	MOVEA	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	MOVHI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	SUB	reg1, reg2	2	1	1	1	Yes (R/L)
	SUBR	reg1, reg2	2	1	1	1	Yes (R/L)
Conditional operation instructions	ADF	cccc, reg1, reg2, reg3	4	1	1	1	No
	SBF	cccc, reg1, reg2, reg3	4	1	1	1	No
Saturated operation instructions	SATADD	reg1, reg2	2	1	1	1	Yes (R/L)
	SATADD	imm5, reg2	2	1	1	1	Yes (R/L)
	SATADD	reg1, reg2, reg3	4	1	1	1	Yes (R/L)
	SATSUB	reg1, reg2	2	1	1	1	Yes (R/L)
	SATSUB	reg1, reg2, reg3	4	1	1	1	Yes (R/L)
	SATSUBI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	SATSUBR	reg1, reg2	2	1	1	1	Yes (R/L)
Logic operation instructions	AND	reg1, reg2	2	1	1	1	Yes (R/L)
	ANDI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	NOT	reg1, reg2	2	1	1	1	Yes (R/L)
	OR	reg1, reg2	2	1	1	1	Yes (R/L)
	ORI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
	TST	reg1, reg2	2	1	1	1	Yes (R/L)
	XOR	reg1, reg2	2	1	1	1	Yes (R/L)
	XORI	imm16, reg1, reg2	4	1	1	1	Yes (R/L)
Data manipulation instructions	BSH	reg2, reg3	4	1	1	1	Yes (R)
	BSW	reg2, reg3	4	1	1	1	Yes (R)
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1	Yes (R)
	CMOV	cccc, imm5, reg2, reg3	4	1	1	1	Yes (R)
	HSH	reg2, reg3	4	1	1	1	Yes (R)
	HSW	reg2, reg3	4	1	1	1	Yes (R)
	SAR	reg1, reg2	4	1	1	1	Yes (R)
	SAR	imm5, reg2	2	1	1	1	Yes (R)
	SAR	reg1, reg2, reg3	4	1	1	1	Yes (R)
	SASF	cccc, reg2	4	1	1	1	Yes (R)
	SETF	cccc, reg2	4	1	1	1	Yes (R)
	SHL	reg1, reg2	4	1	1	1	Yes (R)
	SHL	imm5, reg2	2	1	1	1	Yes (R)
	SHL	reg1, reg2, reg3	4	1	1	1	Yes (R)

Table C-1. Clock Requirements for Basic Instructions (3/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks			Parallel Execution[Note 1]
				issue	repeat	latency	
Data manipulation instructions	SHR	reg1, reg2	4	1	1	1	Yes (R)
	SHR	imm5, reg2	2	1	1	1	Yes (R)
	SHR	reg1, reg2, reg3	4	1	1	1	Yes (R)
Data manipulation instructions	SXB	reg1	2	1	1	1	No
	SXH	reg1	2	1	1	1	No
	ZXB	reg1	2	1	1	1	No
	ZXH	reg1	2	1	1	1	No
Bit search instructions	SCH0L	reg2, reg3	4	1	1	1	Yes (R)
	SCH0R	reg2, reg3	4	1	1	1	Yes (R)
	SCH1L	reg2, reg3	4	1	1	1	Yes (R)
	SCH1R	reg2, reg3	4	1	1	1	Yes (R)
Divide instructions	DIV	reg1, reg2, reg3	4	36	36	36	No
	DIVH	reg1, reg2	2	36	36	36	No
	DIVH	reg1, reg2, reg3	4	36	36	36	No
	DIVHU	reg1, reg2, reg3	4	35	35	35	No
	DIVU	reg1, reg2, reg3	4	35	35	35	No
High-speed divide instructions	DIVQ	reg1, reg2, reg3	4	N+5[Note 3]	N+5[Note 3]	N+5[Note 3]	No
	DIVQU	reg1, reg2, reg3	4	N+4[Note 3]	N+4[Note 3]	N+4[Note 3]	No
Branch instructions	Bcond	disp9 (when condition is met)	2	4[Note 4]	4[Note 4]	4[Note 4]	Yes (R/L)
		disp9 (when condition is not met)	2	1	1	1	Yes (R/L)
	JARL	disp22, reg2	4	4	4	4	Yes (R/L)
	JARL	disp32, reg1	6	4	4	4	No
	JMP	[reg1]	2	4	4	4	Yes (R/L)
	JMP	disp32 [reg1]	6	5	5	5	No
	JR	disp22	4	4	4	4	Yes (R/L)
	JR	disp32	6	4	4	4	No
Bit manipulation instructions	CLR1	bit#3, disp16 [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	CLR1	reg2, [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	NOT1	bit#3, disp16 [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	NOT1	reg2, [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	SET1	bit#3, disp16 [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	SET1	reg2, [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	TST1	bit#3, disp16 [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	TST1	reg2, [reg1]	4	4[Note 5]	4[Note 5]	4[Note 5]	No
Special instructions	CALLT	imm6	2	10	10	10	No
	CAXI	[reg1], reg2, reg3	4	4[Note 5]	4[Note 5]	4[Note 5]	No
	CTRET	–	4	7	7	7	No
	DI	–	4	2	2	2	No
	DISPOSE	imm5, list12	4	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	DISPOSE	imm5, list12, [reg1]	4	n+6[Note 6]	n+6[Note 6]	n+6[Note 6]	No
	EI	–	4	2	2	2	No
	EIRET	–	4	7	7	7	No
	FERET	–	4	7	7	7	No

Table C-1. Clock Requirements for Basic Instructions (4/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks			Parallel Execution[Note 1]
				issue	repeat	latency	
Special instruction	FETRAP	vector	2	7	7	7	No
	HALT	–	4	1	1	1	No
	LDSR	reg2, regID (BSEL register, MCC register)	4	4	4	4	No
		reg2, regID (MPU group (except for MCC register))	4	1	1	1	Yes (R)
		reg2, regID (FPU group, user group)	4	3	3	3	No
		reg2, regID (other defined bank)	4	2	2	2	No
	NOP	–	2	1	1	1	No
	PREPARE	list12, imm5	4	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	PREPARE	list12, imm5, sp	4	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	PREPARE	list12, imm5, imm16	6	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	PREPARE	list12, imm5, imm16<<16	6	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	PREPARE	list12, imm5, imm32	8	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]	No
	RETI	–	4	7	7	7	No
	RIE	–	4	7	7	7	No
	STSR	regID, reg2	4	1	1	1	No
	SWITCH	reg1	2	8	8	8	No
	SYNCE	–	2	Undefined	Undefined	Undefined	No
	SYNCM	–	2	Undefined	Undefined	Undefined	No
	SYNCP	–	2	Undefined	Undefined	Undefined	No
	SYSCALL	vector8	4	10	10	10	No
	TRAP	vector5	4	7	7	7	No
Undefined instruction code (operates as RIE instruction)			4	7	7	7	No

Notes 1. "Yes" indicates that the instruction can be issued in parallel with other instructions, "No" indicates that the instruction cannot be issued in parallel with other instructions (the instruction must be issued individually). The pipeline used for parallel issuance (L: Lpipe, R: Rpipe, R/L: Rpipe or Lpipe) is shown in parentheses. Instructions that use the same pipeline cannot be issued in parallel. For details, refer to **C.3 Pipeline for Basic Instructions**.

2. When there are no wait states (3 + number of read access wait states)

3. N = (Number of valid bits of dividend) – (Number of valid bits of divisor)
However, if N is negative, it is assumed that N = 0.

4. Four clocks even when an instruction that rewrites the PSW register content is placed immediately before. A parallel issuance is also possible even when the instruction immediately before rewrites the PSW register.

5. When there are no wait states (4 + number of read access wait states)

6. n is the total number of registers specified in list × (depends on the number of wait states. When there are no wait states, n matches with the number of registers specified in list ×).
Supplement: 4 clocks when n = 0 or 1 (8 clocks for DISPOSE instruction with JMP).

Remarks 1. Description of operands

Symbol	Description
reg1	General-purpose register (used as source register)
reg2	General-purpose register (mainly used as the destination register, but used as a source register for some instructions)
reg3	General-purpose register (mainly stores remainders from division results and the higher 32 bits from multiplication results)
bit#3	3-bit data for specifying bit number
imm ×	× bit immediate data
disp ×	× bit displacement data
regID	System register number
vector ×	Data specifying vector (× indicates the bit size)
cond	Condition name (see **Table 5-4 Condition Codes** in **PART 2**).
cccc	4-bit data indicating condition code (see **Table 5-4 Condition Codes** in **PART 2**)
sp	Stack pointer (r3)
ep	Element pointer (r30)
list12	Register list

2. Description of execution clocks

Symbol	Description
issue	When next instruction is executed immediately after previous instruction
repeat	When same instruction is executed again immediately after its first execution
latency	When execution result of the current instruction is used by the immediate next instruction

C.2.2 Clock requirements for floating-point instructions

Table C-2 lists the number of execution clocks for single-precision floating-point instructions, and Table C-3 lists the number of execution clocks for double-precision floating-point instructions. The clock requirements vary depending on the combination of instructions.

Table C-2. Clock Requirements for Single-precision Floating-point Instructions (1/2)

Mnemonic	Operand	Byte	No. of Execution Clocks						Parallel Execution[Note]
			Imprecise			Precise			
			issue	repeat	latency	issue	repeat	latency	
ABSF.S	reg2, reg3	4	1	1	4	4	4	4	No
ADDF.S	reg1, reg2, reg3	4	1	1	4	4	4	4	No
CEILF.SL	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.SUL	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.SUW	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.SW	reg2, reg3	4	1	1	4	4	4	4	No
CMOVF.S	cc, reg1, reg2, reg3	4	1	1	4	4	4	4	No
CMPF.S	cond, reg1, reg2, cc	4	1	1	4	4	4	4	No
CVTF.LS	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.SL	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.SUL	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.SUW	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.SW	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.ULS	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.UWS	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.WS	reg2, reg3	4	1	1	4	4	4	4	No
DIVF.S	reg1, reg2, reg3	4	14	14	17	17	17	17	No
FLOORF.SL	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.SUL	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.SUW	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.SW	reg2, reg3	4	1	1	4	4	4	4	No
MADDF.S	reg1, reg2, reg3, reg4	4	2	2	5	5	5	5	No
MAXF.S	reg1, reg2, reg3	4	1	1	4	4	4	4	No
MINF.S	reg1, reg2, reg3	4	1	1	4	4	4	4	No
MSUBF.S	reg1, reg2, reg3, reg4	4	2	2	5	5	5	5	No
MULF.S	reg1, reg2, reg3	4	1	1	4	4	4	4	No
NEGF.S	reg2, reg3	4	1	1	4	4	4	4	No
NMADDF.S	reg1, reg2, reg3, reg4	4	2	2	5	5	5	5	No
NMSUBF.S	reg1, reg2, reg3, reg4	4	2	2	5	5	5	5	No
RECIPF.S	reg2, reg3	4	10	10	13	13	13	13	No
RSQRTF.S	reg2, reg3	4	13	13	16	16	16	16	No
SQRTF.S	reg2, reg3	4	14	14	17	17	17	17	No
SUBF.S	reg1, reg2, reg3	4	1	1	4	4	4	4	No

Table C-2. Clock Requirements for Single-precision Floating-point Instructions (2/2)

Mnemonic	Operand	Byte	No. of Execution Clocks						Parallel Execution[Note]
			Imprecise			Precise			
			issue	repeat	latency	issue	repeat	latency	
TRFSR	cc	4	1	1	1	1	1	1	Yes
TRNCF.SL	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.SUL	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.SUW	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.SW	reg2, reg3	4	1	1	4	4	4	4	No

Table C-3. Clock Requirements for Double-precision Floating-point Instructions (1/2)

Mnemonic	Operand	Byte	No. of Execution Clocks						Parallel Execution[Note]
			Imprecise			Precise			
			issue	repeat	latency	issue	repeat	latency	
ABSF.D	reg2, reg3	4	1	1	4	4	4	4	No
ADDF.D	reg1, reg2, reg3	4	1	1	4	4	4	4	No
CEILF.DL	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.DUL	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.DUW	reg2, reg3	4	1	1	4	4	4	4	No
CEILF.DW	reg2, reg3	4	1	1	4	4	4	4	No
CMOVF.D	cc, reg1, reg2, reg3	4	1	1	4	4	4	4	No
CMPF.D	cond, reg1, reg2, cc	4	1	1	4	4	4	4	No
CVTF.DL	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.DS	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.DUL	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.DUW	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.DW	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.LD	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.SD	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.ULD	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.UWD	reg2, reg3	4	1	1	4	4	4	4	No
CVTF.WD	reg2, reg3	4	1	1	4	4	4	4	No
DIVF.D	reg1, reg2, reg3	4	29	29	32	32	32	32	No
FLOORF.DL	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.DUL	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.DUW	reg2, reg3	4	1	1	4	4	4	4	No
FLOORF.DW	reg2, reg3	4	1	1	4	4	4	4	No
MAXF.D	reg1, reg2, reg3	4	1	1	4	4	4	4	No
MINF.D	reg1, reg2, reg3	4	1	1	4	4	4	4	No
MULF.D	reg1, reg2, reg3	4	2	2	5	5	5	5	No
NEGF.D	reg2, reg3	4	1	1	4	4	4	4	No
RECIPF.D	reg2, reg3	4	22	22	25	25	25	25	No

Table C-3. Clock Requirements for Double-precision Floating-point Instructions (2/2)

Mnemonic	Operand	Byte	No. of Execution Clocks						Parallel Execution[Note]
			Imprecise			Precise			
			issue	repeat	latency	issue	repeat	latency	
RSQRTF.D	reg2, reg3	4	30	30	33	33	33	33	No
SQRTF.D	reg2, reg3	4	29	29	32	32	32	32	No
SUBF.D	reg1, reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.DL	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.DUL	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.DUW	reg2, reg3	4	1	1	4	4	4	4	No
TRNCF.DW	reg2, reg3	4	1	1	4	4	4	4	No

Note "Yes" indicates that the instruction can be issued in parallel with other instructions, "No" indicates that the instruction cannot be issued in parallel with other instructions (the instruction must be issued individually). The pipeline used for parallel issuance (L: Lpipe, R: Rpipe, R/L: Rpipe or Lpipe) is shown in parentheses. Instructions that use the same pipeline cannot be issued in parallel.

Remarks 1. Table C-2 and Table C-3 can be updated.

2. Convention of execution clocks

Symbol	Description
issue	When next instruction is executed immediately after previous instruction
repeat	When same instruction is executed again immediately after its first execution
latency	When execution result of the current instruction is used by the immediate next instruction

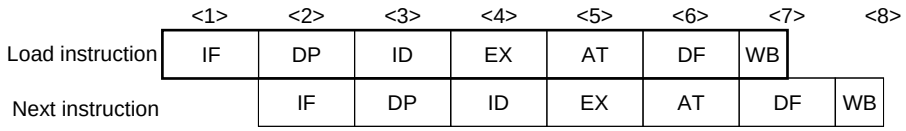
C.3 Pipeline for Basic Instructions

C.3.1 Load instructions

Load instructions are executed by the instruction execution pipeline left (Lpipe) MEM unit.

[Target instructions] LD.B, LD.H, LD.W, LD.BU, LD.HU, SLD.B, SLD.BU, SLD.H, SLD.HU, and SLD.W

[Pipeline]



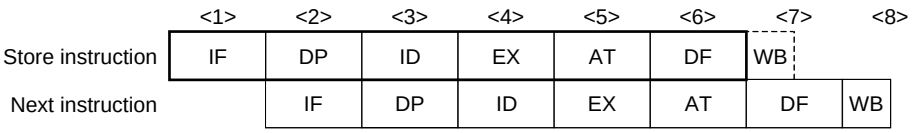
[Description] The pipeline has seven stages: the IF, DP, ID, EX, AT, DF, and WB stages. This figure shows an operation where a load instruction is executed by Lpipe, and the next instruction is issued to Lpipe. The instruction execution pipeline right (Rpipe) executes processing independently when it has no dependency on the load instruction. However, if an instruction that uses the execution result is placed immediately after the multiply instruction, a data wait period may be generated.
Load instruction can be executed in parallel with another instruction.

C.3.2 Store instructions

Store instructions are executed by the instruction execution pipeline left (Lpipe) MEM unit.

[Target instructions] ST.B, ST.H, ST.W, SST.B, SST.H, and SST.W

[Pipeline]



[Description] This pipeline has seven stages: the IF, DP, ID, EX, AT, DF, and WB stages. However, since data cannot be written to registers, nothing is done at the WB stage. This figure shows an operation where a store instruction is executed by Lpipe, and the next instruction is issued to Lpipe. The instruction execution pipeline right (Rpipe) executes processing independently when it has no dependency on the store instruction. Store instruction can be executed in parallel with another instruction.

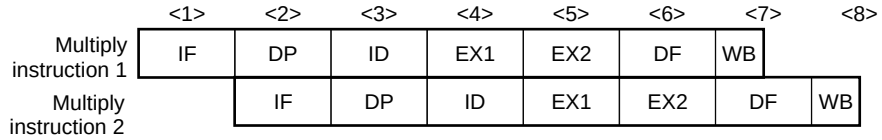
C.3.3 Multiply instructions

Multiply instructions are executed by the instruction execution pipeline left (Lpipe) MUL unit.

[Target instructions] MUL, MULH, MULHI, and MULU

[Pipeline] (a) When the next instruction is not a multiply instruction (or a multiply-accumulate instruction)

(b) When the next instruction is a multiply instruction (or a multiply-accumulate instruction)



[Description] The pipeline has seven stages: the IF, DP, ID, EX, AT, DF, and WB stages. The EX stage requires two clocks, and EX1 and EX2 operate independently. Accordingly, even if the multiply instruction (or the multiply-accumulate instruction) is repeated, the number of execution clocks becomes one.

This figure shows an operation where a multiply instruction is executed by Lpipe, and the next instruction is issued to Lpipe. The instruction execution pipeline right (Rpipe) executes processing independently when it has no dependency on the multiply instruction. However, if an instruction that uses the execution result is placed immediately after the multiply instruction, a data wait period may be generated.

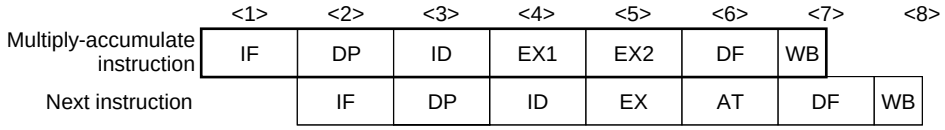
Multiply instruction can be executed in parallel with another instruction.

C.3.4 Multiply-accumulate instructions

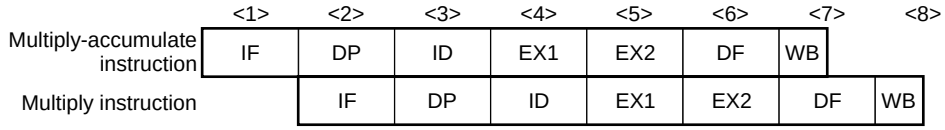
Multiply-accumulate instructions are executed by the instruction execution pipeline left (Lpipe) MUL unit.

[Target instructions] MAC and MACU

[Pipeline] (a) When the next instruction is not a multiply instruction (or a multiply-accumulate instruction)



(b) When the next instruction is a multiply instruction (or a multiply-accumulate instruction)



[Description] The pipeline has seven stages: the IF, DP, ID, EX, AT, DF, and WB stages. The EX stage requires two clocks, and EX1 and EX2 operate independently. Accordingly, even if the multiply instruction (or the multiply-accumulate instruction) is repeated, the number of execution clocks becomes one.

This figure shows an operation where a multiply instruction is executed by Lpipe, and the next instruction is issued to Lpipe. The instruction execution pipeline right (Rpipe) executes processing independently when it has no dependency on the multiply instruction. However, if an instruction that uses the execution result is placed immediately after the multiply instruction, a data wait period may be generated.

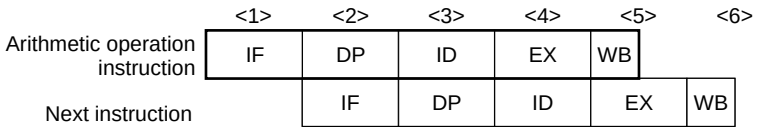
Multiply-accumulate instruction is issued individually.

C.3.5 Arithmetic operation instructions

Arithmetic operation instructions are executed by the instruction execution pipeline left or right (Lpipe or Rpipe) ALU unit.

[Target instructions] ADD, ADDI, CMP, MOV, MOVEA, MOVHI, SUB, and SUBR

[Pipeline]



[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.

This figure shows an operation where an arithmetic operation instruction is executed by Rpipe, and the next instruction is issued to Rpipe. Lpipe executes processing independently when it has no dependency on the arithmetic operation instruction.

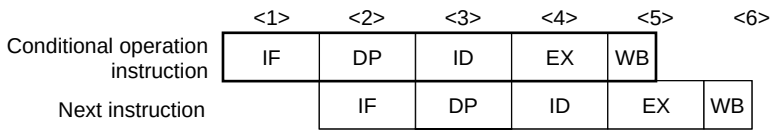
All arithmetic operation instructions except the MOV imm32 and reg1 instructions can be executed in parallel with another instruction (the MOV imm32 and reg1 instruction are issued individually).

C.3.6 Conditional operation instructions

Conditional operation instructions are executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Target instructions] ADF and SBF

[Pipeline]



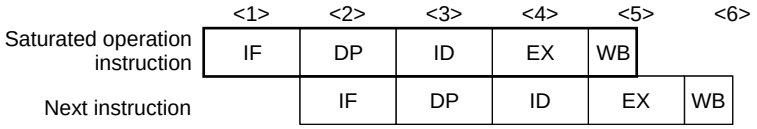
[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where an arithmetic operation instruction is executed by Rpipe, and the next instruction is issued to Rpipe.
Conditional operation instruction is issued individually.

C.3.7 Saturated operation instructions

Saturated operation instructions are executed by the instruction execution pipeline left or right (Lpipe or Rpipe) ALU unit.

[Target instructions] SATADD, SATSUB, SATSUBI, and SATSUBR

[Pipeline]



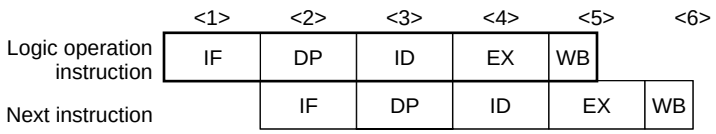
[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where a saturated operation instruction is executed by Rpipe, and the next instruction is issued to Rpipe. Lpipe executes processing independently when it has no dependency on the saturated operation instruction.
Saturated operation instructions can be executed in parallel with another instruction.

C.3.8 Logic operation instructions

Logic operation instructions are executed by the instruction execution pipeline left or right (Lpipe or Rpipe) ALU unit.

[Target instructions] AND, ANDI, NOT, OR, ORI, TST, XOR, and XORI

[Pipeline]



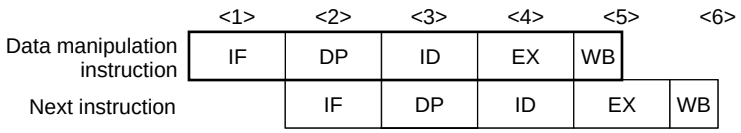
[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where a logic operations instruction is executed by Rpipe, and the next instruction is issued to Rpipe. Lpipe executes processing independently when it has no dependency on the logic operation instruction.
Logic operation instructions can be executed in parallel with another instruction.

C.3.9 Data manipulation instructions

Data manipulation instructions are executed by the instruction execution pipeline right (Rpipe) BSFT unit.

[Target instructions] BSH, BSW, CMOV, HSH, HSW, SAR, SASF, SETF, SHL, SHR, SXB, SXH, ZXB, and ZXH

[Pipeline]



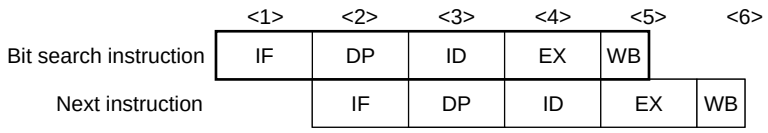
[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where a data manipulation instruction is executed by Rpipe, and the next instruction is issued to Rpipe. The instruction execution pipeline left (Lpipe) executes processing independently when it has no dependency on the data manipulation instruction.
The data manipulation instructions SXB, SXH, ZXB and ZXH are issued individually.
All other data manipulation instructions can be executed in parallel with another instruction.

C.3.10 Bit search instructions

Bit search instructions are executed by the instruction execution pipeline right (Rpipe) BSFT unit.

[Target instructions] SCH0L, SCH0R, SCH1L, and SCH1R

[Pipeline]



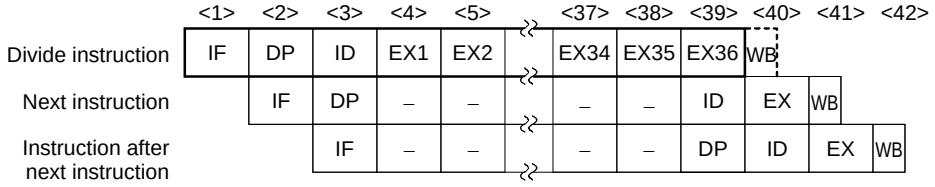
[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where a data manipulation instruction is executed by Rpipe, and the next instruction is issued to Rpipe. The instruction execution pipeline left (Lpipe) executes processing independently when it has no dependency on the data manipulation instruction.
Bit search instructions can be executed in parallel with another instruction.

C.3.11 Divide instructions

Divide instructions are executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Target instructions] DIV, DIVH, DIVHU, and DIVU

[Pipeline] (a) DIV and DIVH



–: Idle inserted for wait period

(b) DIVU and DIVHU

–: Idle inserted for wait period

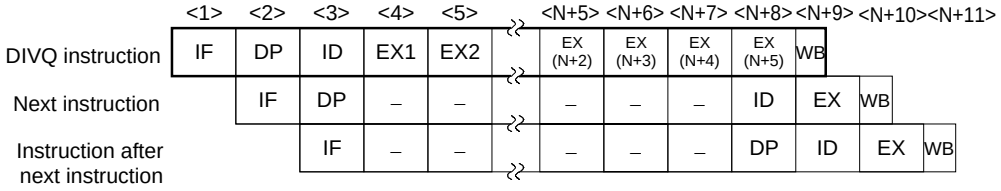
[Description] For a DIV or DIVH instruction, the pipeline has 40 stages: IF, DP, ID, EX1 to EX36, and WB, and for the DIVU and DIVHU instructions, it has 39 stages: IF, DP, ID, EX1 to EX35, and WB.
This figure shows an operation where a divide instruction is executed by Rpipe, and the next instruction is issued to Rpipe.
However, for the divide instruction, during instruction decoding at the ID stage and during instruction execution at the EX stage, the dispatch unit does not issue instructions.
Divide instruction is issued individually.

C.3.12 High-speed divide instructions

High-speed divide instructions are executed by the instruction execution pipeline right (Rpipe) ALU unit.
This instruction automatically determines the minimum number of steps required for the operation.

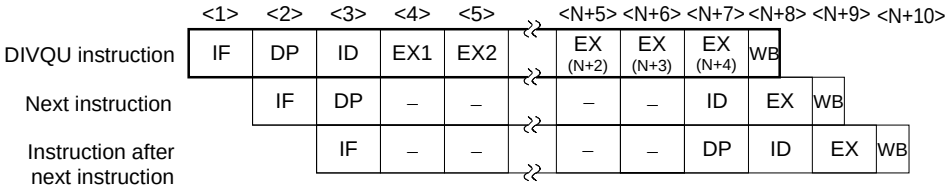
[Target instructions] DIVQ and DIVQU

[Pipeline] (a) DIVQ



–: Idle inserted for wait period

(b) DIVQU



–: Idle inserted for wait period

[Description] For the DIVQ instruction, the pipeline has N + 9 stages: IF, DP, ID, EX1 to EX(N + 5), and WB, and for the DIVQU instruction, it has N +8 stages: IF, DP, ID, EX1 to EX(N+4), and WB.
This figure shows the operation where a high-speed divide instruction is executed by Rpipe, and the next instruction is issued to Rpipe.
During instruction decoding at the ID stage and during instruction execution at the EX stage, the dispatch unit does not issue instructions.
Each instruction is issued individually.

Remark N = (Number of valid bits of dividend) – (Number of valid bits of divisor)
However, if N is negative, it is assumed that N = 0.

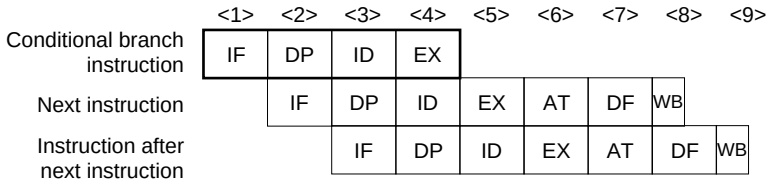
C.3.13 Branch instructions

Branch instructions are executed by the instruction execution pipeline left or right (Lpipe or Rpipe) ALU unit.

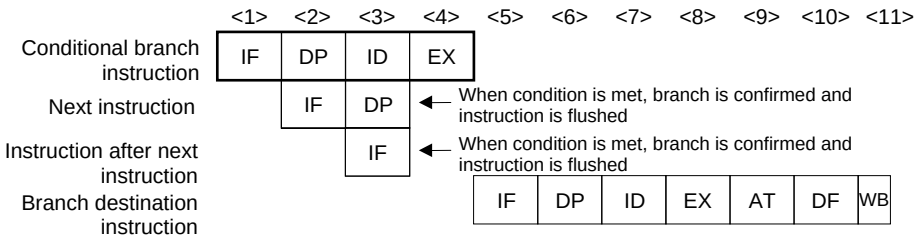
(1) Conditional branch instruction (except BR instruction)

[Target instructions] Bcond instruction

[Pipeline] (a) When condition has not been met



(b) When condition has been met

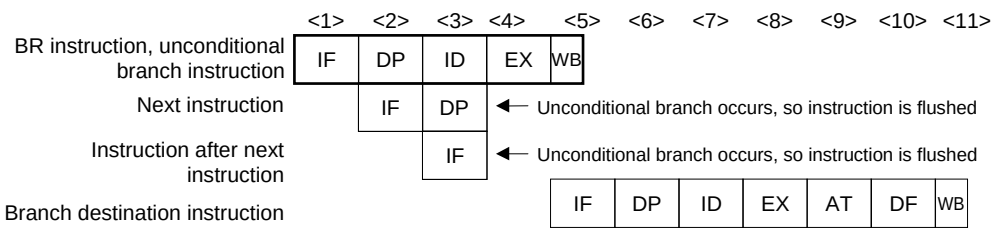


[Description] This figure shows an operation where a Bcond instruction is executed by the instruction execution pipeline right (Rpipe).
Branch instructions can be executed in parallel with another instruction.

(2) BR instruction and unconditional branch instructions (except JMP instruction)

[Target instructions] BR, JARL, and JR instruction

[Pipeline]

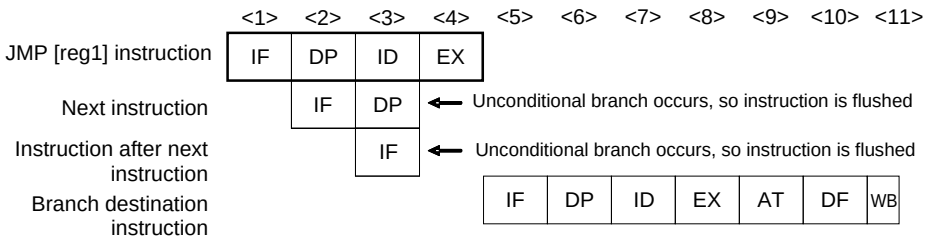


[Description] This figure shows an operation where the Bcond instruction is executed by the instruction execution pipeline right (Rpipe), and all instructions are executed by the instruction execution pipeline left (Lpipe).
BR and unconditional branch instructions can be executed in parallel with another instruction.

(3) JMP instructions

(a) JMP [reg1] instruction

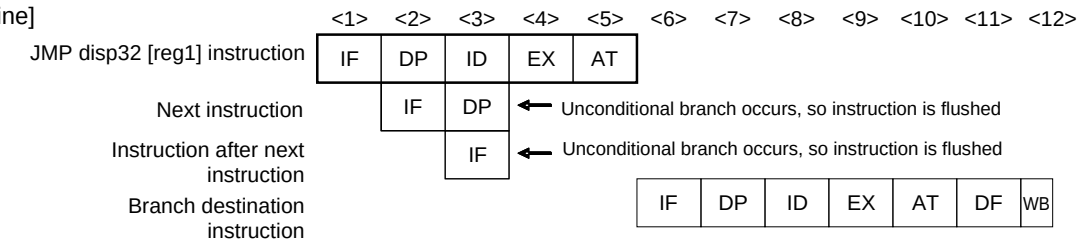
[Pipeline]



[Description] This figure shows an operation where a JMP is executed by the instruction execution pipeline right (Rpipe), and all instructions are executed by instruction execution pipeline left (Lpipe).
This instruction can be executed in parallel with other instructions.

(b) JMP dip32 [reg1] instruction

[Pipeline]



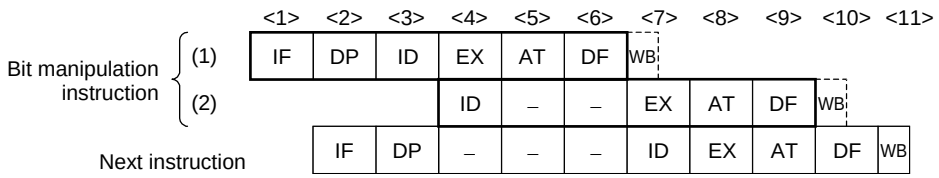
[Description] This figure shows an operation where a JMP is executed by the instruction execution pipeline right (Rpipe) and all instructions are executed by instruction execution pipeline left (Lpipe).
This instruction can be executed in parallel with other instructions.

C.3.14 Bit manipulation instructions

Bit manipulation instructions are executed by the instruction execution pipeline left (Lpipe) ALU unit.

(1) CLR1, NOT1, and SET1 instructions

[Pipeline]



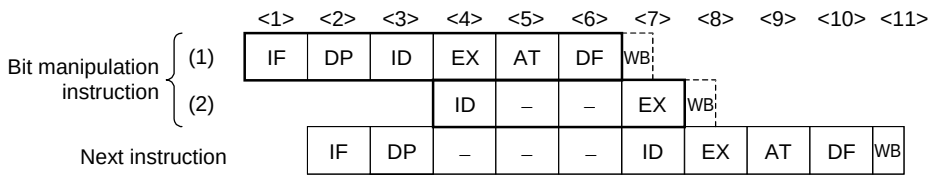
–: Idle inserted for wait period

[Description] At the ID stage, the instruction is divided into two instructions, then the load instruction is executed, followed by the store instruction that includes bit manipulation. However, since no data is written to a register, nothing is done at the WB stage. This figure shows an operation where a bit manipulation instruction is executed by Lpipe and the next instruction is issued to Lpipe. During instruction decoding at the ID stage, the dispatch unit issues instruction to Lpipe.

Bit manipulation instruction is issued individually.

(2) TST1 instruction

[Pipeline]



–: Idle inserted for wait period

[Description] At the ID stage, the instruction is divided into two instructions, then the load instruction is executed, following by the store instruction that includes bit manipulation. However, since no data is written to a register, nothing is done at the WB stage. This figure shows an operation where a TST1 instruction is executed by Lpipe and the next instruction is issued to Lpipe. During instruction decoding at the ID stage, the dispatch unit does not issue an instruction to Lpipe.

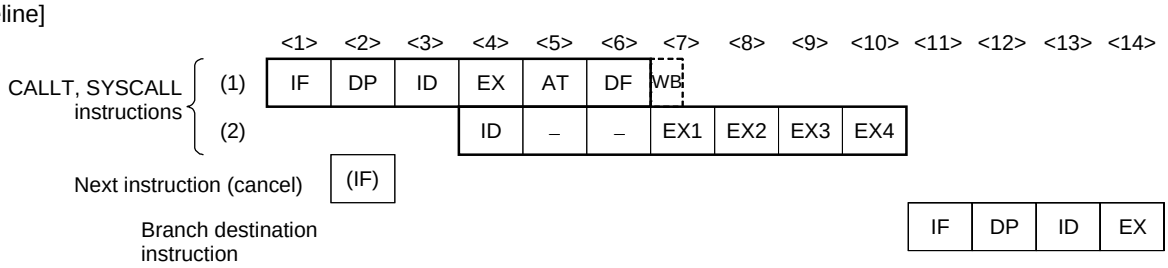
TST1 instruction is issued individually.

C.3.15 Special instructions

(1) CALLT and SYSCALL instruction

The CALLT and SYSCALL instructions are executed by the instruction execution pipeline left (Lpipe) ALU unit.

[Pipeline]

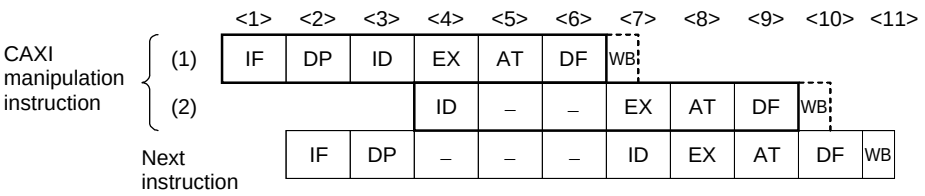


–: Idle inserted for wait period
(IF): Invalid instruction fetch

[Description] At the ID stage, the instruction is divided into two instructions, then the load instruction is executed, followed by execution of the CTBP or CTBP relative branch instruction. However, since no data is written to a register, nothing is done at the WB stage. This figure shows an operation where the CALLT or SYSCALL instruction is executed by the Lpipe, and then the instruction is fetched from the branch destination.
This instruction is issued individually.

(2) CAXI instruction

[Pipeline]



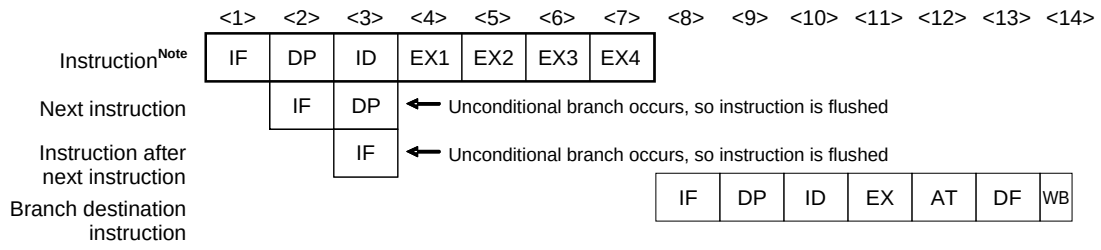
–: Idle inserted for wait period

[Description] At the ID stage, the instruction is divided into two instructions, then the load instruction is executed, followed by the store instruction. This figure shows an operation where a CAXI instruction is executed by Lpipe, and the next instruction is issued to Lpipe. During instruction decoding at the ID stage, the dispatch unit does not issue instructions to Lpipe. Each instruction is issued individually.

(3) CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions

The CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions are executed by the instruction execution pipeline right (Rpipe) .

[Pipeline]



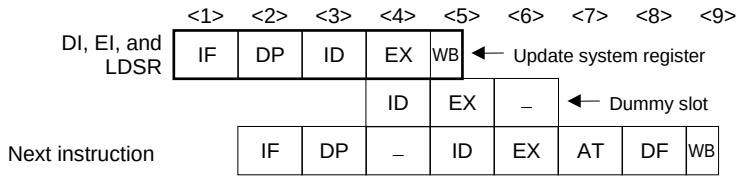
Note CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions

[Description] This figure shows an operation where each instruction is executed by the instruction execution pipeline right (Rpipe) and all instructions are issued by Lpipe.
CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions are issued individually.

(4) DI, EI, and LDSR instructions

The DI, EI, and LDSR instructions are executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Pipeline]



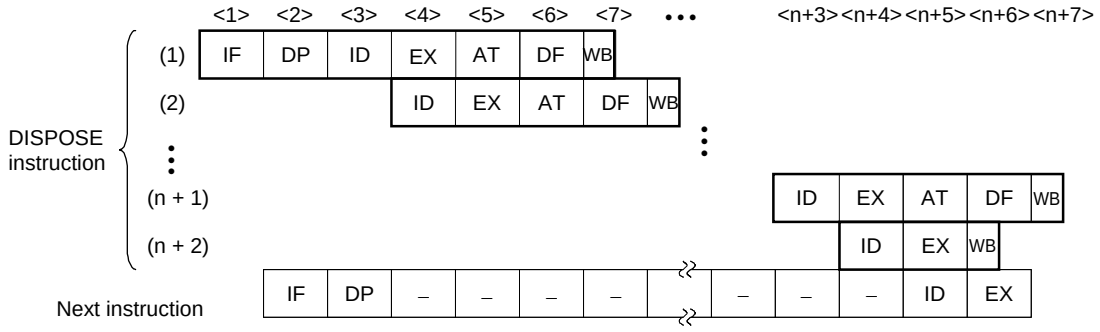
–: Idle inserted for wait period

[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where the DI, EI, and LDSR instructions are executed by Rpipe and all instructions are issued by Rpipe.
DI, EI, and LDSR instructions are issued individually.

(5) DISPOSE instruction

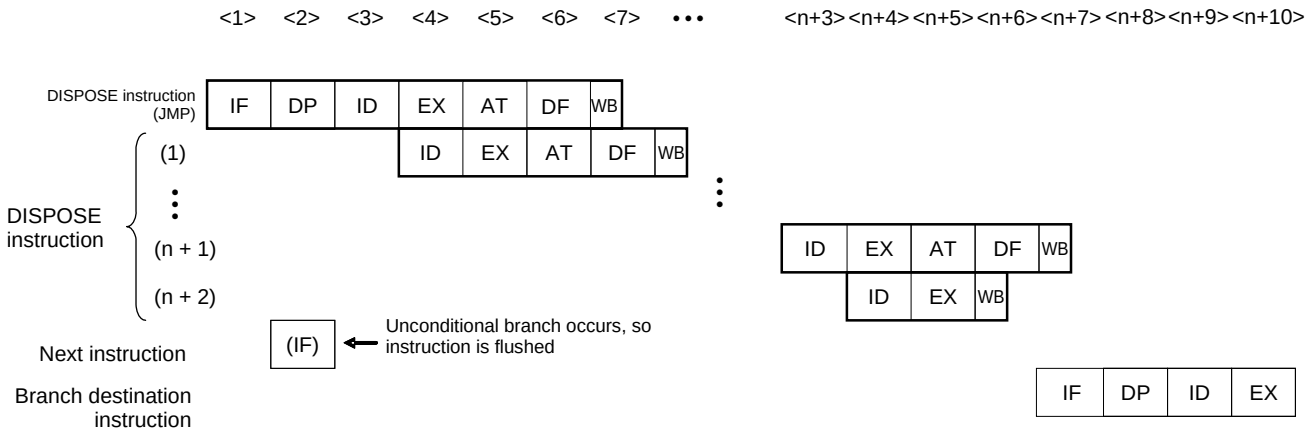
The DISPOSE instruction is executed by the instruction execution pipeline left (Lpipe) ALU unit.

[Pipeline] (a) No branch



–: Idle inserted for wait period

(b) Branch



–: Idle inserted for wait period
(IF): Invalid instruction fetch

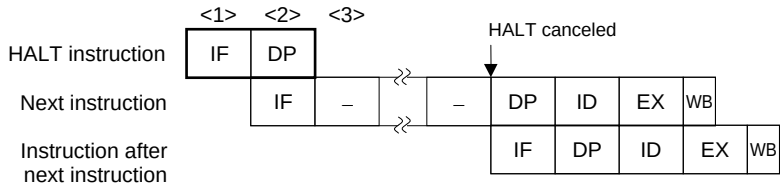
Remark n is the number of registers specified by the register list (list12).

[Description] At the ID stage, this instruction is divided into n + 2 instructions, then n load instructions are executed, followed by an instruction that writes to the stack pointer (SP). This figure shows an operation where the DISPOSE instruction is executed by the Lpipe and the next instruction is issued to the Lpipe. The instruction execution pipeline right (Rpipe) executes processing independently when it has no dependency on the DISPOSE instruction. This instruction is issued individually.

(6) HALT instruction

The HALT instruction is executed by the instruction execution pipeline right (Rpipe).

[Pipeline]



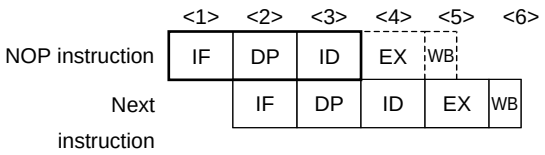
–: Idle inserted for wait period

[Description] When a HALT instruction is detected at the DP stage, issuing of instructions to the ID stage is stopped until the HALT instruction has been canceled. Accordingly, the next instruction is delayed at the ID stage until the HALT instruction is canceled. This figure shows an operation where the HALT instruction is executed by the instruction execution pipeline right (Rpipe), and the next instruction is issued to Rpipe.
This instruction is issued individually.

(7) NOP instruction

The NOP instruction is executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Pipeline]

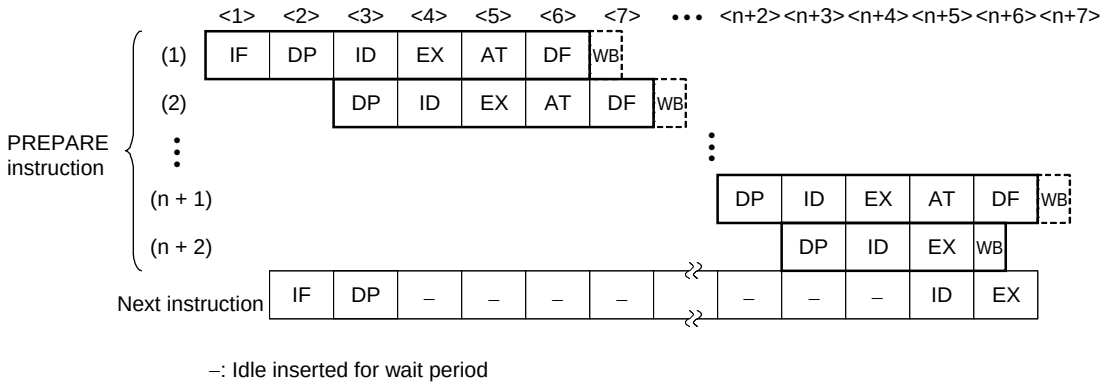


[Description] The pipeline has five stages: IF, DP, ID, EX, and WB, but since no operations are performed and there is no writing of data to registers, nothing is done at the EX and WB stages.
This instruction is issued individually.

(8) PREPARE instruction

The PREPARE instruction is executed by the instruction execution pipeline left (Lpipe) ALU unit.

[Pipeline]



–: Idle inserted for wait period

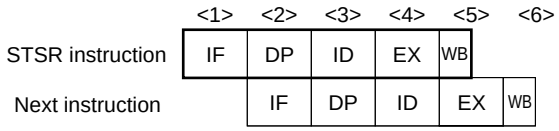
Remark n is the number of registers specified in the register list (list12).

[Description] At the ID stage, this instruction is divided into n + 2 instructions, then n store instructions are executed, followed by an instruction that writes to the stack pointer (SP). Since this store instruction does not write data to a register, nothing is done at the WB stage. This figure shows an operation where the PREPARE instruction is executed by Lpipe and the next instruction is issued to Lpipe.
This instruction is issued individually.

(9) STSR instruction

The STSR instruction is executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Pipeline]

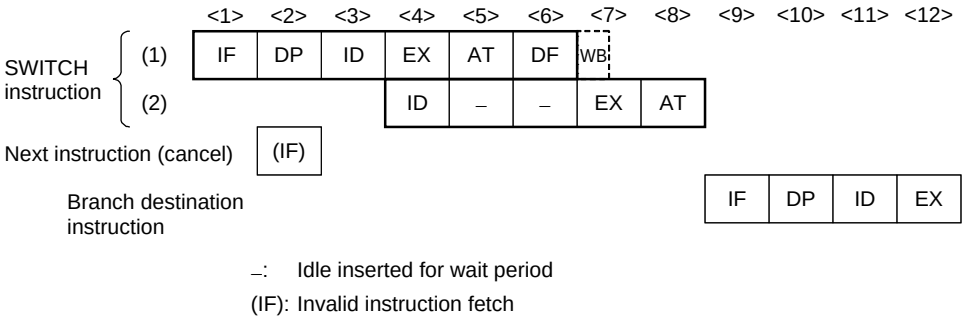


[Description] The pipeline has five stages: the IF, DP, ID, EX, and WB stages.
This figure shows an operation where the STSR instruction is executed by Rpipe and the next instruction is issued to Rpipe. The instruction execution pipeline left (Lpipe) executes processing independently when it has no dependency on the STSR instruction.
This instruction is issued individually.

(10) SWITCH instruction

The SWITCH instruction is executed by the instruction execution pipeline left (Lpipe) ALU unit.

[Pipeline]



–: Idle inserted for wait period
(IF): Invalid instruction fetch

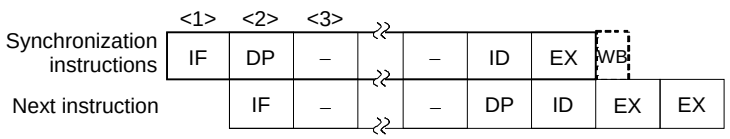
[Description] At the ID stage, the instruction is divided into two instructions, then the load instruction is executed, followed by execution of the PC relative branch instruction. However, since no data is written to a register, nothing is done at the WB stage. This figure shows an operation where a SWITCH instruction is executed by Lpipe, and the next instruction is issued to Lpipe. The instruction execution pipeline right (Rpipe) performs processing independently when it has no dependency on the SWITCH instruction.
This instruction is issued individually.

(11) Synchronization instructions

The synchronization instructions are executed by the instruction execution pipeline right (Rpipe) ALU unit.

[Target instructions] SYNCE, SYNCM, SYNCP

[Pipeline]



[Description] This figure shows an operation where the synchronization instructions are executed by Rpipe and all instructions are issued by the instruction execution pipeline left (Lpipe). Each instruction is issued individually.
The synchronization instructions are not issued until processing of all instructions held pending by the CPU has been completed.

APPENDIX D LIST OF PERIPHERAL DEVICE PROTECTION AREAS

Each bit of the peripheral device protection setup registers (PPC0 to PPC8) for the V850E2M CPU corresponds to a peripheral device.

For the V850E2M CPU, the bits PPx2, PPx4 to PPx6, PPx12 to PPx15, PPx18, and PPx19, which correspond to the areas in the PPC0 register set enclosed in parentheses, are fixed to 0. In addition, PPx16 and the reserved area PPx17, where the setup registers for system protection (peripheral device protection and timing monitoring) are placed, are fixed to 1 as OS peripheral devices.

PPC0 (PPS0, PPP0, PPV0, PPT0)

Bit Name	Area Name	Address Range	Remarks
PPx0	INTC	FFFF6000 to FFFF645FH	INTC
PPx1	Fcache	FFFF6480 to FFFF6487H	FlashCache
PPx2	(R.F.U.)	R.F.U	Reserved
PPx3	SEG	FFFF64B0 to FFFF64B3H	System error
PPx4	(R.F.U.)	R.F.U	Reserved
PPx5	(R.F.U.)	FFFF6500 to FFFF65FFH	Reserved
PPx6	(R.F.U.)	FFFF6600 to FFFF66FFH	Reserved
PPx7	EXTBRK	FFFF6700 to FFFF67FFH	Debbug function
PPx8	MIR	FFFF6800 to FFFF687FH	Inter-PE interrupts
PPx9	MEV	FFFF6900 to FFFF697FH	MEV
PPx10	MEC	FFFF6980 to FFFF699FH	MEC
PPx11	PEG	FFFF69A0 to FFFF69BFH	PE guard
PPx12	(SPB12)	FFFF6A00 to FFFF6AFFH	Reserved
PPx13	(SPB13)	FFFF6B00 to FFFF6BFFH	Reserved
PPx14	(SPB14)	FFFF6C00 to FFFF6DFFH	Reserved
PPx15	(SPB15)	FFFF6E00 to FFFF6EFFH	Reserved
PPx16	SPF	FFFF5000 to FFFF53FFH	TSU/PPU
PPx17	(R.F.U.)	FFFF5400 to FFFF57FFH	Reserved
PPx18	(R.F.U.)	FFFF5800 to FFFF5BFFH	Reserved
PPx19	(R.F.U.)	FFFF5C00 to FFFF5FFFH	Reserved
PPx20	EXT20	FFFF7000 to FFFF70FFH	Area for expansion
PPx21	EXT21	FFFF7100 to FFFF71FFH	Area for expansion
PPx22	EXT22	FFFF7200 to FFFF72FFH	Area for expansion
PPx23	EXT23	FFFF7300 to FFFF73FFH	Area for expansion
PPx24	EXT24	FFFF7400 to FFFF74FFH	Area for expansion
PPx25	EXT25	FFFF7500 to FFFF76FFH	Area for expansion
PPx26	EXT26	FFFF7700 to FFFF78FFH	Area for expansion
PPx27	EXT27	FFFF7900 to FFFF7AFFH	Area for expansion
PPx28	EXT28	FFFF7B00 to FFFF7CFFH	Area for expansion
PPx29	EXT29	FFFF7D00 to FFFF7EFFH	Area for expansion
PPx30	EXT30	FFFF7F00 to FFFF7F7FH	Area for expansion
PPx31	EXT31	FFFF7F80 to FFFF7FFFH	Area for expansion

PPC1 (PPS1, PPP1, PPV1, PPT1)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FF400000 to FF40FFFFH	64 KB	PPx16	FF500000 to FF50FFFFH	64 KB
PPx1	FF410000 to FF41FFFFH	64 KB	PPx17	FF510000 to FF51FFFFH	64 KB
PPx2	FF420000 to FF42FFFFH	64 KB	PPx18	FF520000 to FF52FFFFH	64 KB
PPx3	FF430000 to FF43FFFFH	64 KB	PPx19	FF530000 to FF53FFFFH	64 KB
PPx4	FF440000 to FF44FFFFH	64 KB	PPx20	FF540000 to FF54FFFFH	64 KB
PPx5	FF450000 to FF45FFFFH	64 KB	PPx21	FF550000 to FF55FFFFH	64 KB
PPx6	FF460000 to FF46FFFFH	64 KB	PPx22	FF560000 to FF56FFFFH	64 KB
PPx7	FF470000 to FF47FFFFH	64 KB	PPx23	FF570000 to FF57FFFFH	64 KB
PPx8	FF480000 to FF48FFFFH	64 KB	PPx24	FF580000 to FF58FFFFH	64 KB
PPx9	FF490000 to FF49FFFFH	64 KB	PPx25	FF590000 to FF59FFFFH	64 KB
PPx10	FF4A0000 to FF4AFFFFH	64 KB	PPx26	FF5A0000 to FF5AFFFFH	64 KB
PPx11	FF4B0000 to FF4BFFFFH	64 KB	PPx27	FF5B0000 to FF5BFFFFH	64 KB
PPx12	FF4C0000 to FF4CFFFFH	64 KB	PPx28	FF5C0000 to FF5CFFFFH	64 KB
PPx13	FF4D0000 to FF4DFFFFH	64 KB	PPx29	FF5D0000 to FF5DFFFFH	64 KB
PPx14	FF4E0000 to FF4EFFFFH	64 KB	PPx30	FF5E0000 to FF5EFFFFH	64 KB
PPx15	FF4F0000 to FF4FFFFFH	64 KB	PPx31	FF5F0000 to FF5FFFFFH	64 KB

PPC2 (PPS2, PPP2, PPV2, PPT2)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FF600000 to FF60FFFFH	64 KB	PPx16	FF700000 to FF70FFFFH	64 KB
PPx1	FF610000 to FF61FFFFH	64 KB	PPx17	FF710000 to FF71FFFFH	64 KB
PPx2	FF620000 to FF62FFFFH	64 KB	PPx18	FF720000 to FF72FFFFH	64 KB
PPx3	FF630000 to FF63FFFFH	64 KB	PPx19	FF730000 to FF73FFFFH	64 KB
PPx4	FF640000 to FF64FFFFH	64 KB	PPx20	FF740000 to FF74FFFFH	64 KB
PPx5	FF650000 to FF65FFFFH	64 KB	PPx21	FF750000 to FF75FFFFH	64 KB
PPx6	FF660000 to FF66FFFFH	64 KB	PPx22	FF760000 to FF76FFFFH	64 KB
PPx7	FF670000 to FF67FFFFH	64 KB	PPx23	FF770000 to FF77FFFFH	64 KB
PPx8	FF680000 to FF68FFFFH	64 KB	PPx24	FF780000 to FF78FFFFH	64 KB
PPx9	FF690000 to FF69FFFFH	64 KB	PPx25	FF790000 to FF79FFFFH	64 KB
PPx10	FF6A0000 to FF6AFFFFH	64 KB	PPx26	FF7A0000 to FF7AFFFFH	64 KB
PPx11	FF6B0000 to FF6BFFFFH	64 KB	PPx27	FF7B0000 to FF7BFFFFH	64 KB
PPx12	FF6C0000 to FF6CFFFFH	64 KB	PPx28	FF7C0000 to FF7CFFFFH	64 KB
PPx13	FF6D0000 to FF6DFFFFH	64 KB	PPx29	FF7D0000 to FF7DFFFFH	64 KB
PPx14	FF6E0000 to FF6EFFFFH	64 KB	PPx30	FF7E0000 to FF7EFFFFH	64 KB
PPx15	FF6F0000 to FF6FFFFFH	64 KB	PPx31	FF7F0000 to FF7FFFFFH	64 KB

PPC3 (PPS3, PPP3, PPV3, PPT3)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FF800000 to FF800FFFH	4 KB	PPx16	FF810000 to FF810FFFH	4 KB
PPx1	FF801000 to FF801FFFH	4 KB	PPx17	FF811000 to FF811FFFH	4 KB
PPx2	FF802000 to FF802FFFH	4 KB	PPx18	FF812000 to FF812FFFH	4 KB
PPx3	FF803000 to FF803FFFH	4 KB	PPx19	FF813000 to FF813FFFH	4 KB
PPx4	FF804000 to FF804FFFH	4 KB	PPx20	FF814000 to FF814FFFH	4 KB
PPx5	FF805000 to FF805FFFH	4 KB	PPx21	FF815000 to FF815FFFH	4 KB
PPx6	FF806000 to FF806FFFH	4 KB	PPx22	FF816000 to FF816FFFH	4 KB
PPx7	FF807000 to FF807FFFH	4 KB	PPx23	FF817000 to FF817FFFH	4 KB
PPx8	FF808000 to FF808FFFH	4 KB	PPx24	FF818000 to FF818FFFH	4 KB
PPx9	FF809000 to FF809FFFH	4 KB	PPx25	FF819000 to FF819FFFH	4 KB
PPx10	FF80A000 to FF80AFFFH	4 KB	PPx26	FF81A000 to FF81AFFFH	4 KB
PPx11	FF80B000 to FF80BFFFH	4 KB	PPx27	FF81B000 to FF81BFFFH	4 KB
PPx12	FF80C000 to FF80CFFFH	4 KB	PPx28	FF81C000 to FF81CFFFH	4 KB
PPx13	FF80D000 to FF80DFFFH	4 KB	PPx29	FF81D000 to FF81DFFFH	4 KB
PPx14	FF80E000 to FF80EFFFH	4 KB	PPx30	FF81E000 to FF81EFFFH	4 KB
PPx15	FF80F000 to FF80FFFFH	4 KB	PPx31	FF81F000 to FF81FFFFH	4 KB

PPC4 (PPS4, PPP4, PPV4, PPT4)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FF820000 to FF820FFFH	4 KB	PPx16	FF830000 to FF830FFFH	4 KB
PPx1	FF821000 to FF821FFFH	4 KB	PPx17	FF831000 to FF831FFFH	4 KB
PPx2	FF822000 to FF822FFFH	4 KB	PPx18	FF832000 to FF832FFFH	4 KB
PPx3	FF823000 to FF823FFFH	4 KB	PPx19	FF833000 to FF833FFFH	4 KB
PPx4	FF824000 to FF824FFFH	4 KB	PPx20	FF834000 to FF834FFFH	4 KB
PPx5	FF825000 to FF825FFFH	4 KB	PPx21	FF835000 to FF835FFFH	4 KB
PPx6	FF826000 to FF826FFFH	4 KB	PPx22	FF836000 to FF836FFFH	4 KB
PPx7	FF827000 to FF827FFFH	4 KB	PPx23	FF837000 to FF837FFFH	4 KB
PPx8	FF828000 to FF828FFFH	4 KB	PPx24	FF838000 to FF838FFFH	4 KB
PPx9	FF829000 to FF829FFFH	4 KB	PPx25	FF839000 to FF839FFFH	4 KB
PPx10	FF82A000 to FF82AFFFH	4 KB	PPx26	FF83A000 to FF83AFFFH	4 KB
PPx11	FF82B000 to FF82BFFFH	4 KB	PPx27	FF83B000 to FF83BFFFH	4 KB
PPx12	FF82C000 to FF82CFFFH	4 KB	PPx28	FF83C000 to FF83CFFFH	4 KB
PPx13	FF82D000 to FF82DFFFH	4 KB	PPx29	FF83D000 to FF83DFFFH	4 KB
PPx14	FF82E000 to FF82EFFFH	4 KB	PPx30	FF83E000 to FF83EFFFH	4 KB
PPx15	FF82F000 to FF82FFFFH	4 KB	PPx31	FF83F000 to FF83FFFFH	4 KB

PPC5 (PPS5, PPP5, PPV5, PPT5)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FFFF8000 to FFFF80FFH	256 byte	PPx16	FFFF9000 to FFFF90FFH	256 byte
PPx1	FFFF8100 to FFFF81FFH	256 byte	PPx17	FFFF9100 to FFFF91FFH	256 byte
PPx2	FFFF8200 to FFFF82FFH	256 byte	PPx18	FFFF9200 to FFFF92FFH	256 byte
PPx3	FFFF8300 to FFFF83FFH	256 byte	PPx19	FFFF9300 to FFFF93FFH	256 byte
PPx4	FFFF8400 to FFFF84FFH	256 byte	PPx20	FFFF9400 to FFFF94FFH	256 byte
PPx5	FFFF8500 to FFFF85FFH	256 byte	PPx21	FFFF9500 to FFFF95FFH	256 byte
PPx6	FFFF8600 to FFFF86FFH	256 byte	PPx22	FFFF9600 to FFFF96FFH	256 byte
PPx7	FFFF8700 to FFFF87FFH	256 byte	PPx23	FFFF9700 to FFFF97FFH	256 byte
PPx8	FFFF8800 to FFFF88FFH	256 byte	PPx24	FFFF9800 to FFFF98FFH	256 byte
PPx9	FFFF8900 to FFFF89FFH	256 byte	PPx25	FFFF9900 to FFFF99FFH	256 byte
PPx10	FFFF8A00 to FFFF8AFFH	256 byte	PPx26	FFFF9A00 to FFFF9AFFH	256 byte
PPx11	FFFF8B00 to FFFF8BFFH	256 byte	PPx27	FFFF9B00 to FFFF9BFFH	256 byte
PPx12	FFFF8C00 to FFFF8CFFH	256 byte	PPx28	FFFF9C00 to FFFF9CFFH	256 byte
PPx13	FFFF8D00 to FFFF8DFFH	256 byte	PPx29	FFFF9D00 to FFFF9DFFH	256 byte
PPx14	FFFF8E00 to FFFF8EFFH	256 byte	PPx30	FFFF9E00 to FFFF9EFFH	256 byte
PPx15	FFFF8F00 to FFFF8FFFH	256 byte	PPx31	FFFF9F00 to FFFF9FFFH	256 byte

PPC6 (PPS6, PPP6, PPV6, PPT6)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FFFFA000 to FFFFA0FFH	256 byte	PPx16	FFFFB000 to FFFFB0FFH	256 byte
PPx1	FFFFA100 to FFFFA1FFH	256 byte	PPx17	FFFFB100 to FFFFB1FFH	256 byte
PPx2	FFFFA200 to FFFFA2FFH	256 byte	PPx18	FFFFB200 to FFFFB2FFH	256 byte
PPx3	FFFFA300 to FFFFA3FFH	256 byte	PPx19	FFFFB300 to FFFFB3FFH	256 byte
PPx4	FFFFA400 to FFFFA4FFH	256 byte	PPx20	FFFFB400 to FFFFB4FFH	256 byte
PPx5	FFFFA500 to FFFFA5FFH	256 byte	PPx21	FFFFB500 to FFFFB5FFH	256 byte
PPx6	FFFFA600 to FFFFA6FFH	256 byte	PPx22	FFFFB600 to FFFFB6FFH	256 byte
PPx7	FFFFA700 to FFFFA7FFH	256 byte	PPx23	FFFFB700 to FFFFB7FFH	256 byte
PPx8	FFFFA800 to FFFFA8FFH	256 byte	PPx24	FFFFB800 to FFFFB8FFH	256 byte
PPx9	FFFFA900 to FFFFA9FFH	256 byte	PPx25	FFFFB900 to FFFFB9FFH	256 byte
PPx10	FFFFAA00 to FFFFAAFFH	256 byte	PPx26	FFFFBA00 to FFFFBAFFH	256 byte
PPx11	FFFFAB00 to FFFFABFFH	256 byte	PPx27	FFFFBB00 to FFFFBBFFH	256 byte
PPx12	FFFFAC00 to FFFFACFFH	256 byte	PPx28	FFFFBC00 to FFFFBCFFH	256 byte
PPx13	FFFFAD00 to FFFFADFFH	256 byte	PPx29	FFFFBD00 to FFFFBDFFH	256 byte
PPx14	FFFFAE00 to FFFFAEFFH	256 byte	PPx30	FFFFBE00 to FFFFBEFFH	256 byte
PPx15	FFFFAF00 to FFFFAFFFH	256 byte	PPx31	FFFFBF00 to FFFFBFFFH	256 byte

PPC7 (PPS7, PPP7, PPV7, PPT7)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FFFFC000 to FFFFC0FFH	256B	PPx16	FFFFD000 to FFFFD0FFH	256B
PPx1	FFFFC100 to FFFFC1FFH	256B	PPx17	FFFFD100 to FFFFD1FFH	256B
PPx2	FFFFC200 to FFFFC2FFH	256B	PPx18	FFFFD200 to FFFFD2FFH	256B
PPx3	FFFFC300 to FFFFC3FFH	256B	PPx19	FFFFD300 to FFFFD3FFH	256B
PPx4	FFFFC400 to FFFFC4FFH	256B	PPx20	FFFFD400 to FFFFD4FFH	256B
PPx5	FFFFC500 to FFFFC5FFH	256B	PPx21	FFFFD500 to FFFFD5FFH	256B
PPx6	FFFFC600 to FFFFC6FFH	256B	PPx22	FFFFD600 to FFFFD6FFH	256B
PPx7	FFFFC700 to FFFFC7FFH	256B	PPx23	FFFFD700 to FFFFD7FFH	256B
PPx8	FFFFC800 to FFFFC8FFH	256B	PPx24	FFFFD800 to FFFFD8FFH	256B
PPx9	FFFFC900 to FFFFC9FFH	256B	PPx25	FFFFD900 to FFFFD9FFH	256B
PPx10	FFFFCA00 to FFFFCAFFH	256B	PPx26	FFFFDA00 to FFFFDAFFH	256B
PPx11	FFFFCB00 to FFFFCBFFH	256B	PPx27	FFFFDB00 to FFFFDBFFH	256B
PPx12	FFFFCC00 to FFFFCCFFH	256B	PPx28	FFFFDC00 to FFFFDCFFH	256B
PPx13	FFFFCD00 to FFFFCDFFH	256B	PPx29	FFFFDD00 to FFFFDDFFH	256B
PPx14	FFFFCE00 to FFFFCEFFH	256B	PPx30	FFFFDE00 to FFFFDEFFH	256B
PPx15	FFFFCF00 to FFFFCFFFH	256B	PPx31	FFFFDF00 to FFFFDFFFH	256B

PPC8 (PPS8, PPP8, PPV8, PPT8)

Bit Name	Address Range	Remarks	Bit Name	Address Range	Remarks
PPx0	FFFFE000 to FFFFE0FFH	256B	PPx16	FFFFF000 to FFFFF0FFH	256B
PPx1	FFFFE100 to FFFFE1FFH	256B	PPx17	FFFFF100 to FFFFF1FFH	256B
PPx2	FFFFE200 to FFFFE2FFH	256B	PPx18	FFFFF200 to FFFFF2FFH	256B
PPx3	FFFFE300 to FFFFE3FFH	256B	PPx19	FFFFF300 to FFFFF3FFH	256B
PPx4	FFFFE400 to FFFFE4FFH	256B	PPx20	FFFFF400 to FFFFF4FFH	256B
PPx5	FFFFE500 to FFFFE5FFH	256B	PPx21	FFFFF500 to FFFFF5FFH	256B
PPx6	FFFFE600 to FFFFE6FFH	256B	PPx22	FFFFF600 to FFFFF6FFH	256B
PPx7	FFFFE700 to FFFFE7FFH	256B	PPx23	FFFFF700 to FFFFF7FFH	256B
PPx8	FFFFE800 to FFFFE8FFH	256B	PPx24	FFFFF800 to FFFFF8FFH	256B
PPx9	FFFFE900 to FFFFE9FFH	256B	PPx25	FFFFF900 to FFFFF9FFH	256B
PPx10	FFFFEA00 to FFFFEAFFH	256B	PPx26	FFFFFA00 to FFFFFAFFH	256B
PPx11	FFFFEB00 to FFFFEBFFH	256B	PPx27	FFFFFB00 to FFFFFBFFH	256B
PPx12	FFFFEC00 to FFFFECFFH	256B	PPx28	FFFFFC00 to FFFFFCFFH	256B
PPx13	FFFFED00 to FFFFEDFFH	256B	PPx29	FFFFFD00 to FFFFFDFFH	256B
PPx14	FFFFEE00 to FFFFEEFFH	256B	PPx30	FFFFFE00 to FFFFFEFFH	256B
PPx15	FFFFEF00 to FFFFEFFFH	256B	PPx31	FFFFFF00 to FFFFFFFFH	256B

APPENDIX E DIFFERENCES BETWEEN V850E2M CPU AND OTHER CPUS

E.1 Difference Between V850E1 and V850E2

(1/3)

<table>
<tr><th colspan="2">Item</th><th>V850E2M</th><th>V850E2</th><th>V850E1</th></tr>
<tr><td rowspan="18">Instructions (including operands)</td><td>ADF cccc, reg1, reg2, reg3</td><td rowspan="18">Provided</td><td rowspan="18"></td><td rowspan="18">Not provided</td></tr>
<tr><td>HSH reg2, reg3</td></tr>
<tr><td>JARL disp32, reg1</td></tr>
<tr><td>JMP disp32, [reg1]</td></tr>
<tr><td>JR disp32</td></tr>
<tr><td>MAC reg1, reg2, reg3, reg4</td></tr>
<tr><td>MACU reg1, reg2, reg3, reg4</td></tr>
<tr><td>SAR reg1, reg2, reg3</td></tr>
<tr><td>SATADD reg1, reg2, reg3</td></tr>
<tr><td>SATSUB reg1, reg2, reg3</td></tr>
<tr><td>SBF cccc, reg1, reg2, reg3</td></tr>
<tr><td>SCH0L reg1, reg2</td></tr>
<tr><td>SCH0R reg1, reg2</td></tr>
<tr><td>SCH1L reg1, reg2</td></tr>
<tr><td>SCH1R reg1, reg2</td></tr>
<tr><td>SHL reg1, reg2, reg3</td></tr>
<tr><td>SHR reg1, reg2, reg3</td></tr>
<tr><td>CAXI [reg1], reg2, reg3</td><td>Provided</td><td colspan="2" rowspan="32">Not provided</td></tr>
<tr><td>DIVQ reg1, reg2, reg3</td></tr>
<tr><td>DIVQU reg1, reg2, reg3</td></tr>
<tr><td>EIRET</td></tr>
<tr><td>FERET</td></tr>
<tr><td>FETRAP vector4</td></tr>
<tr><td>RIE</td></tr>
<tr><td>SYNCM</td></tr>
<tr><td>SYNCP</td></tr>
<tr><td>SYNCE</td></tr>
<tr><td>SYSCALL vector8</td></tr>
<tr><td>LD.B disp23 [reg1] , reg3</td></tr>
<tr><td>LD.BU disp23 [reg1] , reg4</td></tr>
<tr><td>LD.H disp23 [reg1] , reg3</td></tr>
<tr><td>LD.HU disp23 [reg1] , reg3</td></tr>
<tr><td>LD.W disp23 [reg1] , reg3</td></tr>
<tr><td>ST.B reg3, disp23 [reg1]</td></tr>
<tr><td>ST.H reg3, disp23 [reg2]</td></tr>
<tr><td>ST.W reg3, disp23 [reg3]</td></tr>
</table>

(2/3)

Item			V850E2M	V850E2	V850E1
Instructions (including operands)	Floating to point operation exception		Provided	Not provided	
Number of instruction execution clocks			Varies among certain instructions.		
Program area			4 GB[Note 1]	512 MB	64 MB
Valid bits in program counter (PC)			32 bits[Note 1]	Lower 29 bits	Lower 26 bits
Data area			4 GB		256 M/64 MB
System register bank			Provided	Not provided	
	Main bank		Provided	Provided[Note 2]	
		PSW	Functions differ.		
		ECR	Provided (use is generally prohibited)	Provided	
		EIWR	Provided	Not provided	
		FEWR			
		EIIC			
		FEIC			
		BSEL			
		SCCFG			
		SCBP			
	Exception handler address switching function bank 0 Exception handler address switching function bank 1				
	MPU violation bank				
	MPU setting bank				
	Software paging bank				
	FPU status bank				
		FPEC			
	User 0 bank				

Notes 1. For a CPU whose instruction addressing range is limited by the product specification to 512 MB, a value resulting from a sign to extension of bit 28 of EIPC is automatically set to bits 31 to 29.

2. Bank configuration is not employed and only system registers equivalent to the main bank are available.

(3/3)

Item		V850E2M	V850E2	V850E1
Processor protection function		Functions differ	Not provided	
Exceptions	FE level non to maskable exception	FENMI	NMI2[Note]	
	FE level maskable exception	FEINT	NMI0, NMI1[Note]	
	EI level maskable exception	INT	INT	
	Memory protection exception	Provided (30H)	Not provided	
	Floating to point operation exception	Provided (70H)	Not provided	
	Return from FE level exception	FERET	RETI	
	Return from EI level exception	EIRET		
	Checking and cancelling exception	Provided	Not provided	
	Execution of undefined opcodes	Reserved instruction exception FE level exception (30H)	Illegal instruction exception DB level exception (60H)	
Operation mode	Misaligned access enable setting	Always enabled	Can be set as enabled or disabled	
Pipeline		7 stages		5 stages
		Pipeline flow varies for each instruction.		

Note Some specifications such as exception handler addresses and exception code are different.

APPENDIX F INSTRUCTION INDEX

F.1 Basic Instructions

F.2 Floating to point Operation Instructions

REVISION HISTORY	V850E2M Preliminary User's Manual: Architecture

Rev.	Date	Description	
		Page	Summary
0.01	Dec 09, 2009	—	First Edition issued
0.02	Aug 31, 2010	p.89	Modification of **PART2 5. 3 Instruction Set DISPOSE [Operation]**
		p.180	Modification of **PART2 5. 3 Instruction Set TST1 [Operation]**
		p.255	Modification of **PART3 Table 7-1. Peripheral Device Protection Function Register Set**
		p.279	Modification of **PART3 8.1.2 TSECR – Timing supervision exception cause**
		p.403	Modification of **PART4 4. 4 Instruction Set TRNCF.SL [opcode]**
		p.404	Modification of **PART4 4. 4 Instruction Set TRNCF.SUL [opcode]**
		p.425	Modification of **Table B-1. Basic Instruction Opcode Map (16-/32-bit Instruction)**
		pp.429, 430	Modification of **Table B-4. Floating-point Instruction Opcodes**
		pp.435 to 438	Modification of **Table C-1. Clock Requirements for Basic Instructions**
1.00	Oct 17, 2012	p.186	Addition of **Note 7** in **PART 2, Table 6-1 Exception Cause List**
		p.263	Modification of the destination for reference to initial values in **PART 3, 7.1.9 PPSn – Specification of special peripheral device**
		p.264	Modification of the destination for reference to initial values in **PART 3, 7.1.10 PPPn – Specification of OS peripheral device**
		p.265	Modification of the destination for reference to initial values in **PART 3, 7.1.11 PPVn – Validating general peripheral device protection**
		p.266	Modification of the destination for reference to initial values in **PART 3, 7.1.12 PPTn – Specification protection type of general peripheral device**

V850E2M User’s Manual: Architecture

Publication Date: Rev.1.00 Oct 17, 2012

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.3

V850E2M

Renesas Electronics Corporation

R01US0001EJ0100