

V850E2S

User's Manual: Architecture

RENESAS MCU V850E2S Microprocessor Core

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

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 V850E2S CPU core for designing application systems using the V850E2S CPU core.														
Purpose	This manual is intended for users to understand the architecture of the V850E2S CPU core described in the Organization below.														
Organization	This manual contains the following information: <ul style="list-style-type: none">• Basic function• Processor protection function														
How to Use this Manual	<p>It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.</p> <p>To learn about the hardware functions, → Read Hardware User's Manual of each product.</p> <p>To learn about the functions of a specific instruction in detail, → Read PART 2 CHAPTER 5 INSTRUCTIONS</p>														
Conventions	<table><tr><td>Data significance:</td><td>Higher digits on the left and lower digits on the right</td></tr><tr><td>Active low representation:</td><td>xxxB (B is appended to pin or signal name)</td></tr><tr><td>Note:</td><td>Footnote for item marked with Note in the text</td></tr><tr><td>Caution:</td><td>Information requiring particular attention</td></tr><tr><td>Remark:</td><td>Supplementary information</td></tr><tr><td>Numerical representation:</td><td>Binary ... xxxx or xxxxB Decimal ... xxx Hexadecimal ... xxxxH</td></tr><tr><td>Prefix indicating the power of 2 (address space, memory capacity):</td><td>K (Kilo): $2^{10} = 1,024$ M (Mega): $2^{20} = 1,024^2$ G (Giga): $2^{30} = 1,024^3$</td></tr></table>	Data significance:	Higher digits on the left and lower digits on the right	Active low representation:	xxxB (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 ... xxxx or xxxxB Decimal ... xxx Hexadecimal ... xxxxH	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): $2^{30} = 1,024^3$
Data significance:	Higher digits on the left and lower digits on the right														
Active low representation:	xxxB (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 ... xxxx or xxxxB Decimal ... xxx Hexadecimal ... xxxxH														
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	12
CHAPTER 1 FEATURES	13
1.1 Basic function	13
1.2 Processor protection function	13
PART 2 BASIC FUNCTION.....	14
CHAPTER 1 OVERVIEW	15
1.1 Features	15
CHAPTER 2 REGISTER SET	17
2.1 Program Registers.....	18
2.2 System Register Bank.....	19
2.2.1 BSEL – Register bank selection	21
2.3 CPU Function Group/Main Bank	22
2.3.1 EIPC and EIPSW – Status save registers when acknowledging EI level exception	23
2.3.2 FEPC and FEPSW – Status save registers when acknowledging FE level exception	23
2.3.3 ECR – Exception cause	24
2.3.4 PSW – Program status word	25
2.3.5 SCCFG – SYSCALL operation setting	28
2.3.6 SCBP – SYSCALL base pointer	28
2.3.7 EIIC – EI level exception cause	29
2.3.8 FEIC – FE level exception cause.....	29
2.3.9 CTPC and CTPSW – Status save registers when executing CALLT	29
2.3.10 CTBP – CALLT base pointer	30
2.3.11 EIWR – EI level exception working register	30
2.3.12 FEWR – FE level exception working register.....	30
2.3.13 DBIC – DB level exception cause	31
2.3.14 DBPC and DBPSW – Status save registers when acknowledging DB level exception.....	31
2.3.15 DBWR – DB level exception working register	31
2.3.16 DIR – Debug interface register	31
2.4 CPU Function Group/Exception Handler Address Switching Function Banks.....	32
2.4.1 SW_CTL – Exception handler address switching control	34
2.4.2 SW_CFG – Exception handler address switching configuration	34
2.4.3 SW_BASE – Exception handler address switching base address	34
2.4.4 EH_CFG – Exception handler configuration	35
2.4.5 EH_BASE – Exception handler base address	35
2.4.6 EH_RESET – Reset address.....	36
2.5 User Group	37
CHAPTER 3 DATA TYPES.....	38
3.1 Data Formats	38
3.1.1 Byte	38
3.1.2 Halfword	38
3.1.3 Word.....	39

3.1.4 Bit.....	39
3.2 Data Representation	40
3.2.1 Integers.....	40
3.2.2 Unsigned integers	40
3.2.3 Bits.....	40
3.3 Data Alignment.....	41
CHAPTER 4 ADDRESS SPACE.....	42
4.1 Memory Map	43
4.2 Addressing Modes	45
4.2.1 Instruction address.....	45
4.2.2 Operand address	48
CHAPTER 5 INSTRUCTIONS.....	51
5.1 Opcodes and Instruction Formats	51
5.1.1 CPU instructions	51
5.1.2 Coprocessor instructions.....	56
5.1.3 Reserved instructions	56
5.2 Overview of Instructions	57
5.3 Instruction Set	62
ADD	65
ADDI	66
ADF	67
AND	68
ANDI	69
Bcond	70
BSH	72
BSW	73
CALLT.....	74
CAXI	75
CLR1	76
CMOV	78
CMP	80
CTRET.....	81
DI	82
DISPOSE.....	83
DIV	85
DIVH	86
DIVHU.....	88
DIVQ	89
DIVQU.....	90
DIVU	91
EI	92
EIRET	93
FERET	94
FETRAP.....	95
HALT	96
HSH	97
HSW	98
JARL	99

JMP	101
JR	102
LD.B	103
LD.BU	104
LD.H	105
LD.HU	106
LD.W	107
LDSR	108
MAC	109
MACU	110
MOV	111
MOVEA	112
MOVHI	113
MUL	114
MULH	115
MULHI	116
MULU	117
NOP	118
NOT	119
NOT1	120
OR	122
ORI	123
PREPARE	124
RETI	126
RIE	128
SAR	129
SASF	131
SATADD	132
SATSUB	134
SATSUBI	135
SATSUBR	136
SBF	137
SCH0L	138
SCH0R	139
SCH1L	140
SCH1R	141
SET1	142
SETF	144
SHL	146
SHR	148
SLD.B	150
SLD.BU	151
SLD.H	152
SLD.HU	153
SLD.W	154
SST.B	155
SST.H	156
SST.W	157
ST.B	158
ST.H	159

ST.W	160
STSR	161
SUB	162
SUBR	163
SWITCH	164
SXB	165
SXH	166
SYNCE	167
SYNCM	168
SYNCP	169
SYSCALL	170
TRAP	172
TST	173
TST1	174
XOR	175
XORI	176
ZXB	177
ZXH	178
CHAPTER 6 EXCEPTIONS.....	179
6.1 Outline of Exceptions	179
6.1.1 Exception cause list	179
6.1.2 Types of exceptions	182
6.1.3 Exception processing flow.....	184
6.1.4 Exception acknowledgment priority and pending conditions	185
6.1.5 Exception acknowledgment conditions	185
6.1.6 Resume and restoration.....	185
6.1.7 Exception level and context saving	186
6.1.8 Return instructions	187
6.2 Operations When Exception Occurs.....	190
6.2.1 EI level exception without acknowledgment conditions	190
6.2.2 EI level exception with acknowledgment conditions	192
6.2.3 FE level exception without acknowledgment conditions	194
6.2.4 FE level exception with acknowledgment conditions	196
6.2.5 Special operations	198
6.3 Exception Management	200
6.4 Exception Handler Address Switching Function.....	201
6.4.1 Determining exception handler addresses	201
6.4.2 Purpose of exception handler address switching	202
6.4.3 Settings for exception handler address switching function	202
CHAPTER 7 COPROCESSOR UNUSABLE STATUS.....	203
7.1 Coprocessor Unusable Exception	203
7.2 System Registers	203
CHAPTER 8 RESET	204
8.1 Status of Registers After Reset	204
8.2 Start	204

PART 3 PROCESSOR PROTECTION FUNCTION	205
CHAPTER 1 OVERVIEW	206
1.1 Features	206
CHAPTER 2 REGISTER SET	208
2.1 System Register Bank	208
2.2 System Registers	210
2.2.1 PSW – Program Status Word	213
2.2.2 MPM – Setting of processor protection operation mode	214
2.2.3 MPC – Specification of processor protection command	217
2.2.4 TID – Task identifier	217
2.2.5 Other system registers	217
CHAPTER 3 OPERATION SETTING.....	218
3.1 Starting Use of Processor Protection Function	218
3.2 Setting of Execution Level Auto Transition Function.....	218
3.3 Stopping Use of Processor Protection Function	218
CHAPTER 4 EXECUTION LEVEL	219
4.1 Nature of Program	219
4.2 Protection Bits on PSW.....	220
4.2.1 T state (trusted state)	220
4.2.2 NT state (non-trusted state).....	220
4.3 Definition of Execution Level.....	220
4.4 Transition of Execution Level.....	221
4.4.1 Transition by execution of write instruction to system register	221
4.4.2 Transition as result of occurrence of exception	222
4.4.3 Transition by execution of return instruction	222
4.5 Program Model.....	223
4.6 Task Identifier.....	224
CHAPTER 5 SYSTEM REGISTER PROTECTION.....	225
5.1 Register Set	226
5.1.1 VSECR – System register protection violation cause	227
5.1.2 VSTID – System register protection violation task identifier	227
5.1.3 VSADR – System register protection violation address	228
5.2 Access Control.....	229
5.3 Registers to Be Protected	229
5.4 Detection of Violation	230
5.5 Operation Method	230
CHAPTER 6 MEMORY PROTECTION.....	231
6.1 Register Set	232
6.1.1 PAnL – Protection area n lower-limit address (n = 0 to 3)	233
6.1.2 PAnU – Protection area n upper-limit address (n = 0 to 3)	234
6.1.3 VMECR – Memory protection violation cause	235
6.1.4 VMTID – Memory protection violation task identifier.....	236
6.1.5 VMADR – Memory protection violation address	236

6.2	Access Control	237
6.3	Setting Protection Area	238
6.3.1	Valid bit (E bit).....	238
6.3.2	Execution enable bit (X bit)	238
6.3.3	Read enable bit (R bit)	239
6.3.4	Write enable bit (W bit)	239
6.3.5	sp indirect access enable bit (S bit)	239
6.3.6	Protection area lower-limit address (AL31 to AL0 bits)	239
6.3.7	Protection area upper-limit address (AU31 to AU0 bits).....	239
6.4	Notes on Setting Protection Area	240
6.4.1	Crossing of protection area boundaries	240
6.4.2	Invalid protection area setting	240
6.5	Special Memory Access Instructions	241
6.5.1	Load and store instructions executing misaligned access.....	241
6.5.2	Some bit manipulation instructions and CAXI instruction	241
6.5.3	Stack frame manipulation instructions.....	241
6.5.4	SYSCALL instruction	241
6.6	Protection Violation and Exception	242
CHAPTER 7 PROCESSOR PROTECTION EXCEPTION		243
7.1	Types of Violations	243
7.1.1	System register protection violation	243
7.1.2	Execution protection violation	243
7.1.3	Data protection violation.....	243
7.2	Types of Exceptions	244
7.2.1	MIP exception	244
7.2.2	MDP exception.....	244
7.3	Identifying Violation Cause	245
7.3.1	MIP exception	245
7.3.2	MDP exception.....	245
CHAPTER 8 SPECIFAL FUNCTON		247
8.1	Clearing Memory Protection Setting All at Once	247
APPENDIX A LIST OF INSTRUCTIONS		248
A.1	Basic Instructions	248
APPENDIX B INSTRUCTION OPCODE MAP		252
B.1	Basic Instruction Opcode Map	252
APPENDIX C PIPELINES		257
C.1	Features	259
C.2	Clock Requirements	261
C.2.1	Clock requirements for basic instructions.....	261
C.3	Pipeline for Basic Instructions	266
C.3.1	Load instructions.....	266
C.3.2	Store instructions	266
C.3.3	Multiply instructions.....	267
C.3.4	Multiply-accumulate instructions	268
C.3.5	Arithmetic operation instructions	269

C.3.6	Conditional operation instructions.....	269
C.3.7	Saturated operation instructions.....	269
C.3.8	Logic operation instructions.....	269
C.3.9	Data manipulation instructions.....	270
C.3.10	Bit search instructions.....	270
C.3.11	Divide instructions.....	270
C.3.12	High-speed divide instructions.....	271
C.3.13	Branch instructions.....	272
C.3.14	Bit manipulation instructions.....	274
C.3.15	Special instructions.....	275
APPENDIX D DIFFERENCES BETWEEN V850E2S CPU AND OTHER CPUS.....		279
D.1	Difference Between V850E2 and V850E2M.....	279
APPENDIX E INSTRUCTION INDEX.....		282
E.1	Basic Instructions.....	282

PART 1 OVERVIEW

CHAPTER 1 FEATURES

The V850E2S 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 V850E2S CPU supplies the following functions.

- (1) **Basic function**
- (2) **Processor protection function**

The V850E2S CPU executes almost all instructions, such as for address calculation, arithmetic and logic operations, and data transfer, with one clock under control of a 5-stage pipeline.

The V850E2S CPU can use the software resources of the conventional system as is because it is upwardly compatible with the V850 CPU, V850E1 CPU, V850E2 CPU, V850E2M CPU, and V850ES 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 and load/store instructions with an extended displacement range are also provided.

The basic function 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 (16 bits x 16 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 and system registers, 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 and memory protection.

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

PART 2 BASIC FUNCTION

CHAPTER 1 OVERVIEW

The V850E2S 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) High-function OS support

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

(4) 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)

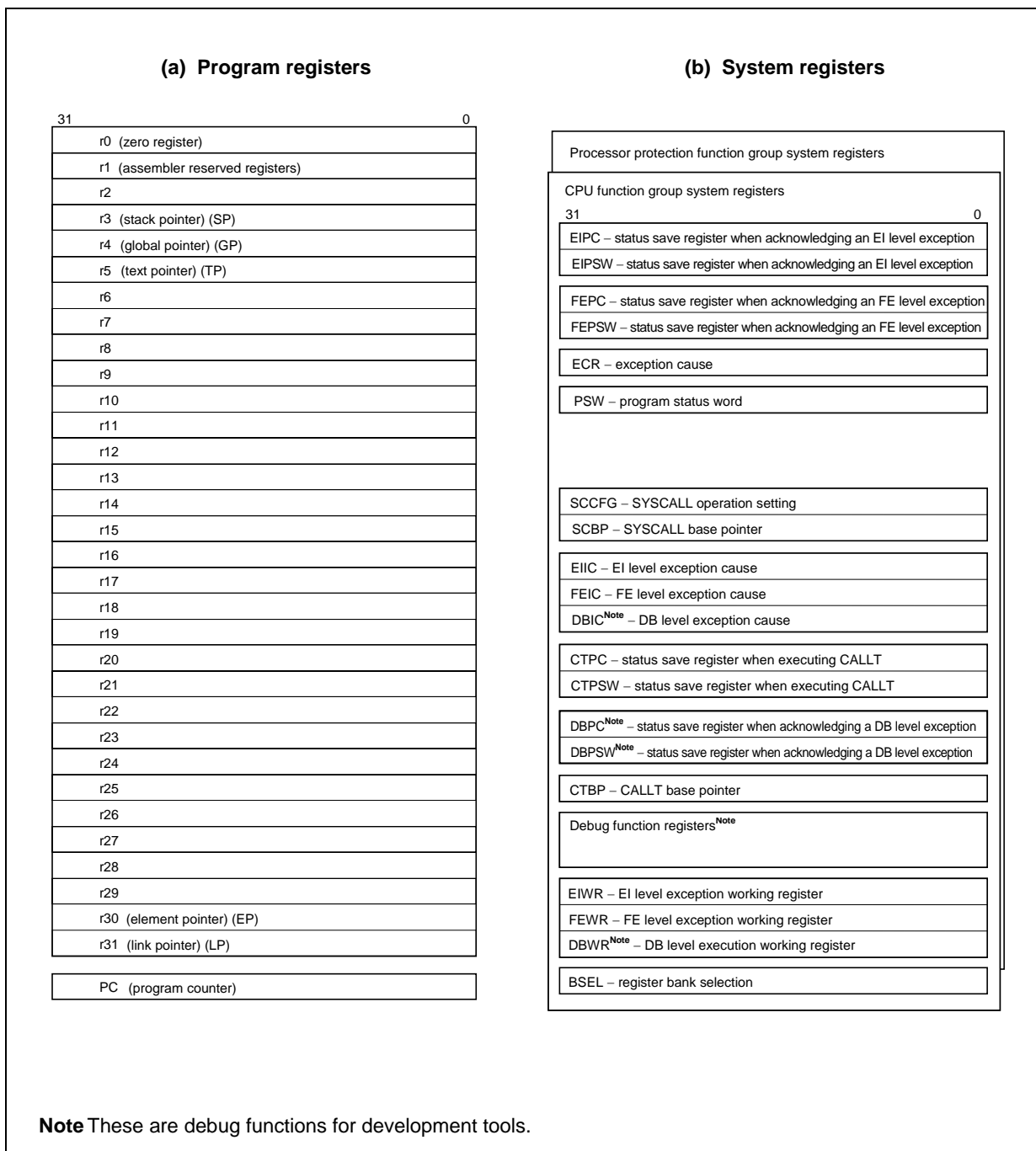
(3) Instructions suitable for high-function/high-performance programming

- 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



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.

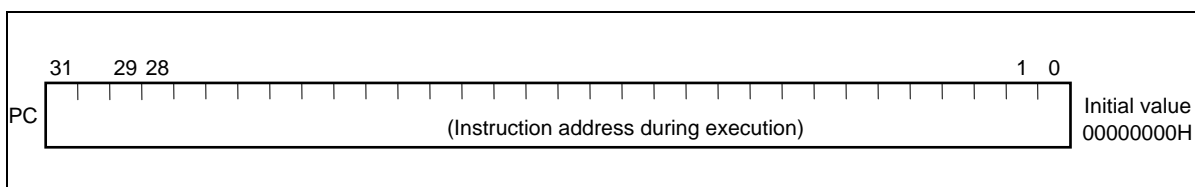
When using these registers, register contents must first be saved so they are not lost and can be restored after the registers are used.

(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 **Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 is automatically set to bits 31 to 26.**

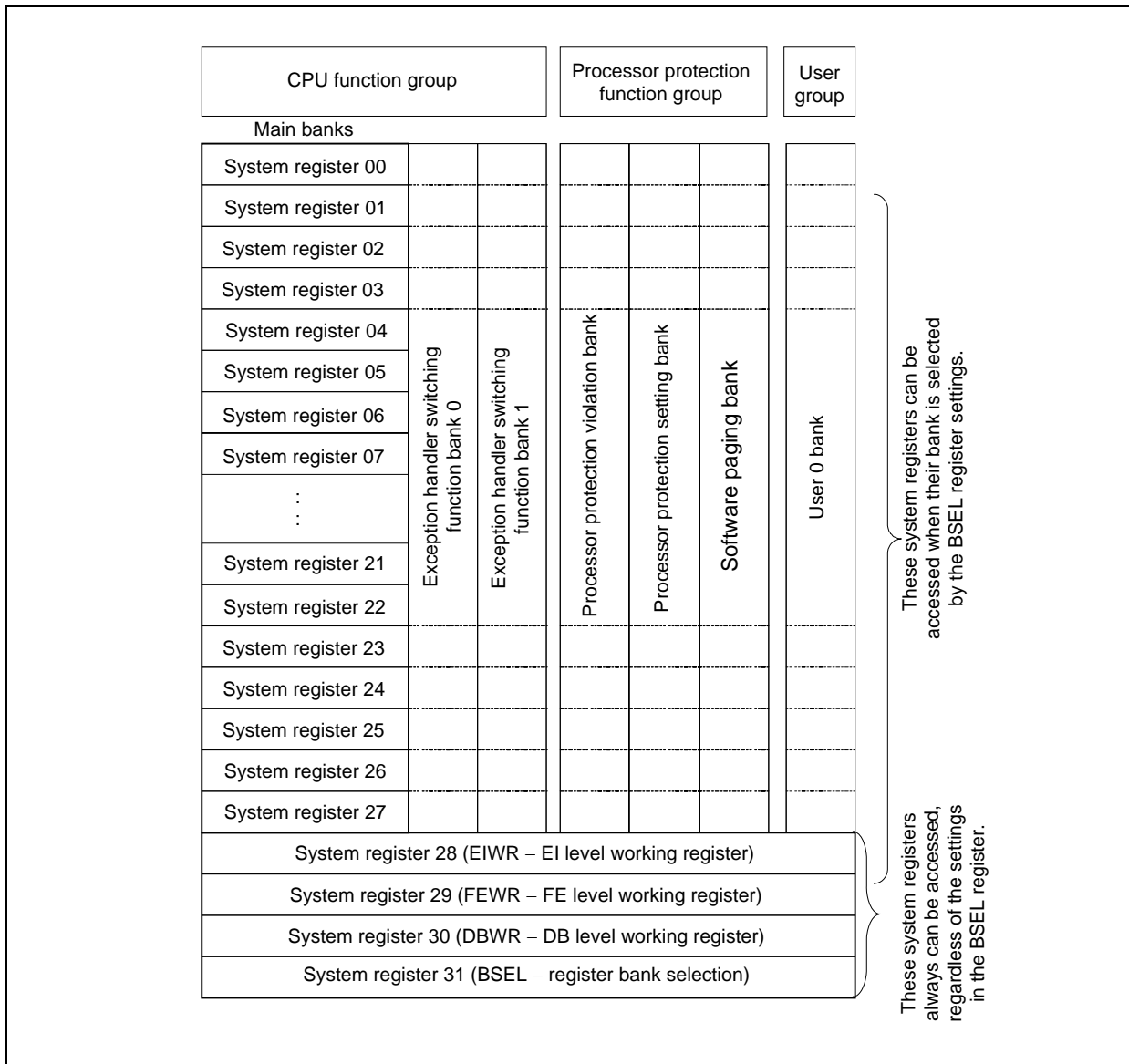
2.2 System Register Bank

The V850E2S 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 V850E2S 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
- User group
 - User 0 bank: This bank is able to access only system registers used by user applications.

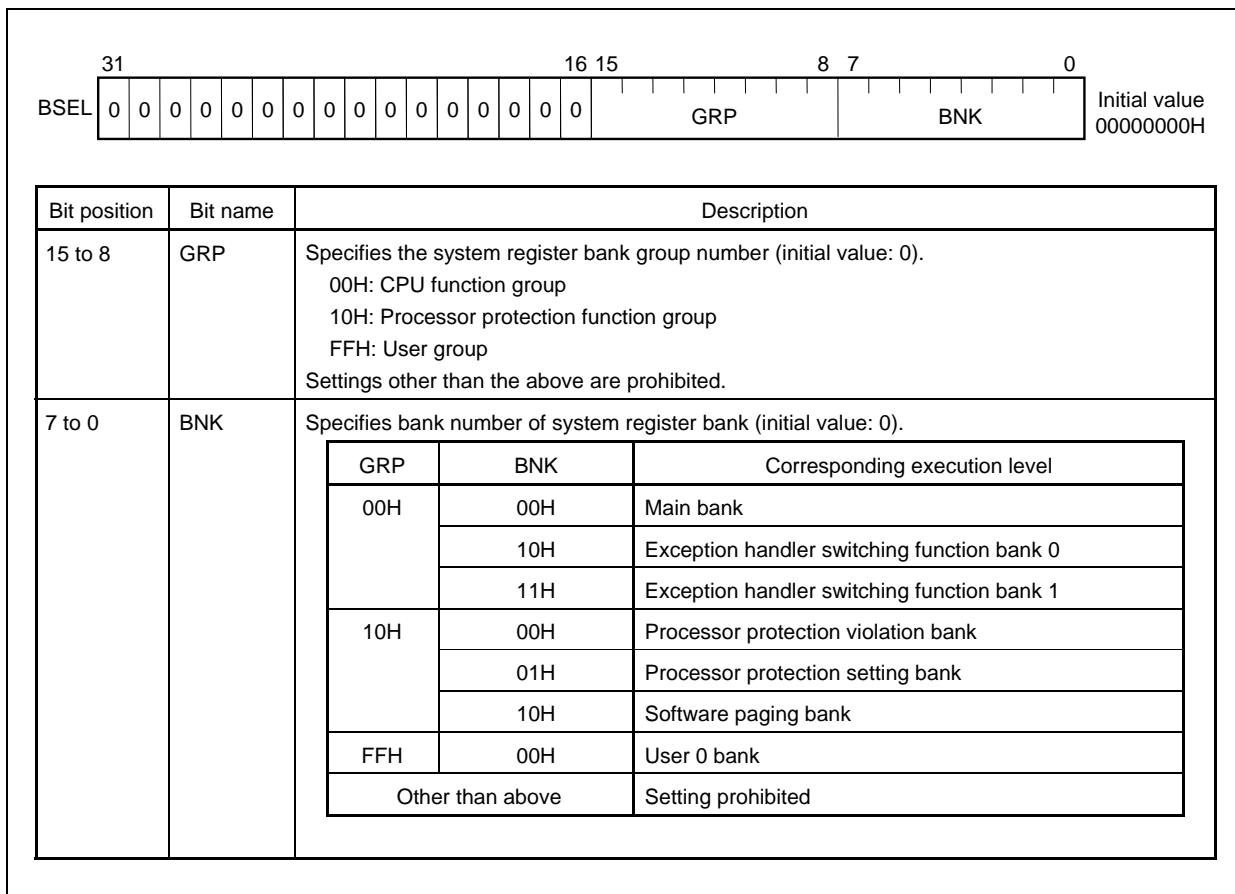
Figure 2-2. System Register Bank



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.



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**.

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	√ ^{Note3}	√ ^{Note4}	√
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.
3. Can be written (updated) only in the debug mode.
4. The value read from some bits may be undefined in the user mode.

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

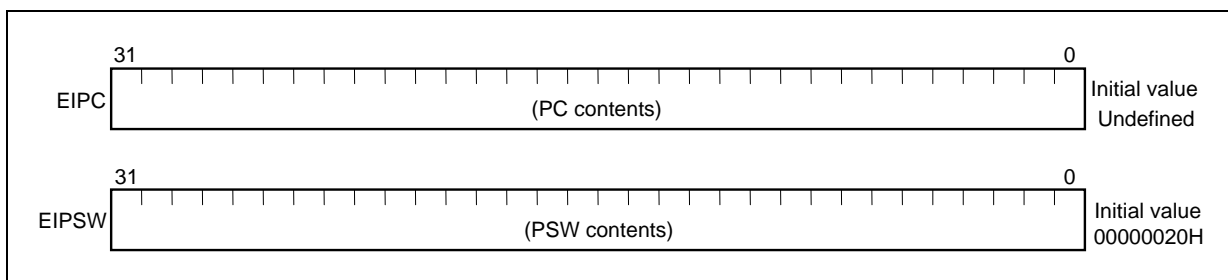
The EI level exception status save registers include EIPC and EIPSW.

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 Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of EIPC is automatically set to bits 31 to 26.

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

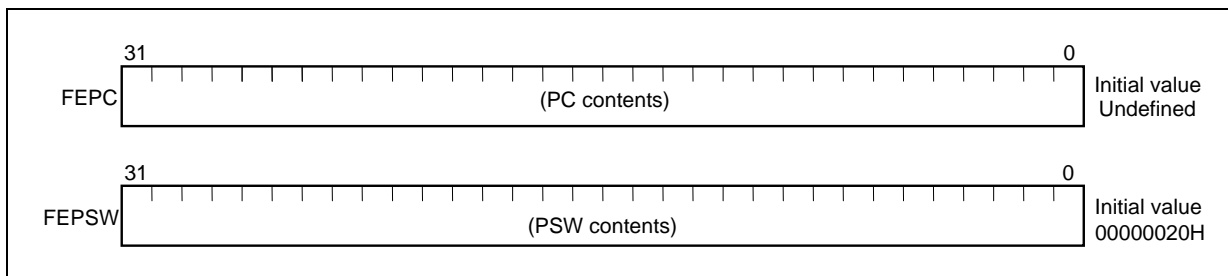
The FE level exception status save register include FEPC and FEPSW.

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.

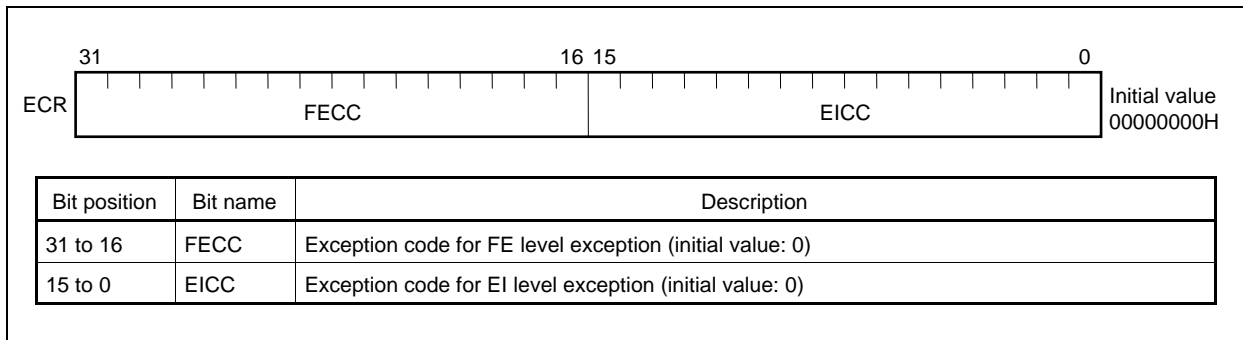


Caution Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of FEPC is automatically set to bits 31 to 26.

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 EIC register or the FEIC register to overwrite all parts that were using the ECR register.**



(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
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)
<p>Note</p> <p>F</p> <p>or details of interrupts, see 6.1.2 Types of exceptions.</p>		

(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.

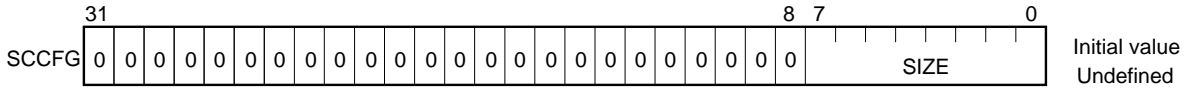
Note The operation result of the saturation processing is determined in accordance with the contents of the OV flag and S flag during a saturated operation. When only the OV flag is set (1) during a saturated operation, the SAT flag is set (1).

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.



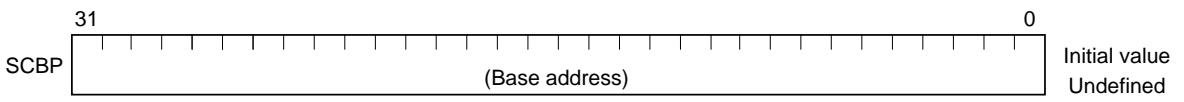
Bit position	Bit name	Description
7 to 0	SIZE	These bits specify the maximum number of entries of a table that the SYSCALL instruction references. The maximum number of entries the SYSCALL instruction references is 1 if SIZE is 0, and 256 if SIZE is 255. By setting the maximum number of entries appropriately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

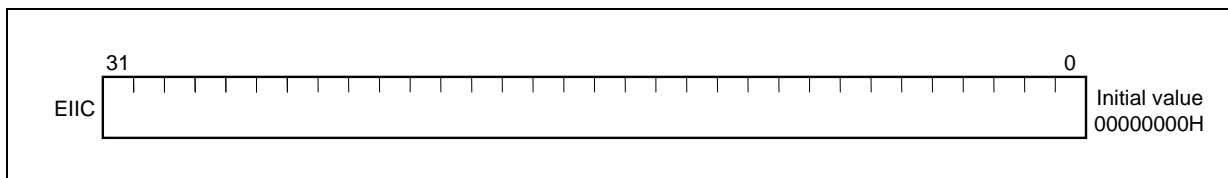
Note that bits 1 and 0 are fixed to 0.



Caution Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of SCBP is automatically set to bits 31 to 26.

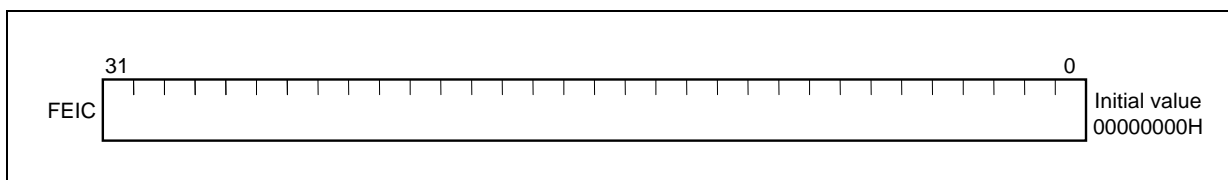
2.3.7 EIIC – EI level exception cause

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**).



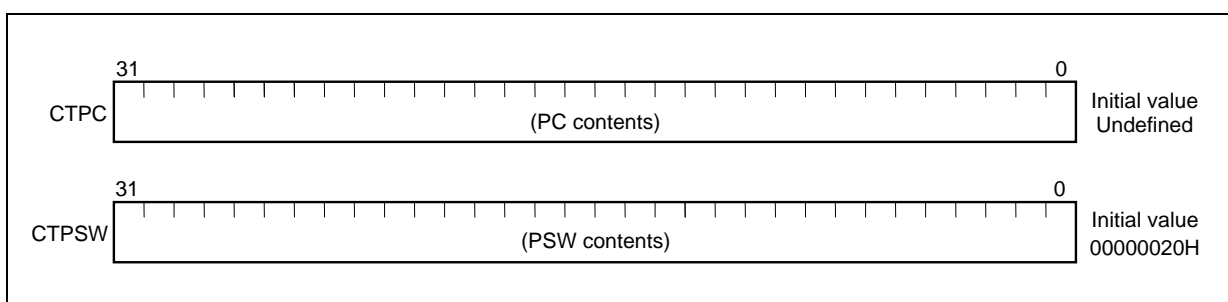
2.3.9 CTPC and CTPSW – Status save registers when executing CALLT

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

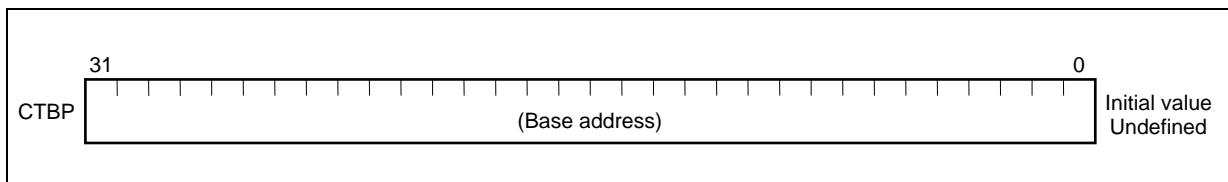
Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of CTPC is automatically set to bits 31 to 26.

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.

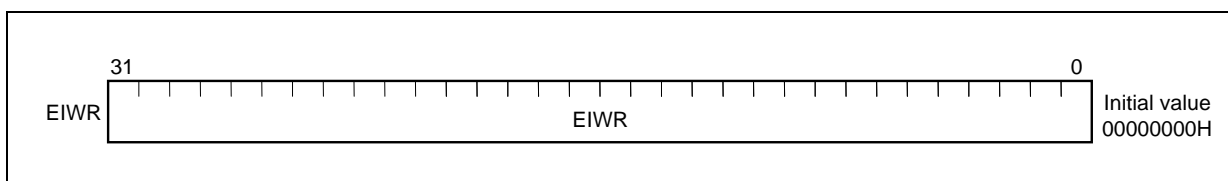


Caution Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of CTBP is automatically set to bits 31 to 26.

2.3.11 EIWR – EI level exception working register

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

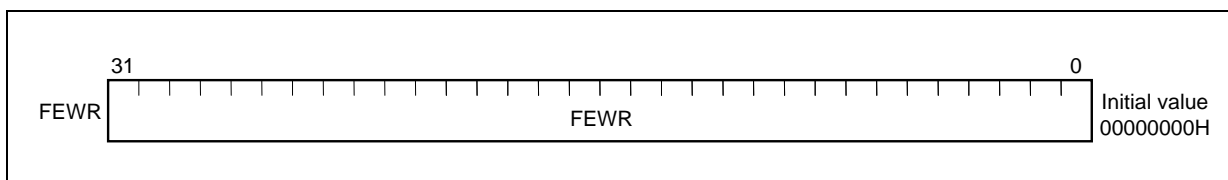
The EIWR register can always be referenced, regardless of which bank is selected.



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”.

31	1	0	Initial value 00000000H
0	0	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”.

31	1	0	Initial value 0000000xH
0	0	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”.

31	29	28	13	12	0	Initial value Undefined
0	SW_BASE31 to SW_BASE13				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 Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of SW_BASE is automatically set to bits 31 to 26.

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.

EH_CFG	31		1 0 RINT	Initial value 000000xH
--------	----	--	-------------	---------------------------

Bit position	Bit name	Description
0	RINT	When the RINT bit is set (1), exception handler addresses INT0 to INT127 are reduced to a single handler address (INT0). When this bit is cleared (0), INT0 to INT127 are held as independent exception handler addresses. The EH_CFG register is set to the initial value defined for each product at reset. 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.

EH_BASE	31		29 28	13 12	0	Initial value Undefined
---------	----	--	-------	-------	---	----------------------------

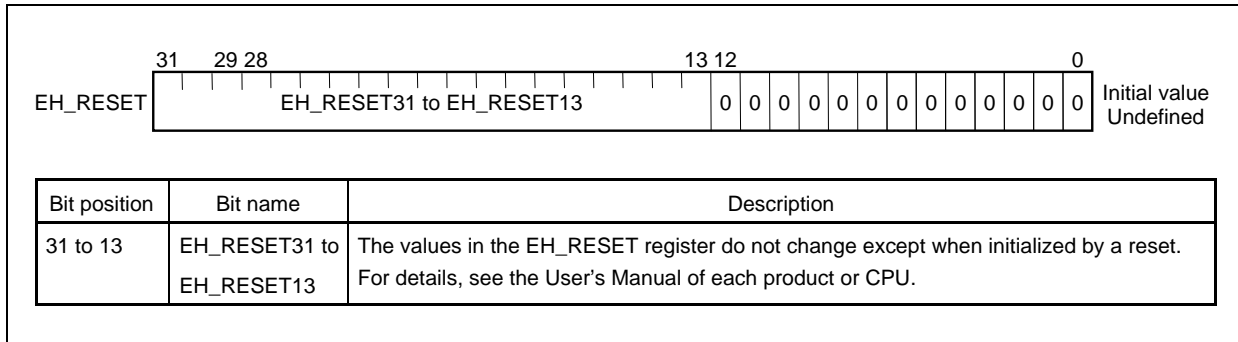
Bit position	Bit name	Description
31 to 13	EH_BASE31 to EH_BASE13	Addresses are changed by adding an offset address for each exception to the base addresses of exception handler routines specified in this register. The EH_BASE register is set to the initial value defined for each product at reset. When the SW_CTL.SET bit is set (1), the contents of SW_BASE are transferred.

Caution **Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of EH_BASE is automatically set to bits 31 to 26.**

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.



Caution Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of EH_RESET is automatically set to bits 31 to 26.

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. The user group includes the following bank.

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

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}
6-15		(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. 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. 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

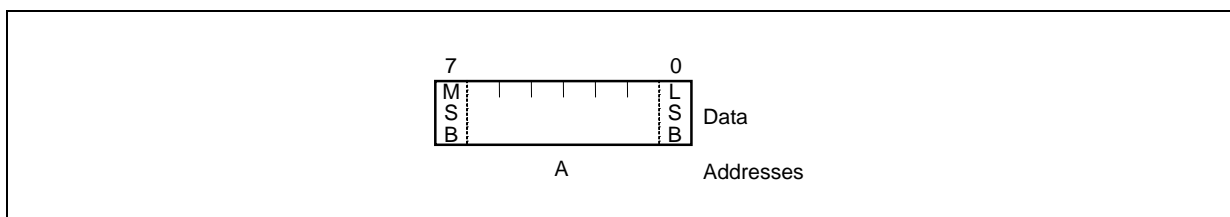
The V850E2S CPU handles data in little endian format. This means that byte 0 of a halfword or a word is always the least significant (rightmost) byte.

The supported data format is as follows.

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

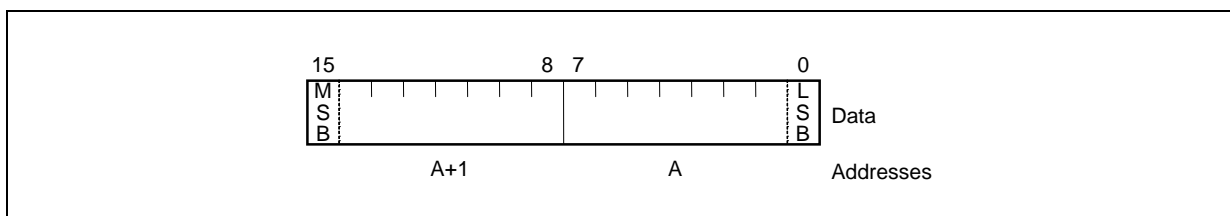
3.1.1 Byte

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

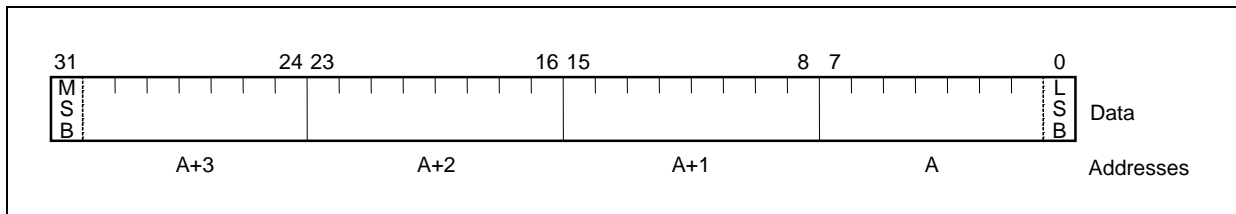
A halfword is two consecutive bytes (16 bits) of data that starts from any byte boundary^{Note}. Numbers from 0 to 15 are assigned to these bits, with bit 0 as the LSB and bit 15 as the MSB. The bytes in a halfword are specified using address "A", so that the two addresses comprise byte data of "A" and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



Note During word access, the V850E2S CPU can be accessed at all byte boundaries.
See **3.3 Data Alignment**.

3.1.3 Word

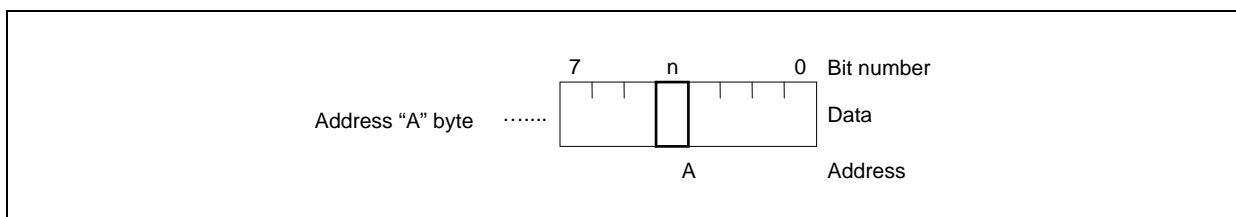
A word is four consecutive bytes (32 bits) of data that starts from any byte boundary^{Note}. Numbers from 0 to 31 are assigned to these bits, with bit 0 as the LSB (least significant bit) and bit 31 as the MSB (most significant bit). A word is specified by address “A” and consists of byte data of four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



Note During word access, the V850E2S 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

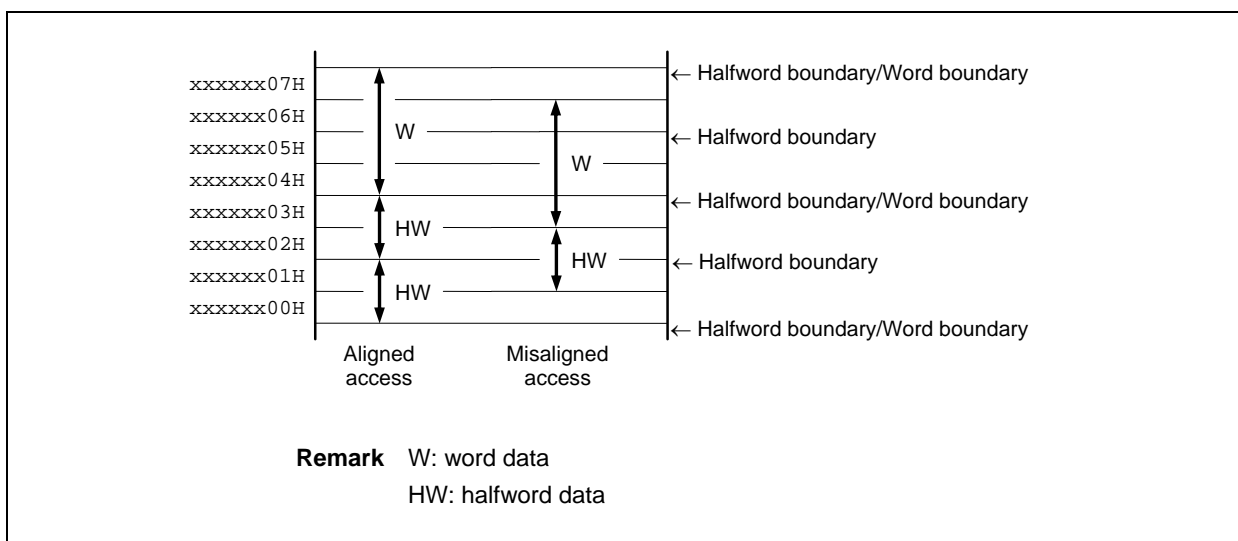
The V850E2S CPU allows misaligned placement of data.

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

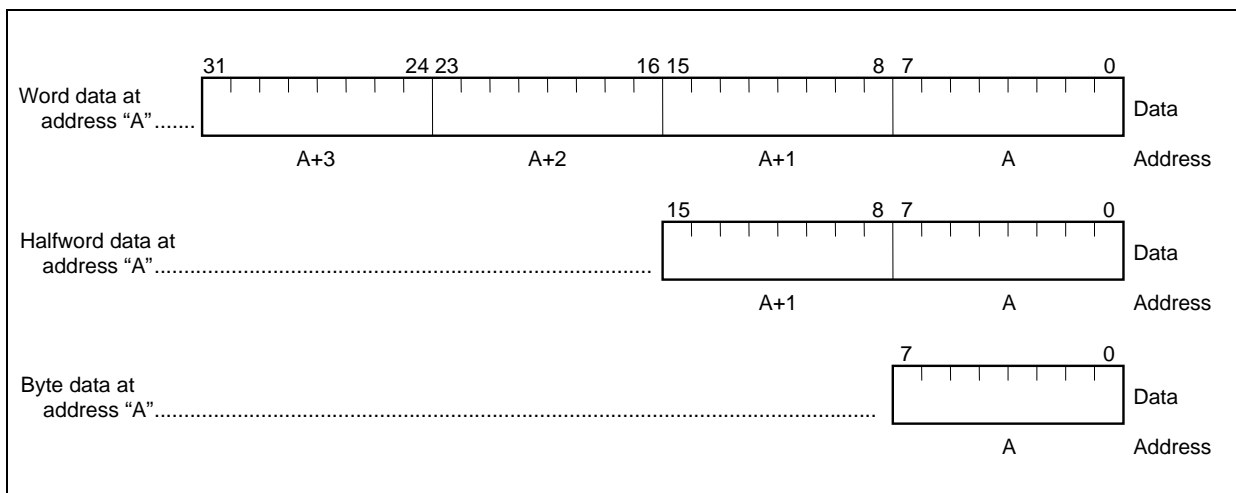


CHAPTER 4 ADDRESS SPACE

The V850E2S 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.

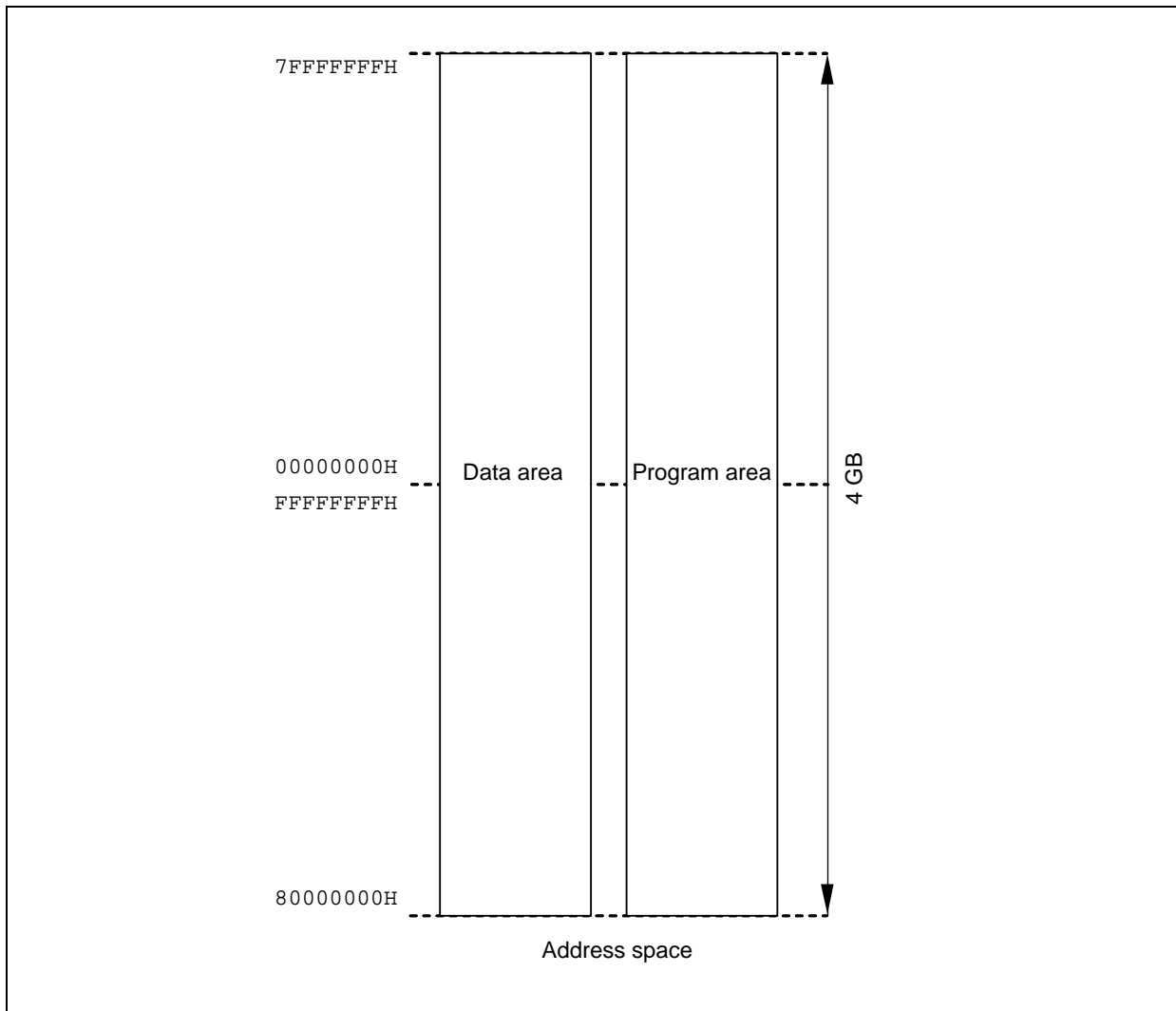


4.1 Memory Map

The V850E2S 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).

A memory map is shown in Figure 4-1.

Figure 4-1. Memory Map (Address Space)

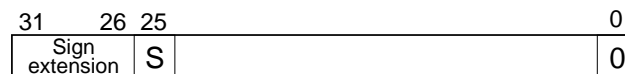


Caution For the V850E2S CPU, the range of the 4 GB program area that can be actually addressed is

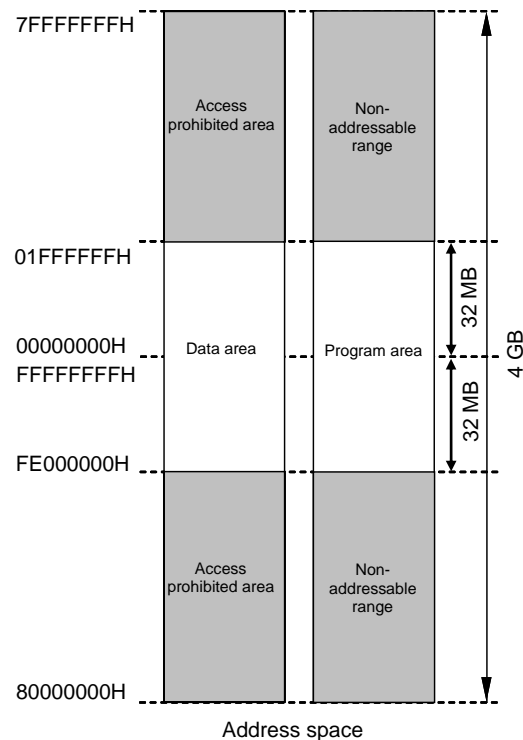
a 64 MB because of physical restrictions of registers that hold an instruction address (such as the program counter). The following registers hold an instruction address.

- 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)
- VSADR (processor protection function)

With the V850E2S CPU whose addressable range of the program area is a 64 MB, the higher 6 bits of these registers are automatically set to values resulting from a sign-extension of bit 25. Therefore, the addressable ranges are 00000000H to 01FFFFFFEH and FE000000H to FFFFFFFFEH (the least significant bit is always 0).



The memory map in this case is shown below.



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 data must also be assigned within a range of 64 MB.

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.

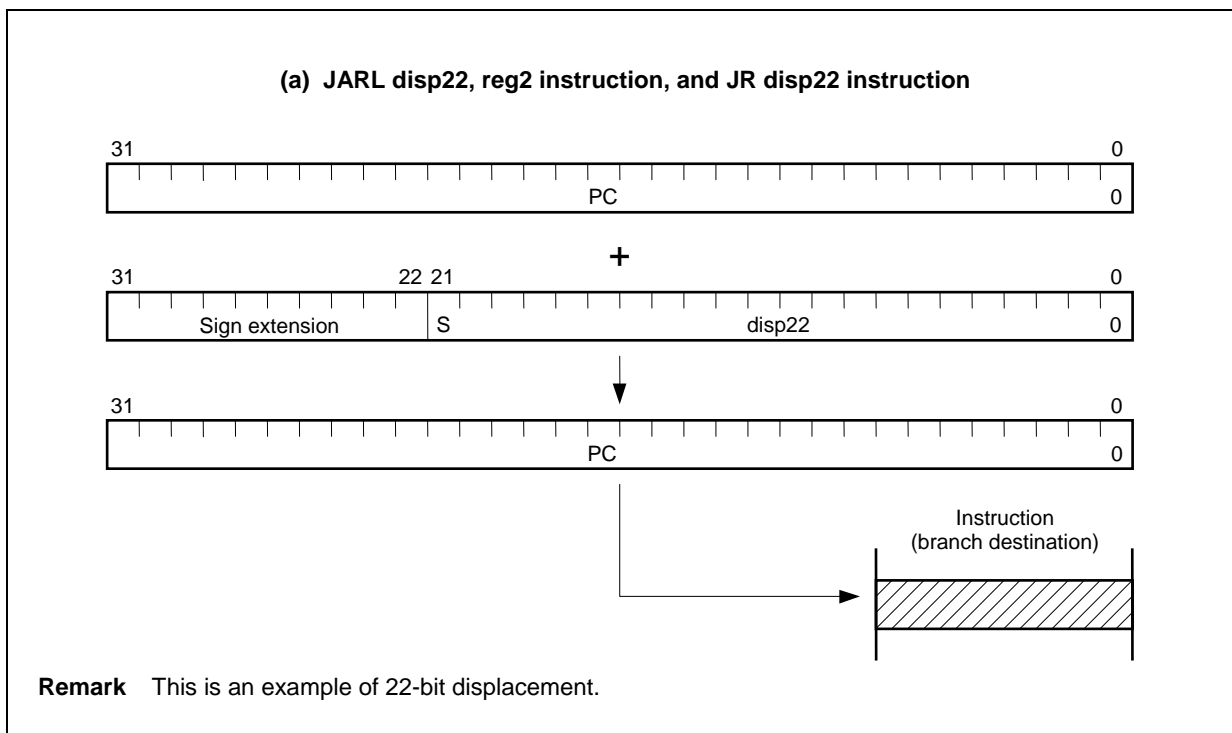
(1) Relative addressing (PC relative)

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.

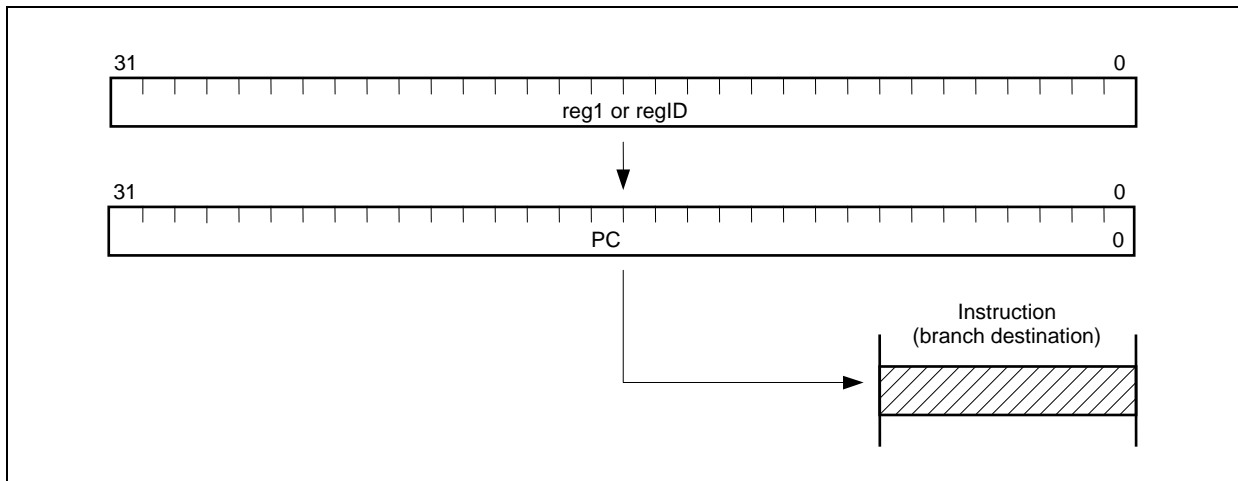
Figure 4-2. Relative Addressing



(2) Register addressing (register indirect)

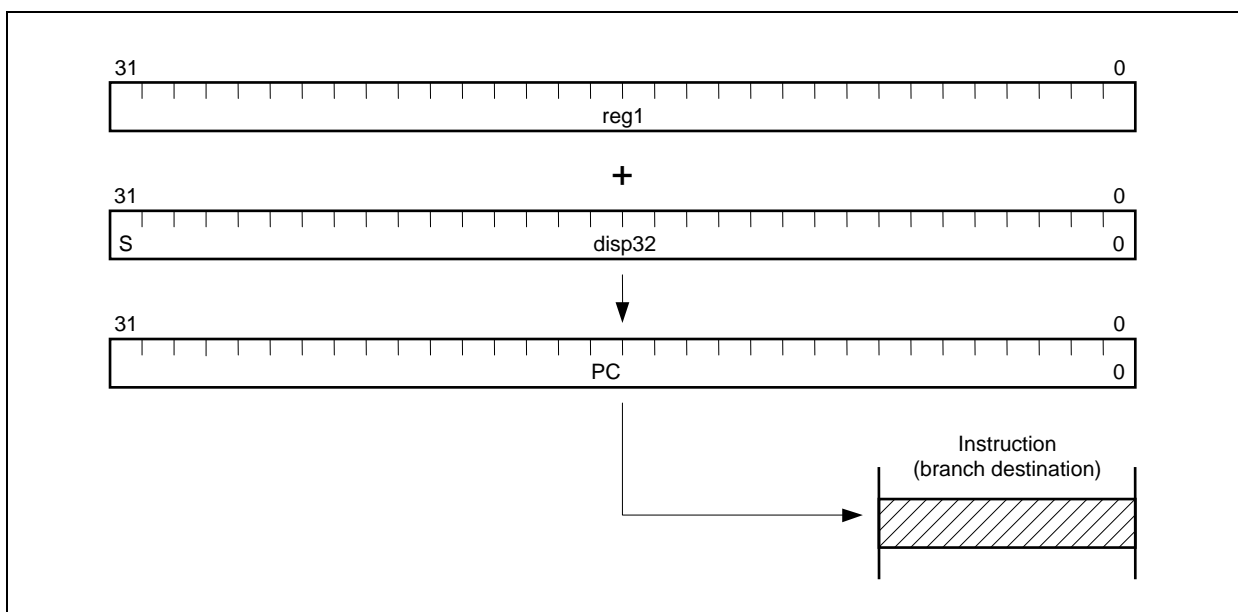
The contents of the general-purpose register (reg1) or system register (regID) specified by the instruction are transferred to the program counter (PC).

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

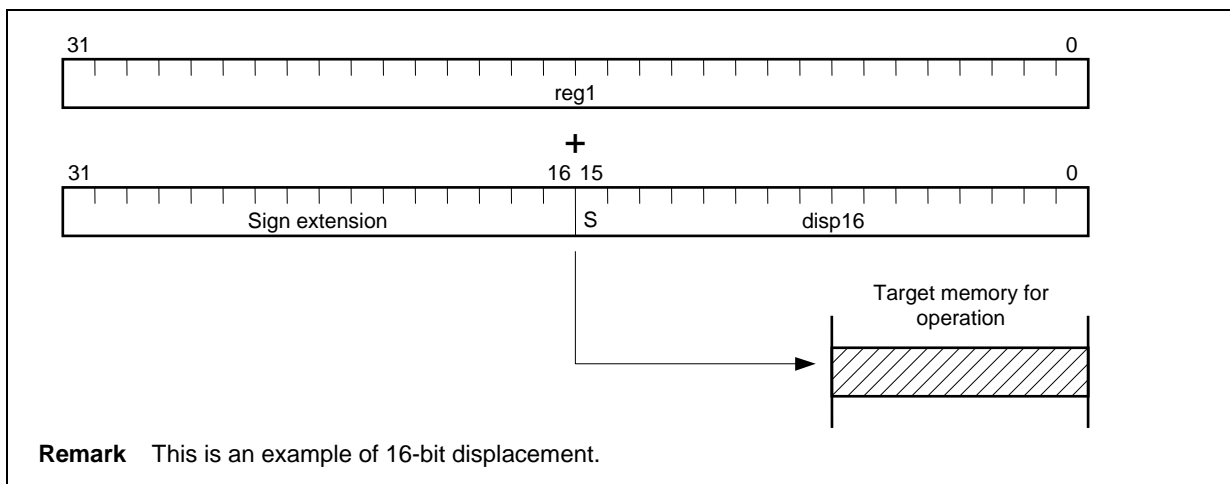
There are two types of based addressing, as described below.

(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)

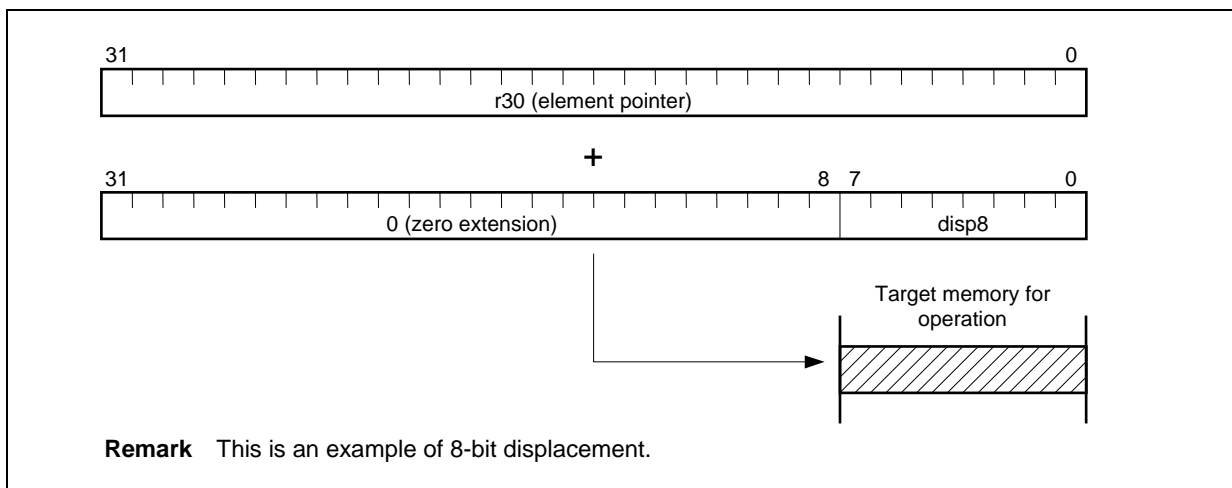


(b) Type 2

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)

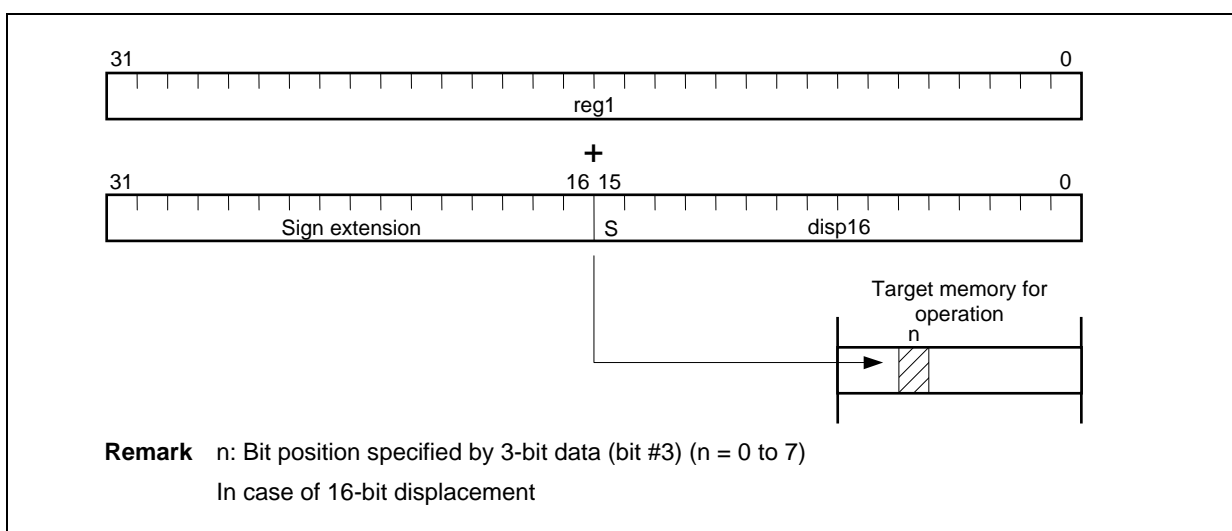


(4) Bit addressing

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



(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 V850E2S CPU has CPU instructions which are defined as basic instructions.

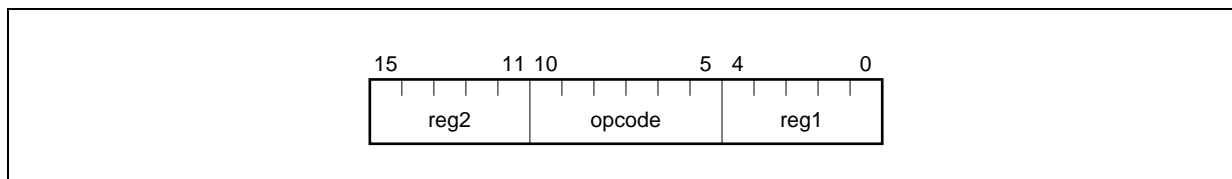
5.1.1 CPU instructions

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**.

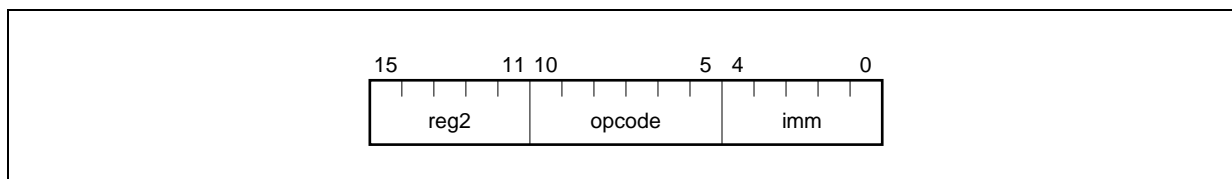
(1) reg-reg instruction (Format I)

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



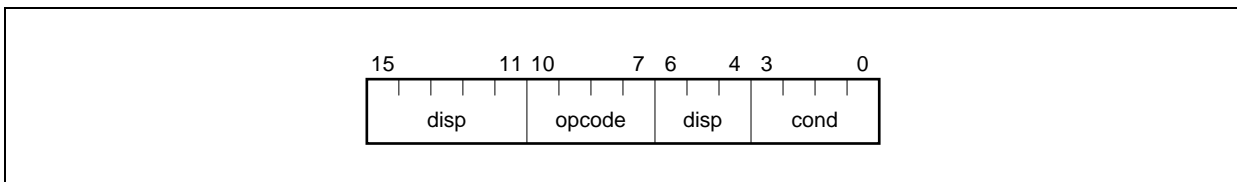
(2) imm-reg instruction (Format II)

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

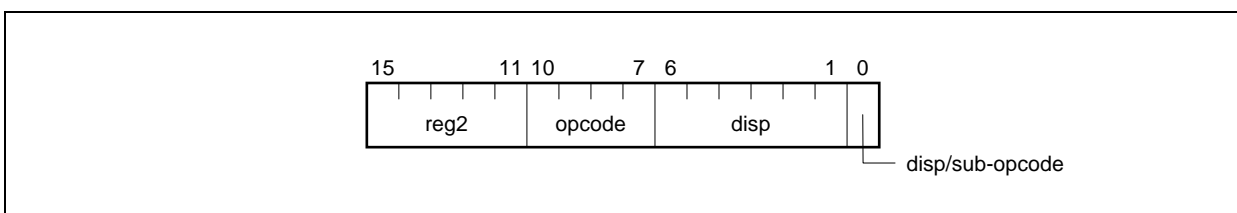


(3) Conditional branch instruction (Format III)

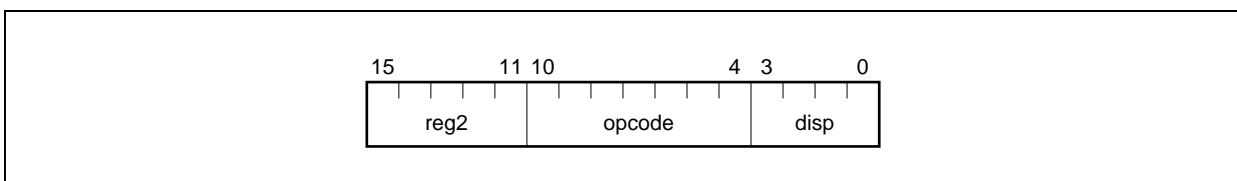
A 16-bit instruction format consists of a 4-bit opcode field, 4-bit condition code field, and an 8-bit displacement field.

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

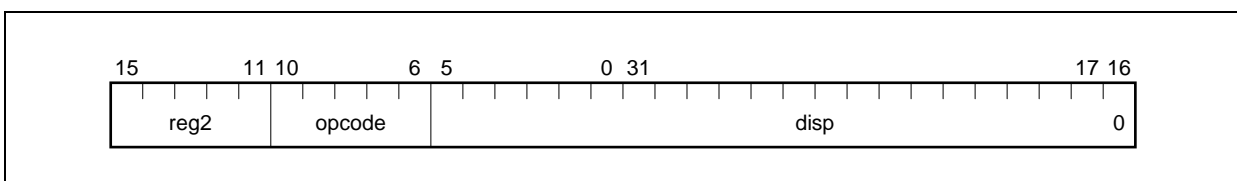
A 16-bit instruction format consists of a 4-bit opcode field, a general-purpose register specification field, and a 7-bit displacement field (or 6-bit displacement field + 1-bit sub-opcode field).



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

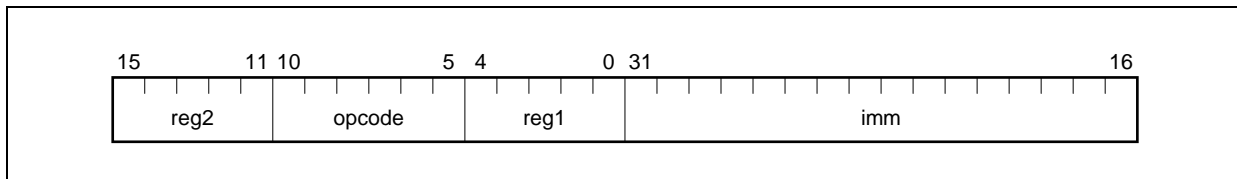
**(5) Jump instruction (Format V)**

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

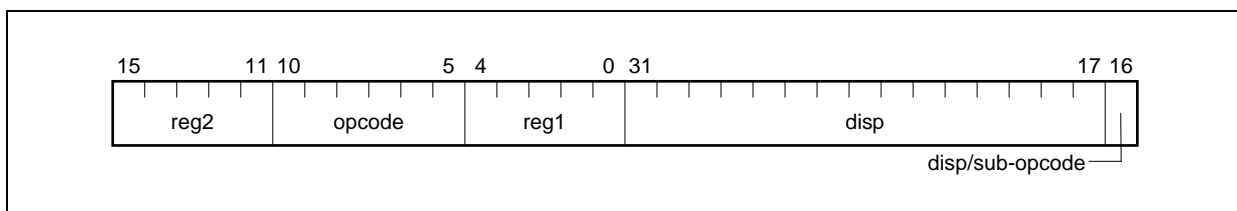


(6) 3-operand instruction (Format VI)

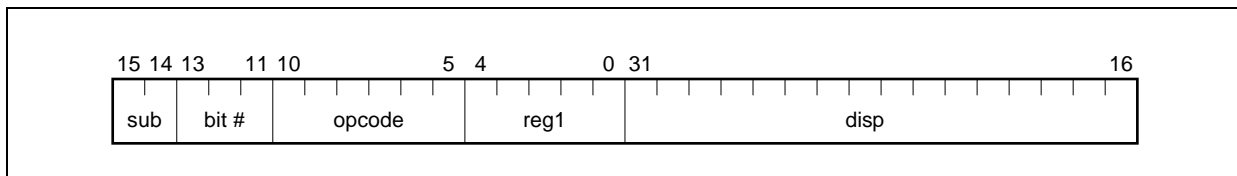
A 32-bit instruction format consists of a 6-bit opcode field, two general-purpose register specification fields, and a 16-bit immediate field.

**(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)**

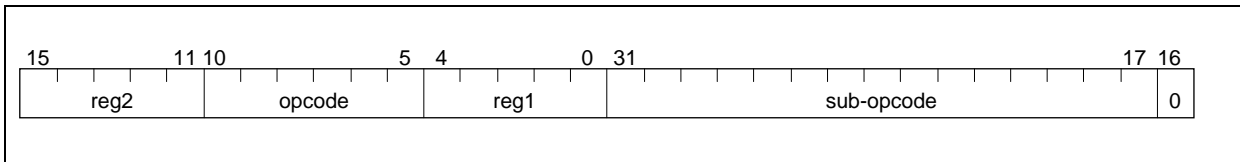
A 32-bit instruction format consists of a 6-bit opcode field, 2-bit sub-opcode field, 3-bit bit specification field, a general-purpose register specification field, and a 16-bit displacement field.



(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.

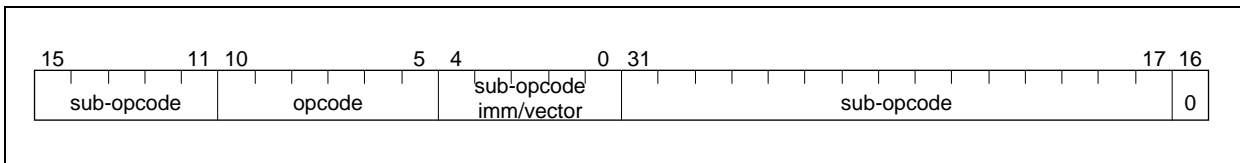
Caution **Extended instruction format 1 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction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

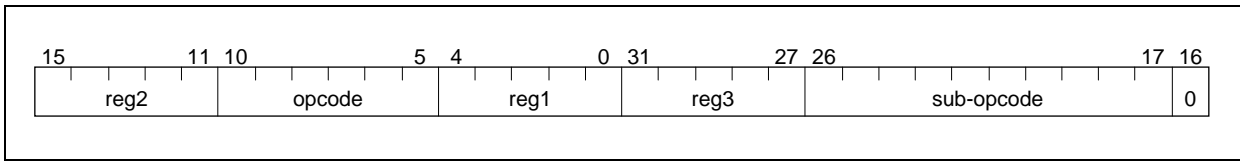
Caution **Extended instruction format 2 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction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

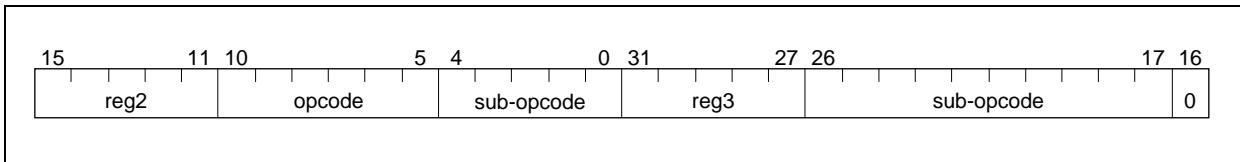
Caution **Extended instruction format 3 may use part of the general-purpose register specification field or the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction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

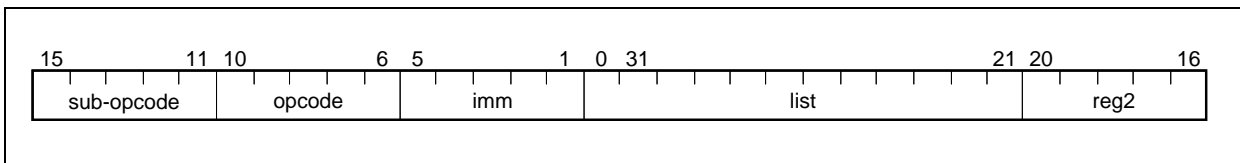
Caution **Extended instruction format 4 may use part of the general-purpose register specification field of the sub-opcode field as a system register number field, condition code field, immediate field, or displacement field. For details, refer to the description of each instruction in 5.3 Instruction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



(13) Stack manipulation instruction format (Format XIII)

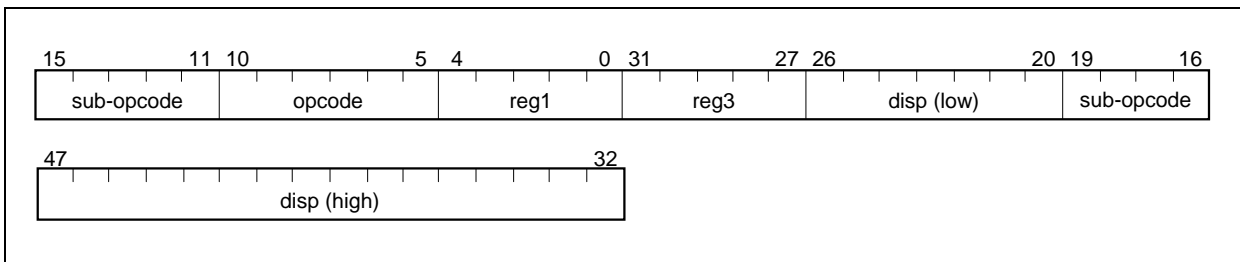
A 32-bit instruction format consists of a 5-bit opcode field, 5-bit immediate field, 12-bit register list field, 5-bit sub-opcode field, and one general-purpose register specification field (or 5-bit sub-opcode field).

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



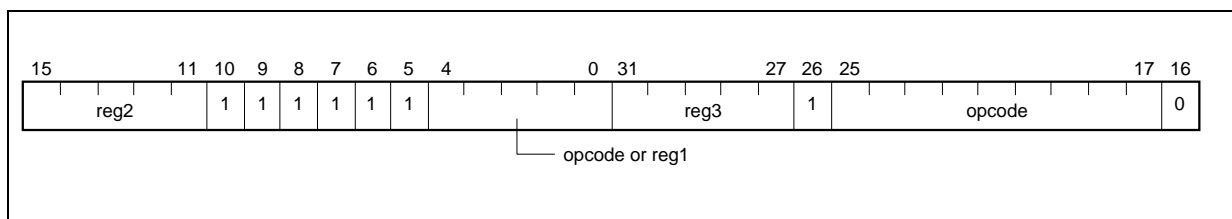
(14) Load/store instruction 48-bit format (Format XIV)

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.

Coprocessor instructions are not defined in the V850E2S CPU.

(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**.

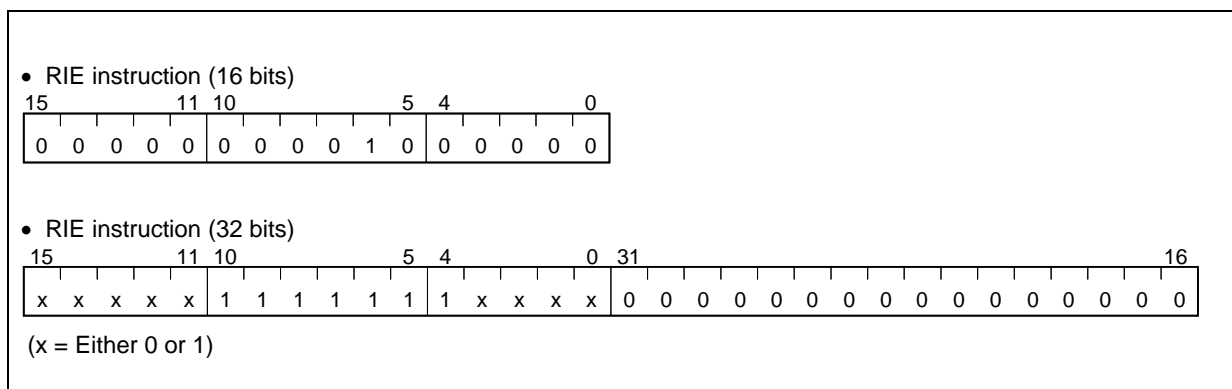
5.1.3 Reserved instructions

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 V850E2S 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 advance 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 _x	_x -bit immediate data
disp _x	_x -bit displacement data
regID	System register number
vector _x	Data to specify vector (_x 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 $n \geq 7FFFFFFFH$, $n = 7FFFFFFFH$. If $n \leq 80000000H$, $n = 80000000H$.
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
l	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	$(CY \text{ or } Z) = 1$
1011	H	$(CY \text{ or } Z) = 0$
0100	S/N	$S = 1$
1100	NS/P	$S = 0$
0101	T	always (Unconditional)
1101	SA	$SAT = 1$
0110	LT	$(S \text{ xor } OV) = 1$
1110	GE	$(S \text{ xor } OV) = 0$
0111	LE	$((S \text{ xor } OV) \text{ or } Z) = 1$
1111	GT	$((S \text{ xor } OV) \text{ or } Z) = 0$

<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)	rrrrr010010iiiiii	

[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	iiiiiiiiiiiiiiiiiii	

[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
	r r r r r 0 0 1 0 1 0 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] 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	iiiiiiiiiiiiiiiiiii
------------------	---------------------

[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
d1011dddcccc

d1011ddd 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	$(S \text{ xor } OV) = 0$	Greater than or equal to signed
	BGT	1111	$((S \text{ xor } OV) \text{ or } Z) = 0$	Greater than signed
	BLE	0111	$((S \text{ xor } OV) \text{ or } Z) = 1$	Less than or equal to signed
	BLT	0110	$(S \text{ xor } OV) = 1$	Less than signed
Unsigned integer	BH	1011	$(CY \text{ or } Z) = 0$	Higher (Greater than)
	BL	0001	$CY = 1$	Lower (Less than)
	BNH	0011	$(CY \text{ or } Z) = 1$	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 ($0 \rightarrow 1$ or $1 \rightarrow 0$). 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>

<p>BSH</p>	<p>Byte swap halfword</p> <p>Byte swap of halfword data</p>
-------------------	---

[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
rrrrr111111100000	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) 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	031	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) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, bit\#3))$
 $token \leftarrow \text{clear-bit} (token, bit\#3)$
 $\text{Store-memory} (adr, token, \text{Byte})$
 (2) $adr \leftarrow GR [reg1]$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, reg2))$
 $token \leftarrow \text{clear-bit} (token, reg2)$
 $\text{Store-memory} (adr, token, \text{Byte})$

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

[Opcode] (1)

	15	0	31		16
10bbb111110RRRRR	dddddddddddddddd				

 (2)

	15	0	31		16
rrrrr11111RRRRR	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]

15	0 31	16
(1) rrrrr111111RRRRR	www011001cccc0	

15	0 31	16
(2) rrrrr111111iiii	www011000cccc0	

[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 \leftarrow GR [reg2] – GR [reg1]
 (2) result \leftarrow GR [reg2] – sign-extend (imm5)

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

[Opcode]

	15	0
(1)	rrrrr001111RRRRR	
	15	0
(2)	rrrrr010011iiiiii	

[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

00000111111100000	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

00000111111100000	00000000101100000
-------------------	-------------------

[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>

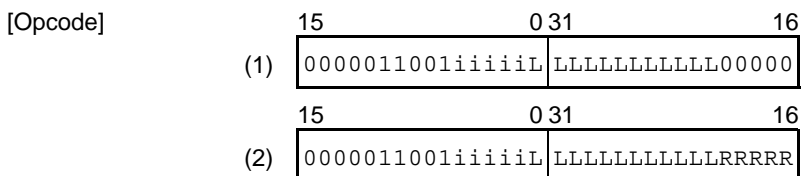
DISPOSE	Function dispose
	Stack frame deletion

- [Instruction format] (1) DISPOSE imm5, list12
 (2) DISPOSE imm5, list12, [reg1]

- [Operation] (1) $adr \leftarrow sp + \text{zero-extend}(\text{imm5 logically shift left by } 2)$
 foreach (all regs in list12) {
 $GR[\text{reg in list12}] \leftarrow \text{Load-memory}(adr, \text{Word})^{\text{Note}}$
 $adr \leftarrow adr + 4$
 }
 $sp \leftarrow adr$
 (2) $adr \leftarrow sp + \text{zero-extend}(\text{imm5 logically shift left by } 2)$
 foreach (all regs in list12) {
 $GR[\text{reg in list12}] \leftarrow \text{Load-memory}(adr, \text{Word})^{\text{Note}}$
 $adr \leftarrow adr + 4$
 }
 $sp \leftarrow adr$
 $PC \leftarrow GR[\text{reg1}]$

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

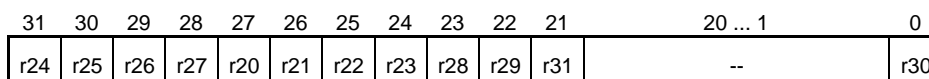
[Format] Format XIII



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

The values of LLLLLLLLLLLL are the corresponding bit values shown in register list “list12” (for example, the “L” at bit 21 of the opcode corresponds to the value of bit21 in list12).

list12 is a 32-bit register list, defined as follows.



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": 081FFFFFFH

[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	wwwww0101100000
------------------	-----------------

[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>

<p style="font-size: 2em; margin: 0;">DIVH</p>	<p style="margin: 0;">Divide halfword</p> <p style="margin: 0;">Division of (signed) halfword data</p>
--	--

[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.

- | |
|---|
| <p>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.</p> <p>2. Do not specify r0 as reg1 and reg2 for DIVH reg1 and reg2 in instruction format (1).</p> |
|---|

<Divide instruction>

<p style="font-size: 2em; margin: 0;">DIVHU</p>	<p style="margin: 0;">Divide halfword unsigned</p> <p style="margin: 0;">Division of (unsigned) halfword data</p>
---	---

[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>

<p>DIVQ</p>	<p>Divide word quickly</p> <p>Division of (signed) word data (variable steps)</p>
--------------------	---

[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>

<div data-bbox="169 302 282 338" data-label="Text"> <p>DIVQU</p> </div>	<div data-bbox="1080 266 1372 295" data-label="Text"> <p>Divide word unsigned quickly</p> </div> <div data-bbox="900 358 1372 389" data-label="Text"> <p>Division of (unsigned) word data (variable steps)</p> </div>
--	---

[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
10000111111100000	00000000101100000	

[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

00000111111100000	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

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 \leftarrow PC + 2 (return PC)
 FEPSW \leftarrow PSW
 ECR.FECC \leftarrow exception code (31H-3FH)
 FEIC \leftarrow exception code (31H-3FH)
 PSW.EP \leftarrow 1
 PSW.ID \leftarrow 1
 PSW.NP \leftarrow 1

If (MPM.AUE==1) is satisfied
 then PSW.IMP \leftarrow 0
 PSW.DMP \leftarrow 0
 PSW.NPV \leftarrow 0

PC \leftarrow 00000030H

[Format] Format I

[Opcode]

15	0
0vvvvv00001000000	

Where vvvv is vector4.

Do not set 0H to vector4 (vvvv \neq 0000).

[Flags]

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

[Description]

Saves the contents of the return PC (address of the instruction next to the FETRAP instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores 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.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
00000111111100000	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 INT127)
- FE level maskable interrupt input (FEINT)
- FE level non-maskable interrupt input (FENMI)
- 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 INT127, 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>

<p>HSH</p>	<p>Halfword swap halfword</p> <p>Halfword swap of halfword data</p>
------------	---

[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)

15	0	31	16
rrrrrr	11110	ddddddd	ddddddddddddddd0

 dddddddddddddddddddd is the higher 21 bits of disp22.
 rrrrrr ≠ 00000 (Do not specify r0 for reg2.)
- (2)

15	0	31	16	47	32
00000010111	RRRRR	ddddddddddddddd0	DDDDDDDDDDDDDDDD		

 DDDDDDDDDDDDDDDDDddddddddddddddd 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.
----------------	--

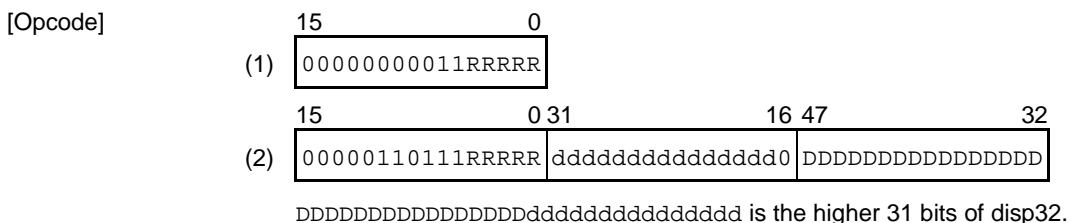
<Branch instruction>



[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



[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
0000011110	ddddddd	ddddddddddddddd0

ddddddddddddddddddd is the higher 21 bits of disp22.

(2)

15	0 31	16 47	32
000001011100000	ddddddddddddddd0	DDDDDDDDDDDDDDDD	DDDDDDDDDDDDDDDD

DDDDDDDDDDDDDDDD 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.

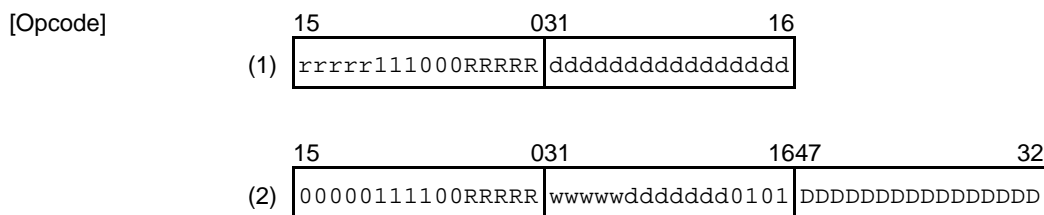
<Load instruction>



- [Instruction format] (1) LD.B disp16 [reg1] , reg2
 (2) LD.B disp23 [reg1] , reg3

- [Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $GR [reg2] \leftarrow \text{sign-extend} (\text{Load-memory} (adr, \text{Byte}))$
 (2) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp23)$
 $GR [reg3] \leftarrow \text{sign-extend} (\text{Load-memory} (adr, \text{Byte}))$

- [Format] (1) Format VII
 (2) Format XIV



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

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

- [Description] (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, 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) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $GR [reg2] \leftarrow \text{zero-extend} (\text{Load-memory} (adr, \text{Byte}))$
 (2) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp23)$
 $GR [reg3] \leftarrow \text{zero-extend} (\text{Load-memory} (adr, \text{Byte}))$

[Format] (1) Format VII
(2) Format XIV

[Opcode]

(1)

15			031		16
rrrrr	11110	b	RRRRR	d	d

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

(2)

15			031		1647	32
00000	11110	1	RRRRR	www	w	d

 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, 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.

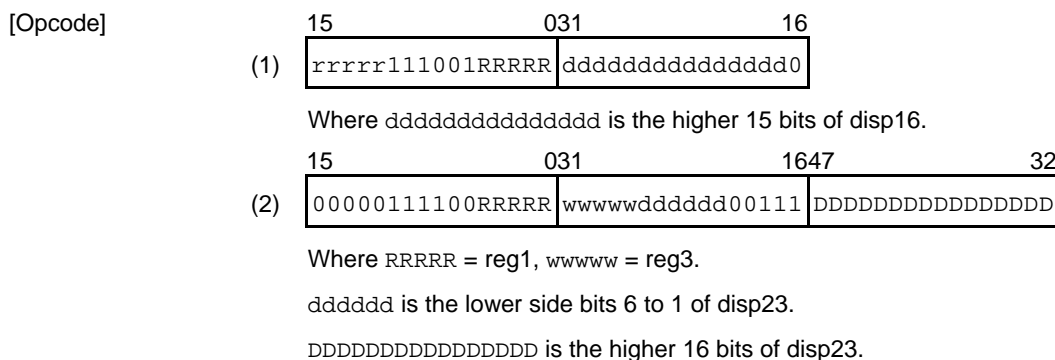
<Load instruction>



- [Instruction format] (1) LD.H disp16 [reg1] , reg2
 (2) LD.H disp23 [reg1] , reg3

- [Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $GR [reg2] \leftarrow \text{sign-extend} (\text{Load-memory} (adr, \text{Halfword}))$
 (2) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp23)$
 $GR [reg3] \leftarrow \text{sign-extend} (\text{Load-memory} (adr, \text{Halfword}))$

- [Format] (1) Format VII
 (2) Format XIV



- [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) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $GR [reg2] \leftarrow \text{zero-extend} (\text{Load-memory} (adr, \text{Halfword}))$
 (2) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp23)$
 $GR [reg3] \leftarrow \text{zero-extend} (\text{Load-memory} (adr, \text{Halfword}))$

- [Format] (1) Format VII
 (2) Format XIV

- [Opcode]
- | | | | |
|-----|-----------------|-----------------|----|
| | 15 | 031 | 16 |
| (1) | rrrrr11111RRRRR | dddddddddddddd1 | |
- Where ddddddddddddddd is the higher 15 bits of disp16.
 rrrrr ≠ 00000 (Do not specify r0 for reg2.)
- | | | | | |
|-----|------------------|---------------|------------------|----|
| | 15 | 031 | 1647 | 32 |
| (2) | 00000111101RRRRR | wwwwdddd00111 | DDDDDDDDDDDDDDDD | |
- Where RRRRR = reg1, wwwww = reg3.
 ddddd is the lower side bits 6 to1 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. 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.

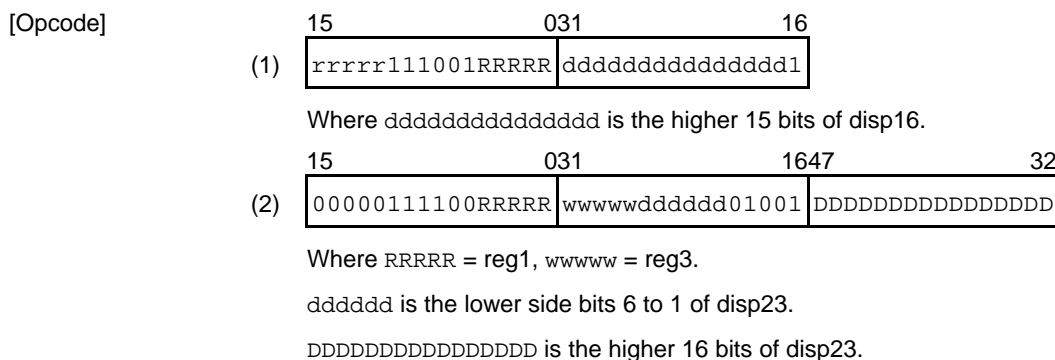
<Load instruction>



- [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



- [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	www0011110	mmmm0

[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
rrrrr	111111	RRRRR	www00	11111mmmm0

[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]
- (1)

15	0
rrrrr00000RRRRR	

 rrrrr ≠ 00000 (Do not specify r0 for reg2.)
 - (2)

15	0
rrrrr010000iiii	

 rrrrr ≠ 00000 (Do not specify r0 for reg2.)
 - (3)

15	0	31	16	47	32
00000110001RRRRR	iiiiiiiiiiiiiiii	IIIIIIIIIIIIIIII			

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

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

- [Description]
- (1) Copies and transfers the word data of general-purpose register reg1 to general-purpose register reg2. General-purpose register reg1 is not affected.
 - (2) Copies and transfers the 5-bit immediate data, sign-extended to word length, to general-purpose register reg2.
 - (3) Copies and transfers the 32-bit immediate data to general-purpose register reg1.

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

<Arithmetic instruction>

MOVEA

Move effective address

Effective address transfer

[Instruction format] MOVEA imm16, reg1, reg2

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

[Format] Format VI

[Opcode]

15	0 31	16
rrrrr110001RRRRR	iiiiiiiiiiiiiiiiiii	

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 || 0¹⁶)

[Format] Format VI

[Opcode]

15	0 31	16
rrrrr110010RRRRR	iiiiiiiiiiiiiiiiiii	

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>

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

[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)	rrrrr11111RRRRR	wwwww01000100000

15	0 31	16
(2)	rrrrr11111iiiiii	wwwww01001IIII00

iiiiii are the lower 5 bits of 9-bit immediate data.
 IIIII 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) \leftarrow GR [reg2] (16) \times GR [reg1] (16)
 (2) GR [reg2] \leftarrow GR [reg2] \times sign-extend (imm5)

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

[Opcode]

(1)

15	0
rrrrr	000111RRRRR

 rrrrr \neq 00000 (Do not specify r0 for reg2.)

(2)

15	0
rrrrr	010111iiiiii

 rrrrr \neq 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	iiiiiiiiiiiiiiiiiii	

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>

<p>MULU</p>	<p>Multiply word unsigned by register/immediate (9-bit)</p> <p>Multiplication of (unsigned) word data</p>
-------------	---

[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) rrrrr11111RRRRR	wwwww01000100010	

15	0 31	16
(2) rrrrr11111iiii	wwwww01001IIII10	

iiii 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
000000000000000000

[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
rrrrr	000001RRRRR

[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.

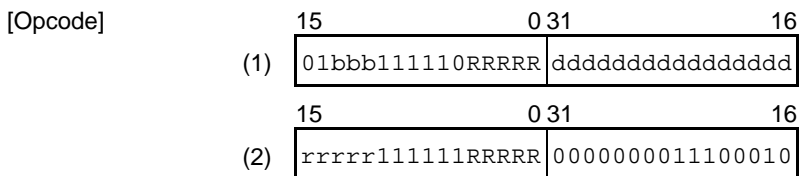
<Bit manipulation instruction>

NOT1	NOT bit
	NOT bit

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

[Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, bit\#3))$
 $token \leftarrow \text{not-bit} (token, bit\#3)$
 $\text{Store-memory} (adr, token, \text{Byte})$
 (2) $adr \leftarrow GR [reg1]$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, reg2))$
 $token \leftarrow \text{not-bit} (token, reg2)$
 $\text{Store-memory} (adr, token, \text{Byte})$

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



[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

r r r r r 0 0 1 0 0 0 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] 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.

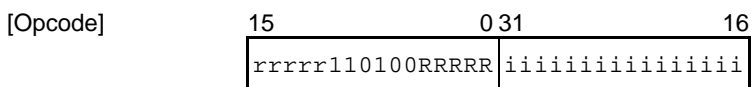
<Logical instruction>



[Instruction format] ORI imm16, reg1, reg2

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

[Format] Format VI



[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>

<p style="font-size: 24px; margin: 0;">PREPARE</p>	<p style="font-size: 12px; margin: 0;">Function prepare</p> <p style="font-size: 12px; margin: 0;">Create stack frame</p>
--	---

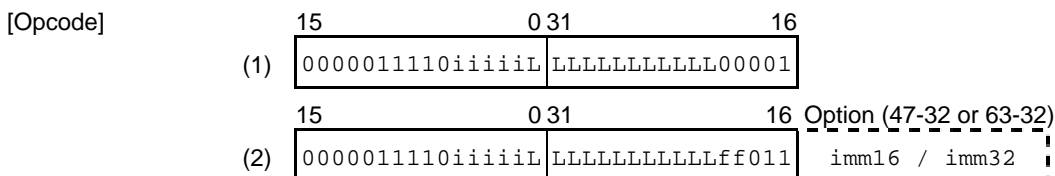
- [Instruction format]
- (1) PREPARE list12, imm5
 - (2) PREPARE list12, imm5, sp/imm^{Note}

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

- [Operation]
- (1) $adr \leftarrow sp$
 foreach (all regs in list12) {
 $adr \leftarrow adr - 4$
 Store-memory (adr, GR[reg in list12], Word)^{Note}
 }
 $sp \leftarrow adr - \text{zero-extend}(\text{imm5 logically shift left by 2})$
 - (2) $adr \leftarrow sp$
 foreach (all regs in list12) {
 $adr \leftarrow adr - 4$
 Store-memory (adr, GR[reg in list12], Word)^{Note}
 }
 $sp \leftarrow adr - \text{zero-extend}(\text{imm5 logically shift left by 2})$
- case
- ff = 00: $ep \leftarrow sp$
 - ff = 01: $ep \leftarrow \text{sign-extend}(\text{imm16})$
 - ff = 10: $ep \leftarrow \text{imm16 logically shift left by 16}$
 - ff = 11: $ep \leftarrow \text{imm32}$

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

[Format] Format XIII



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

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

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

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

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

- When all of the register’s non-corresponding bits are “0”: 0800001H
- When all of the register’s non-corresponding bits are “1”: 081FFFFFFH

[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>

<p>RETI</p>	<p>Return from trap or interrupt</p> <p>Return from EI level software exception or interrupt</p>
-------------	--

[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
00000111111100000	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 (TRAP): 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
 PC ← 00000030H

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

[Opcode] (1)

15	0
00000000001000000	

(2)

15	031	16
iiiiii11111111IIII 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.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]
- | | | | |
|-----|-----------------|------|------------------|
| | 15 | 0 31 | 16 |
| (1) | rrrrr11111RRRRR | | 0000000010100000 |
| | 15 | 0 | |
| (2) | rrrrr010101iiii | | |
| | 15 | 0 31 | 16 |
| (3) | rrrrr11111RRRRR | | www00010100010 |

- [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.

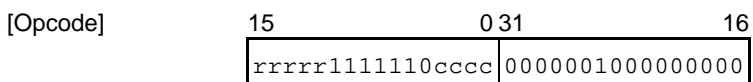
<Data manipulation instruction>

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



[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>

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

[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
rrrrrr000110RRRRR	

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

(2)

15	0
rrrrrr010001iiiiii	

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

(3)

15	0	31	16
rrrrrr111111RRRRR	wwwww011110111010		

[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]

(1)

15	0
rrrrr000101RRRRR	

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

(2)

15	0	31	16
rrrrr111111RRRRR	www	ww	01110011010

[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	<ol style="list-style-type: none"> 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	iiiiiiiiiiiiiiiiiii	

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 | <ol style="list-style-type: none"> 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.

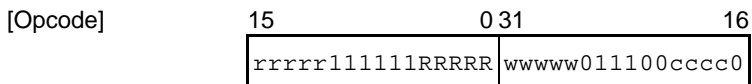
<Conditional operation instructions>

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



- [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
rrrrr	11111100000	www01101100100

[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	www01101100000	

[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	www01101100010	

[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).

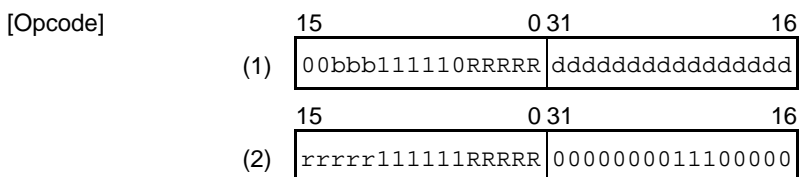
<Bit manipulation instruction>



- [Instruction format] (1) SET1 bit#3, disp16 [reg1]
 (2) SET1 reg2, [reg1]

- [Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 token \leftarrow Load-memory (adr, Byte)
 Z flag \leftarrow Not (extract-bit (token, bit#3))
 token \leftarrow set-bit (token, bit#3)
 Store-memory (adr, token, Byte)
- (2) $adr \leftarrow GR [reg1]$
 token \leftarrow Load-memory (adr, Byte)
 Z flag \leftarrow Not (extract-bit (token, reg2))
 token \leftarrow set-bit (token, reg2)
 Store-memory (adr, token, Byte)

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



- [Flags] CY --
 OV --
 S --
 Z "1" if bit specified by operand = "0", "0" if bit specified by operand = "1".
 SAT --

- [Description] (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Byte data is read from the generated address, the bits indicated by the 3-bit bit number are set (1) and the data is written back to the original address.
 If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".
- (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Byte data is read from the generated address, the lower 3 bits indicated of general-purpose register reg2 are set (1) and the data is written back to the original address.
 If the specified bit of the read byte data is "0", the Z flag is set to "1", and if the specified bit is "1", the Z flag is cleared to "0".

[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) | <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: right; padding-right: 5px;">15</td> <td style="text-align: center; padding: 0 10px;">0 31</td> <td style="text-align: left; padding-left: 5px;">16</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">rrrrr111111RRRRR</td> <td style="border: 1px solid black; padding: 2px;">0000000011000000</td> <td></td> </tr> </table> | 15 | 0 31 | 16 | rrrrr111111RRRRR | 0000000011000000 | |
| 15 | 0 31 | 16 | | | | | |
| rrrrr111111RRRRR | 0000000011000000 | | | | | | |
| (2) | <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: right; padding-right: 5px;">15</td> <td style="text-align: center; padding: 0 10px;">0</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">rrrrr010110iiii</td> <td></td> </tr> </table> | 15 | 0 | rrrrr010110iiii | | | |
| 15 | 0 | | | | | | |
| rrrrr010110iiii | | | | | | | |
| (3) | <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: right; padding-right: 5px;">15</td> <td style="text-align: center; padding: 0 10px;">0 31</td> <td style="text-align: left; padding-left: 5px;">16</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">rrrrr111111RRRRR</td> <td style="border: 1px solid black; padding: 2px;">www00011000010</td> <td></td> </tr> </table> | 15 | 0 31 | 16 | rrrrr111111RRRRR | www00011000010 | |
| 15 | 0 31 | 16 | | | | | |
| rrrrr111111RRRRR | www00011000010 | | | | | | |

- [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]
- | | | | |
|-----|----------------------------------|------|----|
| | 15 | 0 31 | 16 |
| (1) | rrrrr11111RRRRR 0000000010000000 | | |
| | 15 | 0 | |
| (2) | rrrrr010100iiii | | |
| | 15 | 0 31 | 16 |
| (3) | rrrrr11111RRRRR 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.

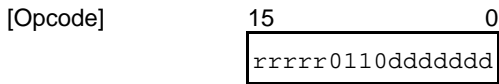
<Load instruction>



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

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

[Format] Format IV



[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] $adr \leftarrow ep + \text{zero-extend}(\text{disp8})$
 $GR[\text{reg2}] \leftarrow \text{sign-extend}(\text{Load-memory}(adr, \text{Halfword}))$

[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] $adr \leftarrow ep + \text{zero-extend}(\text{disp5})$
 $GR[\text{reg2}] \leftarrow \text{zero-extend}(\text{Load-memory}(adr, \text{Halfword}))$

[Format] Format IV

[Opcode] $\begin{matrix} 15 & & 0 \\ \boxed{rrrrr0000111dddd} \end{matrix}$
 $rrrrr \neq 00000$ (Do not specify r0 for reg2.)
 dddd is the higher 4 bits of disp5.

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

[Description] Adds the element pointer to the 5-bit displacement data, zero-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address, zero-extended to word length, and stored in general-purpose register reg2.

Caution Do not specify r0 for reg2.

<Load instruction>



[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.

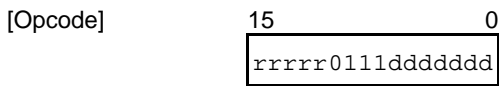
<Store instruction>



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

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

[Format] Format IV



[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>



[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>



[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.

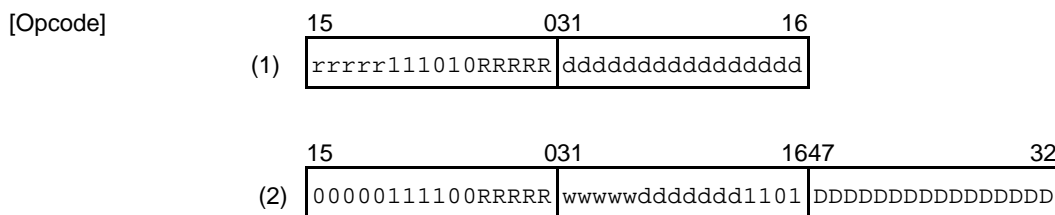
<Store instruction>



- [Instruction format] (1) ST.B reg2, disp16 [reg1]
 (2) ST.B reg3, disp23 [reg1]

- [Operation] (1) $adr \leftarrow GR [reg1] + sign\text{-}extend (disp16)$
 Store-memory (adr, GR [reg2], Byte)
 (2) $adr \leftarrow GR [reg1] + sign\text{-}extend (disp23)$
 Store-memory (adr, GR [reg3], Byte)

- [Format] (1) Format VII
 (2) Format XIV



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 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.

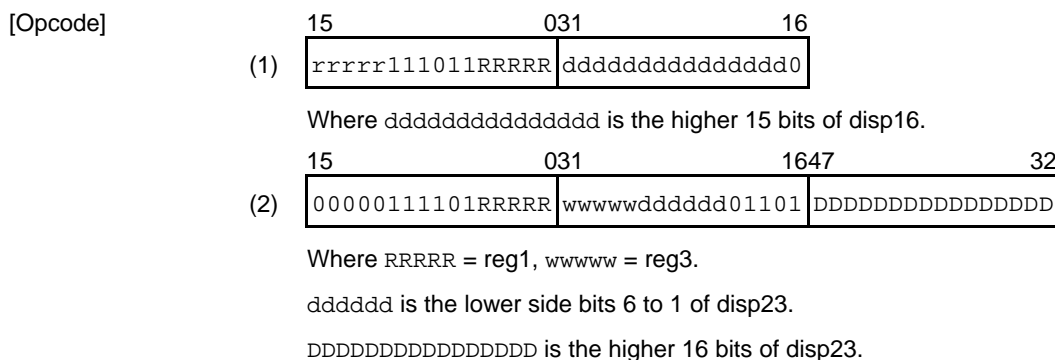
<Store instruction>



- [Instruction format] (1) ST.H reg2, disp16 [reg1]
 (2) ST.H reg3, disp23 [reg1]

- [Operation] (1) $adr \leftarrow GR [reg1] + sign\text{-}extend (disp16)$
 Store-memory (adr, GR [reg2], Halfword)
 (2) $adr \leftarrow GR [reg1] + sign\text{-}extend (disp23)$
 Store-memory (adr, GR [reg3], Halfword)

- [Format] (1) Format VII
 (2) Format XIV



- [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.

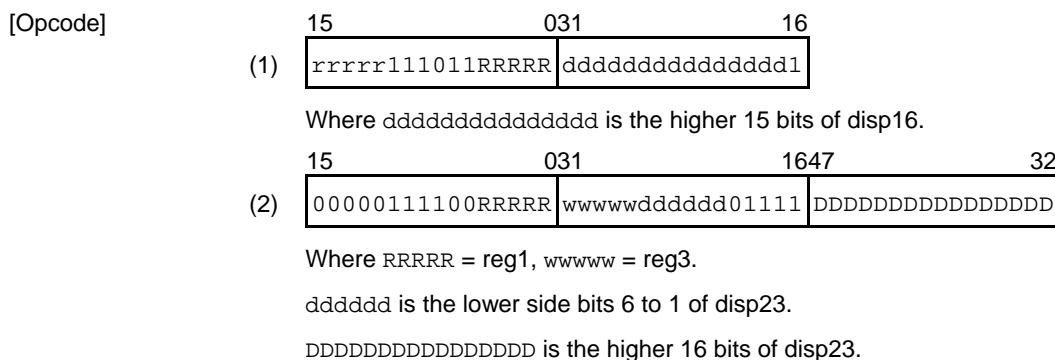
<Store instruction>



- [Instruction format] (1) ST.W reg2, disp16 [reg1]
 (2) ST.W reg3, disp23 [reg1]

- [Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 Store-memory (adr, GR [reg2], Word)
 (2) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp23)$
 Store-memory (adr, GR [reg3], Word)

- [Format] (1) Format VII
 (2) Format XIV



- [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
rrrrr	111111RRRRR	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.
----------------	--

<Special instruction>

SWITCH	Jump with table look up
	Jump with table look up

[Instruction format] SWITCH reg1

[Operation] $adr \leftarrow (PC + 2) + (GR [reg1] \text{ logically shift left by } 1)$
 $PC \leftarrow (PC + 2) + (\text{sign-extend}(\text{Load-memory}(adr, \text{Halfword})) \text{ logically shift left by } 1)$

[Format] Format I

[Opcode] $\begin{matrix} 15 & & 0 \\ \boxed{00000000010RRRRR} \end{matrix}$
 RRRRR \neq 00000 (Do not specify r0 for reg1.)

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

[Description] The following steps are taken.

- (1) Adds the start address (the one subsequent to the SWITCH instruction) to general-purpose register reg1, logically left-shifted by 1, to generate a 32-bit table entry address.
- (2) Loads the halfword entry data indicated by the address generated in step (1).
- (3) Adds the table start address after sign-extending the loaded halfword data and logically left-shifting it by 1 (the one subsequent to the SWITCH instruction) to generate a 32-bit target address.
- (4) Jumps to the target address generated in step (3).

- Cautions**
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) 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>

<p>SYNCE</p>	<p>Synchronize exceptions</p> <p>Exception synchronization instruction</p>
--------------	--

[Instruction format] SYNCE

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

[Format] Format I

[Opcode] 15 0
00000000000011101

[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 that are generated because of the preceding instructions are always acknowledged before execution of this instruction is completed. However, imprecise exceptions are not generated in the V850E2S CPU. Though the SYNCE instruction is supported, since the CPU does not have any exception causes that are kept waiting for the execution, it is replaced by the SYNCM instruction when executed.

<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>

<p>SYNCP</p>	<p>Synchronize pipeline</p> <p>Pipeline synchronize instruction</p>
--------------	---

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

[Format] Format X

[Opcode]

15	0 31	16
110101111111vvvvv00vvv00101100000		

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.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) 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 \leftarrow PC + 4 (return PC)
 EIPSW \leftarrow PSW
 ECR.EICC \leftarrow exception code (40H to 5FH)
 EIIC \leftarrow exception code (40H to 5FH)
 PSW.EP \leftarrow 1
 PSW.ID \leftarrow 1
 If (MPM.AUE==1) is satisfied
 then PSW.IMP \leftarrow 0
 PSW.DMP \leftarrow 0
 PSW.NPV \leftarrow 0
 PC \leftarrow 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
000001111111vvvvvv	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.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.

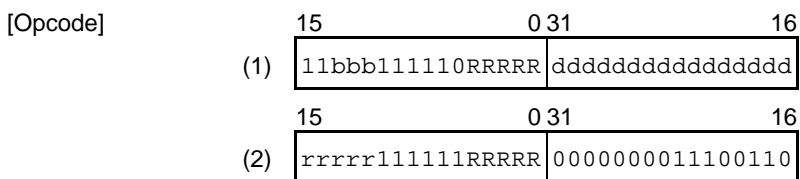
<Bit manipulation instruction>

TST1	Test bit
	Bit test

- [Instruction format] (1) TST1 bit#3, disp16 [reg1]
 (2) TST1 reg2, [reg1]

- [Operation] (1) $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, \text{bit}\#3))$
 (2) $adr \leftarrow GR [reg1]$
 $token \leftarrow \text{Load-memory} (adr, \text{Byte})$
 $Z \text{ flag} \leftarrow \text{Not} (\text{extract-bit} (token, reg2))$

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



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

<Logical instruction>

XOR

Exclusive OR

Exclusive OR

[Instruction format] XOR reg1, reg2

[Operation] GR [reg2] ← GR [reg2] XOR GR [reg1]

[Format] Format I

[Opcode] 15 0

rrrrr001001RRRRR

[Flags] CY --
 OV 0
 S "1" if operation result word data MSB is "1"; otherwise, "0".
 Z "1" if the operation result is "0"; otherwise, "0".
 SAT --

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

<Logical instruction>

XORI

Exclusive OR immediate (16-bit)

Exclusive OR immediate

[Instruction format] XORI imm16, reg1, reg2

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

[Format] Format VI

15	0 31	16
<div style="border: 1px solid black; padding: 2px; display: inline-block;"> rrrrrr110101RRRRR iiiiiiiiiiiiiiiii </div>		

[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>



[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**).

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 V850E2S 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 (x: 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	x	x	None	None	+0000H	0	0	0	1	None
FE level non-maskable interrupt	FENMI	FENMI input ^{Note 4}	3	FE	Interrupt	NG	NG	x	x	0000020H	currentPC	+0020H	Note 5	1	0	1	FERET
System error exception	SYSERR	SYSERR input (4 causes)	4	FE	Note 6	NG	NG	x	x	00000230H : 00000233H	currentPC	+0030H	Note 5	1	1	1	FERET
FE level maskable interrupt	FEINT	FEINT input ^{Note 4}	7	FE	Interrupt	OK	OK	x	0	0000010H	currentPC	+0010H	Note 5	1	0	1	FERET
EI level maskable interrupt	INT	INTn input ^{Note 4} (n = 0 to 255)	9	EI	Interrupt	OK	OK	0	0	0000080H : 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, depending 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 (x: 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}	x	x	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}	x	x	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}	x	x	00000431H	currentPC	+0030H	0	1	1	1	FERET
Coprocessor unusable exception	UCPOP	Coprocessor instruction		FE	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	00000530H : 00000537H	currentPC	+0030H	Note 6	1	1	1	FERET
Reserved instruction exception	RIE	Reserved instruction		FE	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	00000130H	currentPC	+0030H	Note 6	1	1	1	FERET
FE level software exception	FETRAP	FETRAP instruction (vector = 1H to FH)		FE	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	00000031H : 0000003FH	nextPC	+0030H	Note 6	1	1	1	FERET
EI level software exception	TRAP	TRAP0n instruction (vector = 00 to 0FH)		EI	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	00000040H : 0000004FH	nextPC	+0040H	Note 6	s	1	1	EIRET
EI level software exception	TRAP	TRAP1n instruction (vector = 10H to 1FH)		EI	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	00000050H : 0000005FH	nextPC	+0050H	Note 6	s	1	1	EIRET
System call exception	SYSCALL	SYSCALL instruction (vector = 00H to FFH)		EI	Precise	OK ^{Note 4}	OK ^{Note 4}	x	x	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 V850E2S CPU classifies exceptions into the following three types by their timing of generation and characteristics.

- Precise 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
- 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) 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.)

(3) 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.

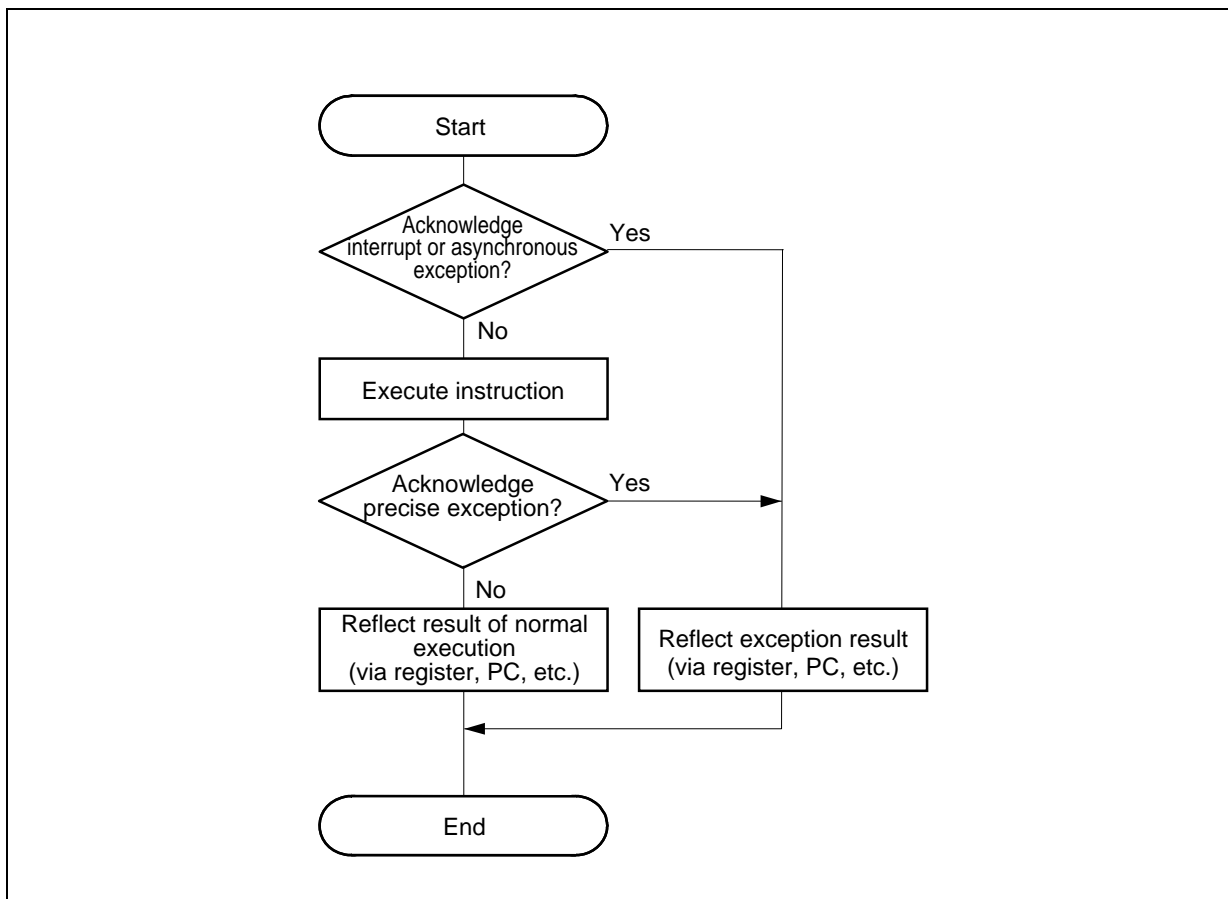
- FE level non-maskable interrupt
- FE level maskable interrupt
- EI level maskable interrupt

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 INT127, 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 and asynchronous exception is decided before an instruction is executed. If the exception can be acknowledged, processing branches to exception processing. After exception handling, 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   Low	CPU initialization (RESET)	Before instruction execution
	FE level non-maskable interrupt (FENMI)	
	System error exception (SYSERR)	
	FE level maskable interrupt (FEINT)	
	EI level maskable interrupt (INT)	
	Execution protection exception (MIP)	After instruction execution
	Memory error exception (MEP)	
	Data protection exception (MDP) ^{Note}	
	Coprocessor unusable exception (UCPOP) ^{Note}	
	Reserved instruction exception (RIE) ^{Note}	
FE level software exception (FETRAP) ^{Note}		
EI level software exception (TRAP) ^{Note}		
System call exception (SYSCALL) ^{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 V850E2S 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 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 Coprocessor unusable exception	Debug exception ^{Note}

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.

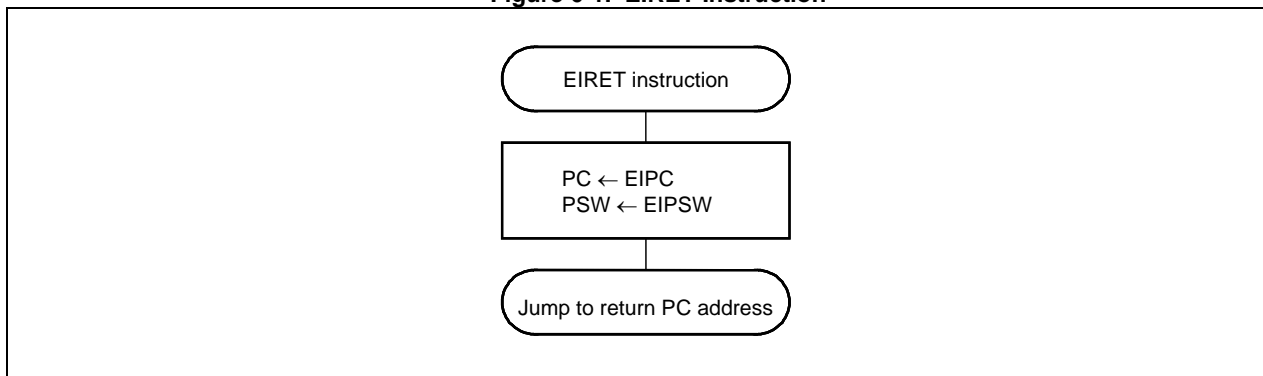
<1> Return PC and PSW are loaded from the EIPC and EIPSW registers.

<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

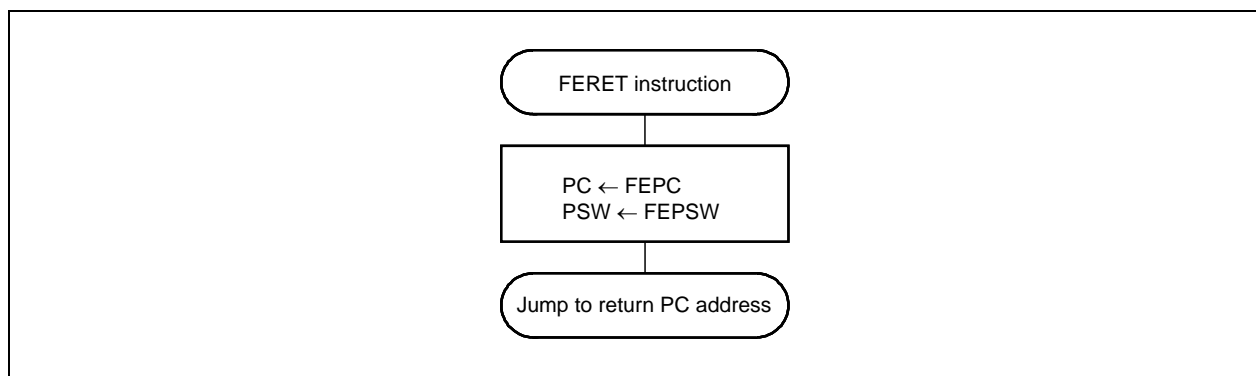
To return from FE level exception processing, execute the FERET instruction.

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 (TRAP)

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 (TRAP).

Execute the RETI instruction to return from an interrupt or EI level software exception (TRAP).
 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.

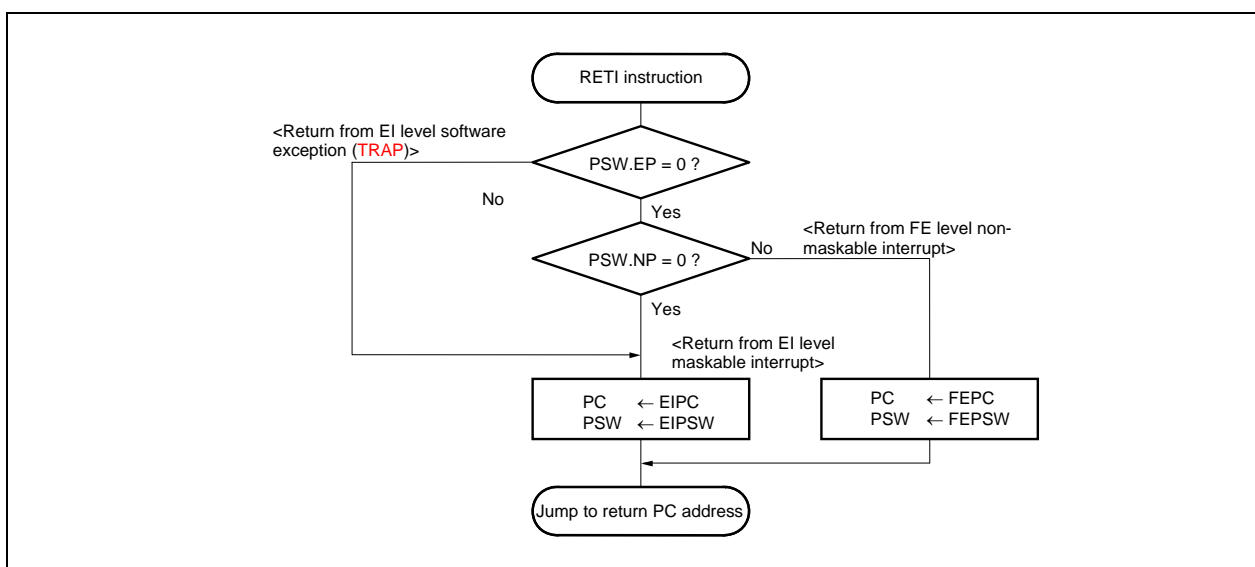
When returning from any type of exception processing, the LDSR instruction must be used just before the RETI instruction to correctly restore the PC and PSW, and the flags for the PSW.NP and PSW.EP bits must be set to the following status.

- 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 (TRAP) 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.NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.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.

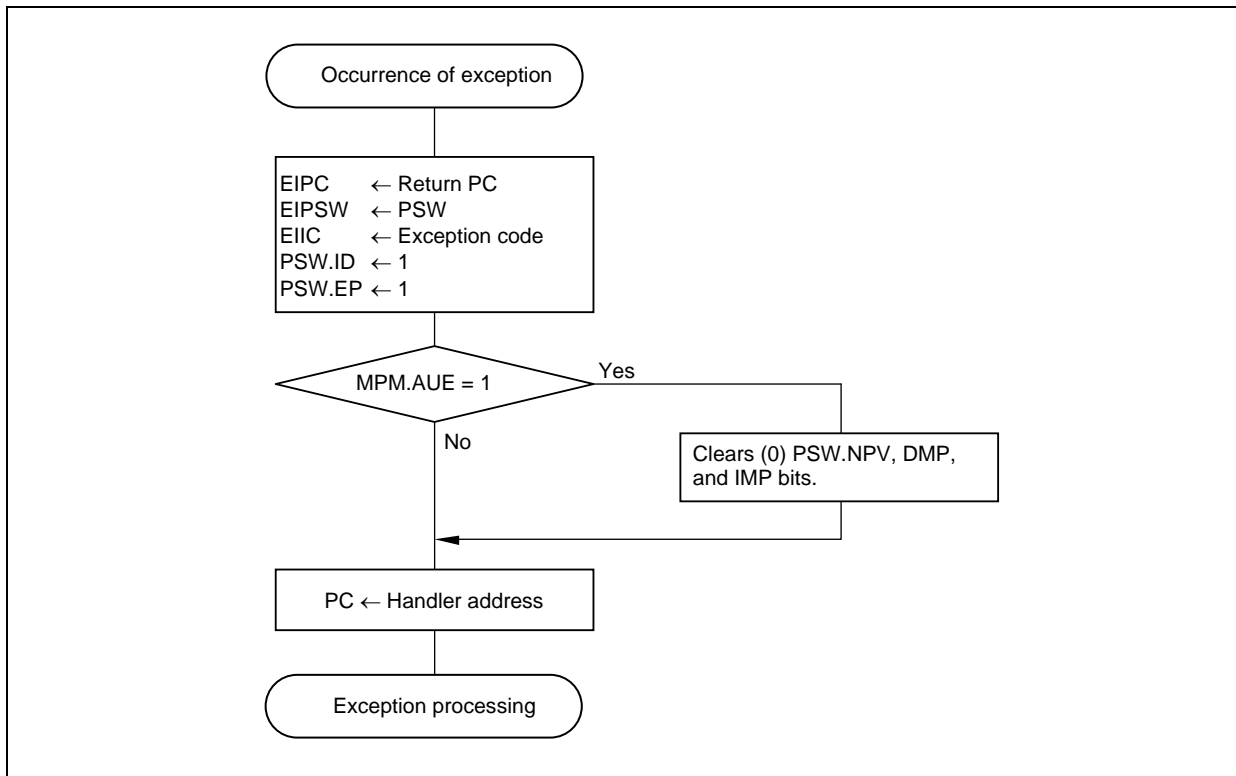
Note Although the exception code is also written to the lower 16 bits (EICC) of the ECR register, the EIIC register should be used except when using an existing program that cannot be revised.

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.NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.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.

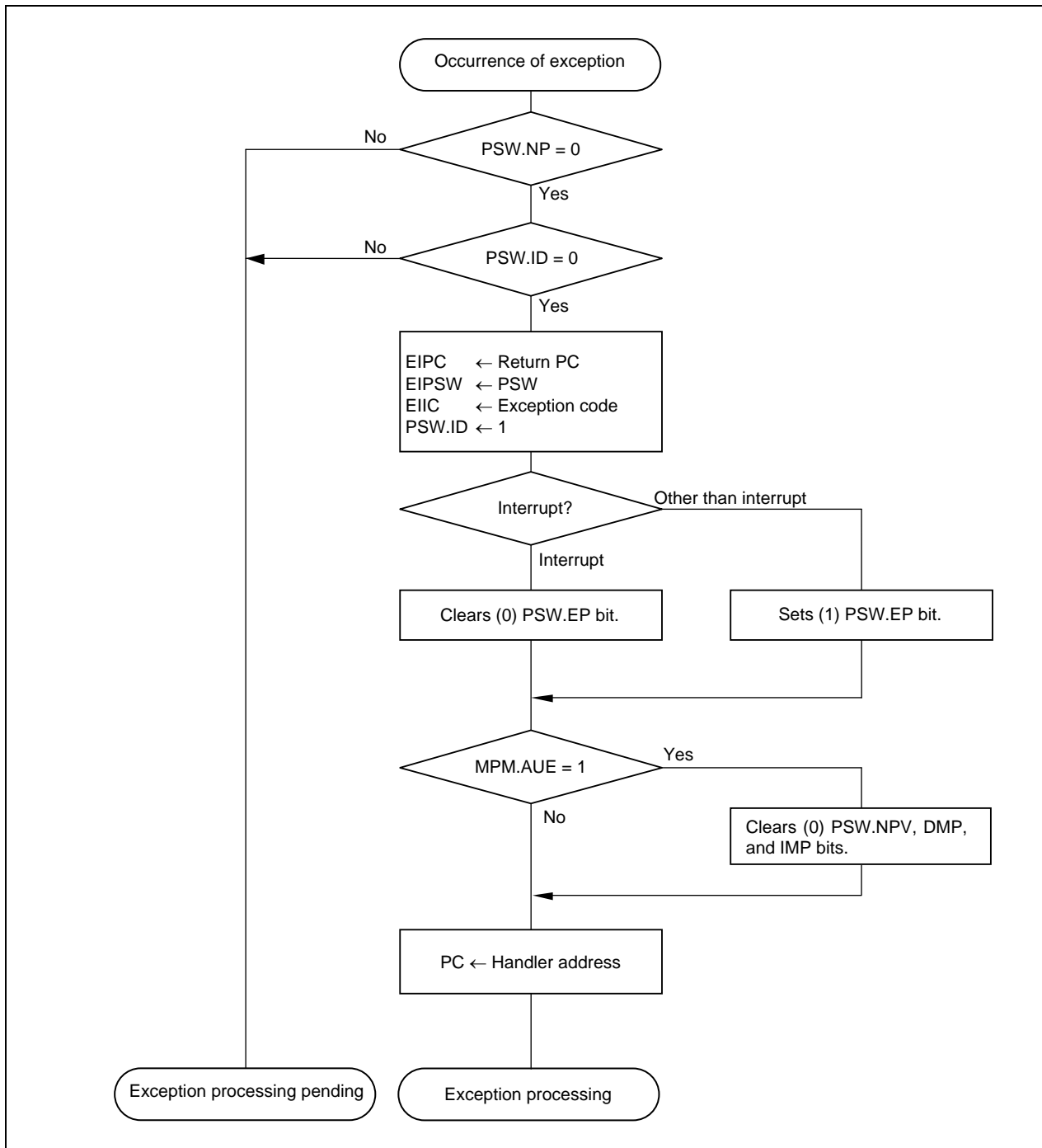
Note Although the exception code is also written to the lower 16 bits (EICC) of the ECR register, the EIIC register should be used except when using an existing program that cannot be revised.

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.NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.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.

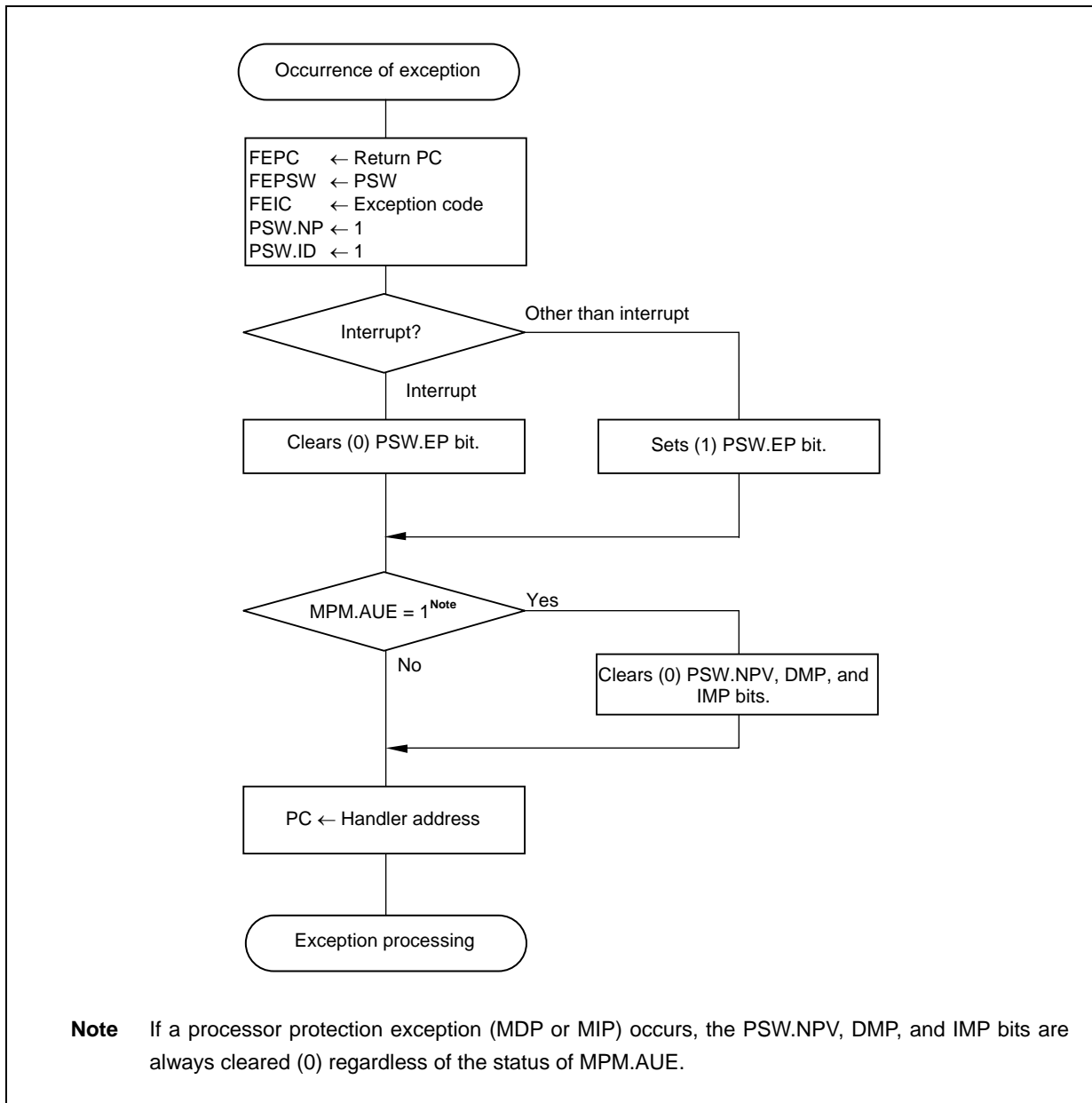
- Notes**
1. Although the exception code is also written to the higher 16 bits (FECC) of the ECR register, the FEIC register should be used except for when using the existing program that cannot be revised.
 2. The PSW.NPV, DMP, and IMP bits are always cleared (0) if an exception related to processor 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



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.NPV, DMP, and IMP bits if the MPM.AUE bit is set (1). Otherwise, the PSW.NPV, DMP, and IMP bits will not be updated.
- <8> Sets an exception handler address to the PC and transfers control to the exception handler.

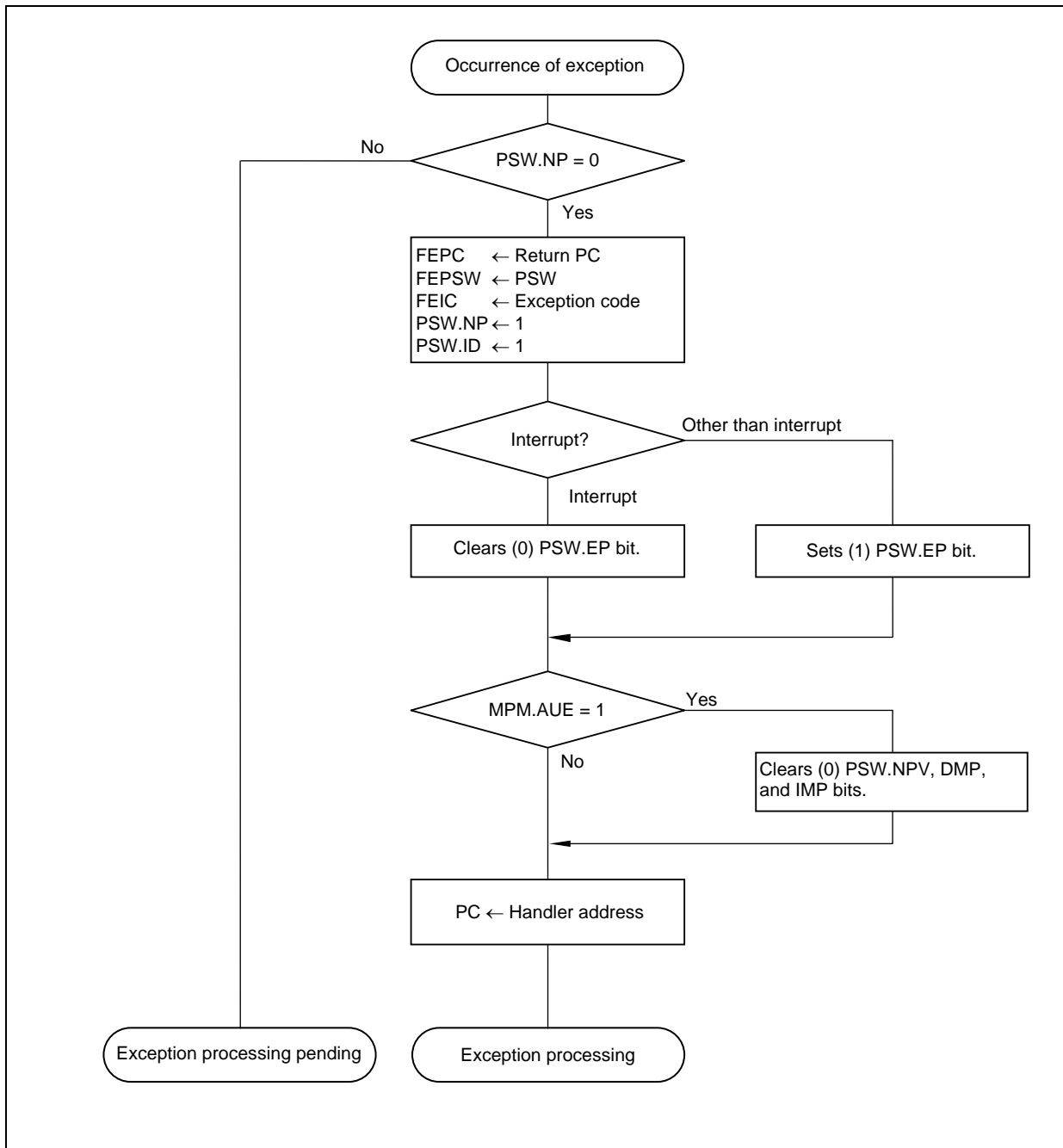
Notes 1. Although the exception code is also written to the lower 16 bits (FECC) of the ECR register, the FEIC register should be used except for when using the existing program that cannot be revised.

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



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) NPV, DMP, and IMP bits of PSW register

If a processor protection exception is acknowledged, the PSW.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.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.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 (RIE) 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 + n H	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 SSCFG.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 V850E2S CPU provides the exception synchronization instruction (SYNCE instruction). The exceptions that are synchronized by the SYNCE instruction are imprecise exceptions. However, since the V850E2S CPU does not support imprecise exceptions, the SYNCE instruction is replaced by the SYNCM instruction when executed.

6.4 Exception Handler Address Switching Function

The V850E2S 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 INT127)
- 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 INT127)

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), 128 different exception handler addresses that include the EH_BASE register and offset addresses are used as the exception handler addresses for INT0 to INT127.

When the RINT bit is set (1), the exception handler addresses for INT0 to INT127 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 INT127 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 INT127.

(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.

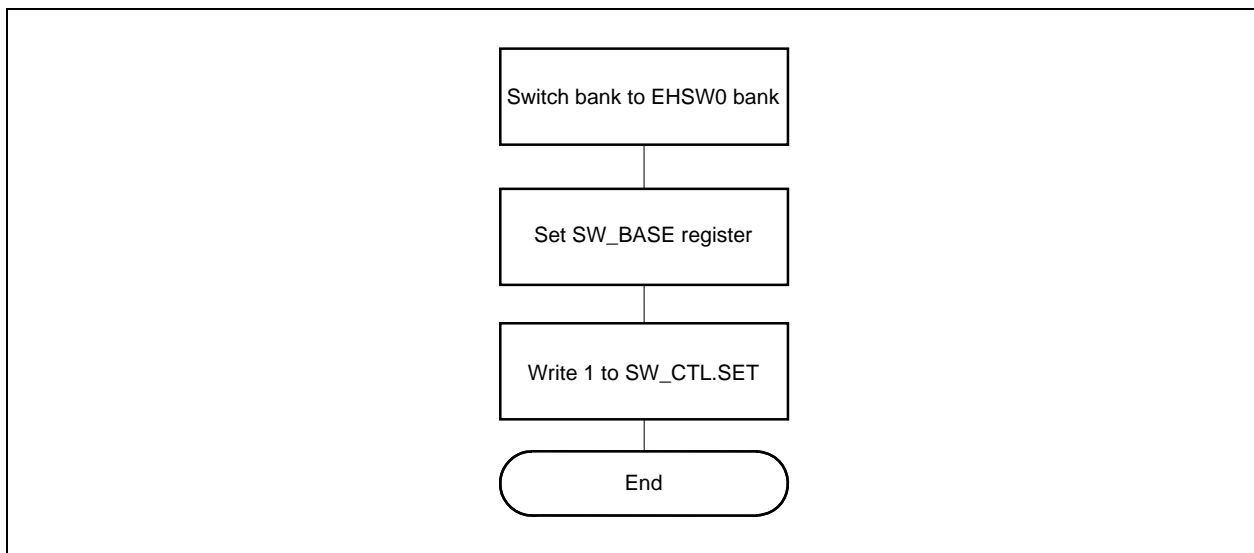
(1) Switch according to software

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 V850E2S CPU defines a function limited to a specific application 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
Undefined	UCPOP0 to UCP0P7 ^{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.
Coprocessor instructions are not defined in the V850E2S CPU.

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 V850E2S 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 V850E2S CPU supplies a function to control accesses to the following two types of resources.

- **System register protection**

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

- **Memory protection**

A total of four protection areas can be set, which are shared as instruction/constant protection areas and data protection areas in 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.

(2) Management by execution level

The V850E2S 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 V850E2S 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

Processor Protection Function (10H)					
Group	Processor protection violation (00H)		Processor protection setting (01H)		Software paging (10H)
Bank	MPV, PROT00		MPU, PROT01		PROT10
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	PA0L	Protection area 0 lower-limit address	PA0L
7	Reserved for function expansion		PA0U	Protection area 0 upper-limit address	PA0U
8			PA1L	Protection area 1 lower-limit address	PA1L
9			PA1U	Protection area 1 upper-limit address	PA1U
10			PA2L	Protection area 2 lower-limit address	PA2L
11			PA2U	Protection area 2 upper-limit address	PA2U
12			PA3L	Protection area 3 lower-limit address	PA3L
13			PA3U	Protection area 3 upper-limit address	PA3U
14			Reserved for function expansion		Reserved for function expansion
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
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	PA0L	Protection area 0 lower-limit address	√	√	√
7	PA0U	Protection area 0 upper-limit address	√	√	√
8	PA1L	Protection area 1 lower-limit address	√	√	√
9	PA1U	Protection area 1 upper-limit address	√	√	√
10	PA2L	Protection area 2 lower-limit address	√	√	√
11	PA2U	Protection area 2 upper-limit address	√	√	√
12	PA3L	Protection area 3 lower-limit address	√	√	√
13	PA3U	Protection area 3 upper-limit address	√	√	√
14-27	(Reserved for future function expansion (Operation is not guaranteed if these registers are accessed.))		×	×	√

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 27	(Reserved for future function expansion (Operation is not guaranteed if these registers are accessed.))		×	×	√

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 V850E2S 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	PA0L	Protection area 0 lower-limit address	√	√	√
7	PA0U	Protection area 0 upper-limit address	√	√	√
8	PA1L	Protection area 1 lower-limit address	√	√	√
9	PA1U	Protection area 1 upper-limit address	√	√	√
10	PA2L	Protection area 2 lower-limit address	√	√	√
11	PA2U	Protection area 2 upper-limit address	√	√	√
12	PA3L	Protection area 3 lower-limit address	√	√	√
13	PA3U	Protection area 3 upper-limit address	√	√	√
14-27	(Reserved for future function expansion (Operation is not guaranteed if these registers are accessed.))		×	×	√

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

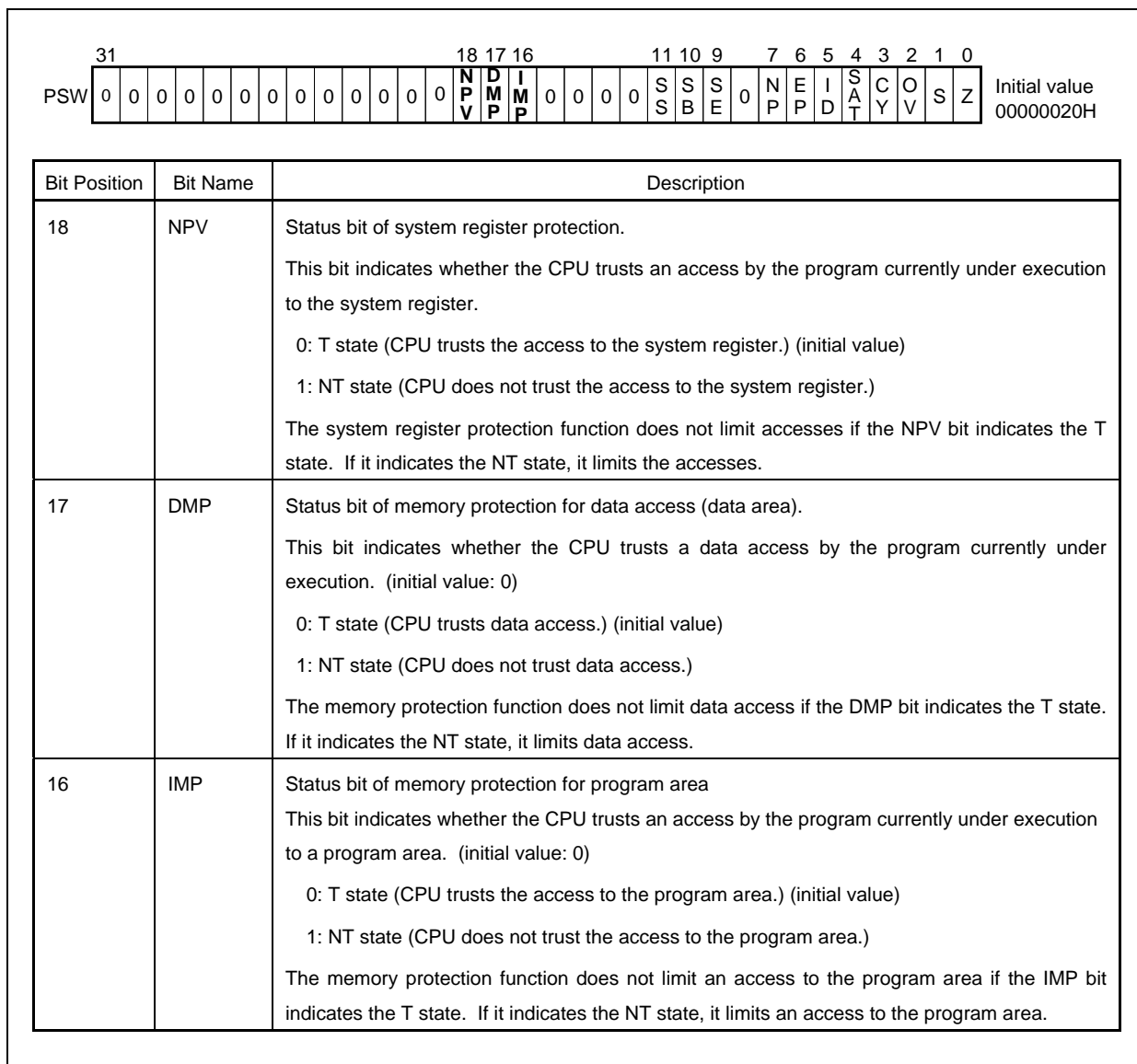
Remark √: Indicates in the column of “Able to Specify Operands?” that the register can be specified. In the column of “System Register Protection”, this symbol indicates that the register is protected.

×: 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 18 of the PSW register in the CPU function bank.

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



(2/3)

Bit Position	Bit Name	Description
1	AUE	<p>This bit automatically changes the execution level if an exception occurs, and sets a function to update the PSW.NPV, DMP, and IMP bits (initial value: 0).</p> <p>0: Does not automatically update.</p> <p>The value of the PSW.NPV, DMP, and IMP bits are not changed even if an exception occurs^{Note2, 3}.</p> <p>1: Automatically updates.</p> <p>The value of the PSW.NPV, DMP, and IMP bits is cleared to 0 if an exception occurs.</p>

- Notes**
1. DWE, DRE, and DXE are valid only for the 64 MB space from 0000_0000H to 01FF_FFFFH and from FE00_0000H to FFFF_FFFFH. Instruction execution and access are prohibited outside of the 64 MB space described above.
 2. The exception that changes the execution level to the DB level and the processor protection violation exceptions (MDP and MIP exceptions) are excluded. When these exceptions are acknowledged, the PSW.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).

Remark The PANL/U register setting takes priority among all settings in the 64 MB space where addressing is enabled. For example, even if read access is enabled for a space that is not covered by the area specification in DRE = 1, read access is still prohibited in any area set as access-prohibited in the PANL/U register.

(3/3)

Bit Position	Bit Name	Description
0	MPE	<p>This bit enables or disables the operation of the processor protection function (initial value: 0).</p> <p>0: Processor protection function disabled</p> <ul style="list-style-type: none"> 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. <p>1: Processor protection function enabled</p> <ul style="list-style-type: none"> 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.

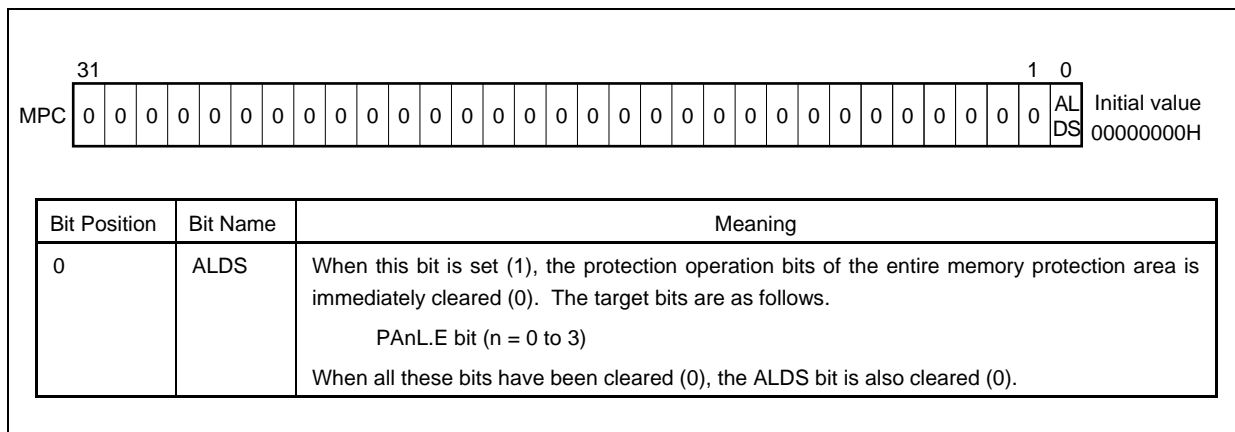
Remark For details on the NPV, IMP, and DMP bits, refer to **2.2.1 PSW – Program Status Word**.

Caution When data access occurs outside the 64 MB area where addressing is not enabled, if the processor protection function is enabled (MPE = 1), all data access is detected as a memory protection violation that throws an MDP exception. However, it is not possible to detect memory protection violations (MIP exceptions) that occur due to execution of an instruction outside the 64 MB area where addressing is not enabled.

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”.

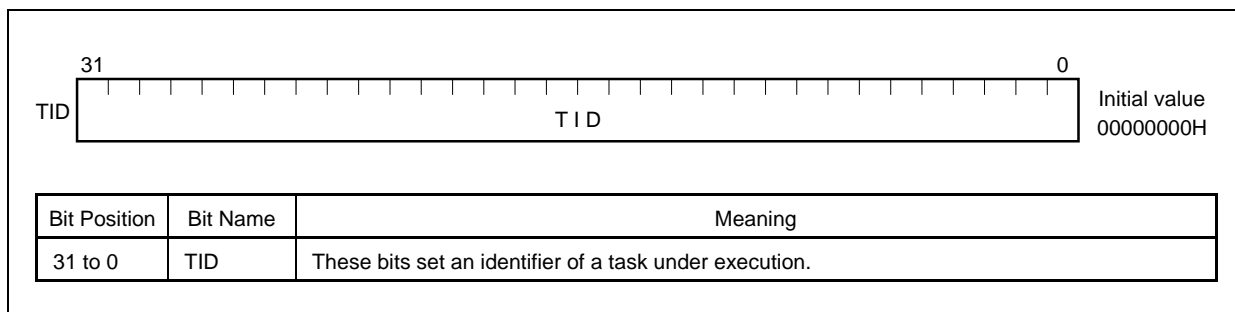


Caution Even when the bits cleared (0) by the function of the ALDS bit are set (1) by the LDSR instruction immediately after the LDSR instruction that has set the ALDS bit (1), the result in accordance with the execution sequence of the instruction can be obtained (the target bits are set (1)).

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.



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 8 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.NPV, DMP, and IMP bits 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.NPV, DMP, and IMP bits can be updated.
(The system register protection function and memory protection 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.NPV, DMP, and IMP bits are fixed to 0, and use of the processor protection function is stopped as follows.

- The PSW.NPV, DMP, and IMP bits are fixed to 0.
(The system register protection function and memory protection function cannot be used.)

CHAPTER 4 EXECUTION LEVEL

The V850E2S 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 3 bits of the PSW (Program Status Word). These bits are called protection bits, and specific combinations of these bits are called execution levels.

- 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 drivers. 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 three protected subjects; system registers, data area, and program area, PSW.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 (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 18, 17, and 16 of the PSW are defined as NPV, DMP, and IMP bits. Set these bits appropriately when using the processor protection function.

Caution The PSW.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.

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 V850E2S CPU assumes that some combinations of the statuses of the PSW.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	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	Exception handler, OS kernel, etc.
1	0	0	0	Device driver, etc.
2	0	1	1	Common library, etc.
3	1	1	1	User task

4.4 Transition of Execution Level

With the V850E2S 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.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 V850E2S 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.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.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.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 or MIP) occurs, the PSW.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 NPV, DMP, and IMP bits of the register that stores the value of the return PSW is copied to the NPV, DMP, and IMP bits of the PSW. The value of the PSW.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 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.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 V850E2S 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 V850E2S 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 processor protection violation bank.

5.1 Register Set

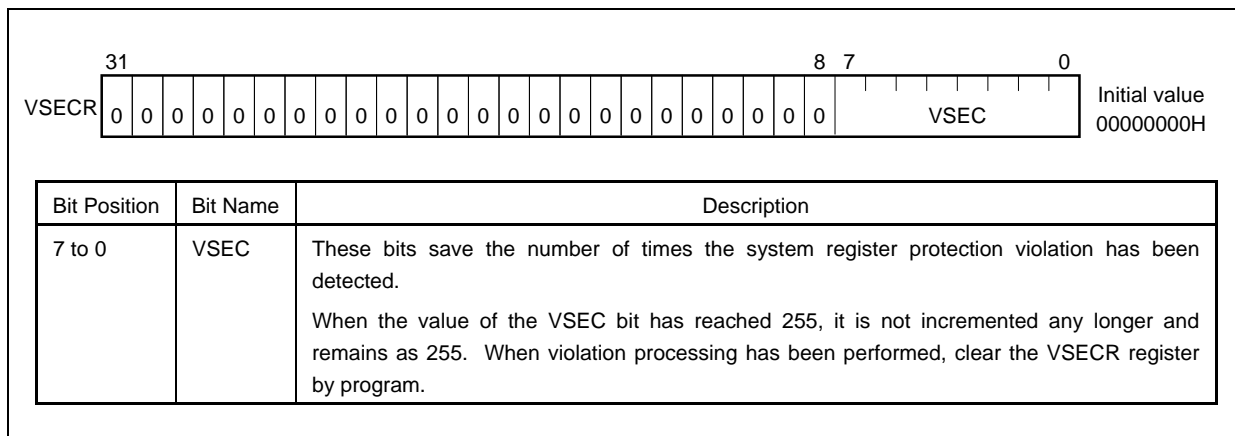
Table 5-1 shows the system registers related to the system register protection function.

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	PA0L	Protection area 0 lower-limit address	PA0L		
7	Reserved for function expansion		PA0U	Protection area 0 upper-limit address	PA0U		
8			PA1L	Protection area 1 lower-limit address	PA1L		
9			PA1U	Protection area 1 upper-limit address	PA1U		
10			PA2L	Protection area 2 lower-limit address	PA2L		
11			PA2U	Protection area 2 upper-limit address	PA2U		
12			PA3L	Protection area 3 lower-limit address	PA3L		
13			PA3U	Protection area 3 upper-limit address	PA3U		
14					Reserved for function expansion		Reserved for function expansion
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							

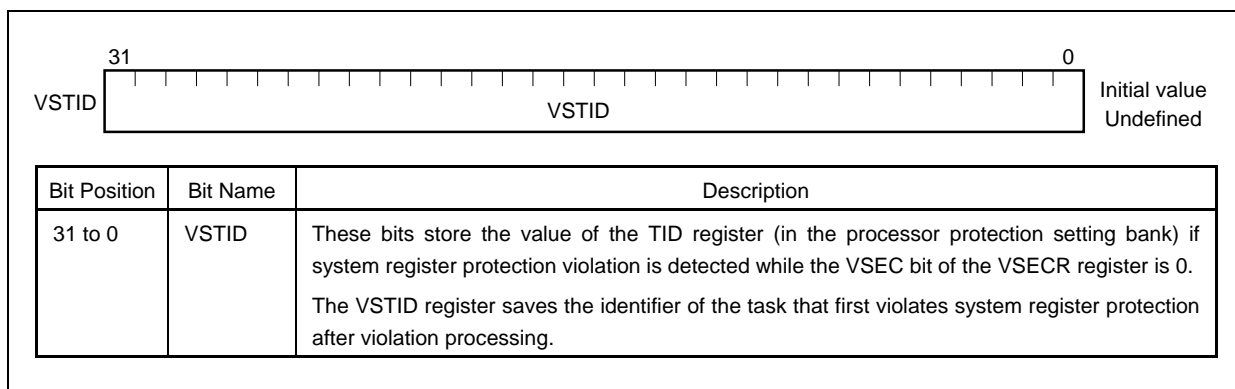
5.1.1 VSECR – System register protection violation cause

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.



5.1.2 VSTID – System register protection violation task identifier

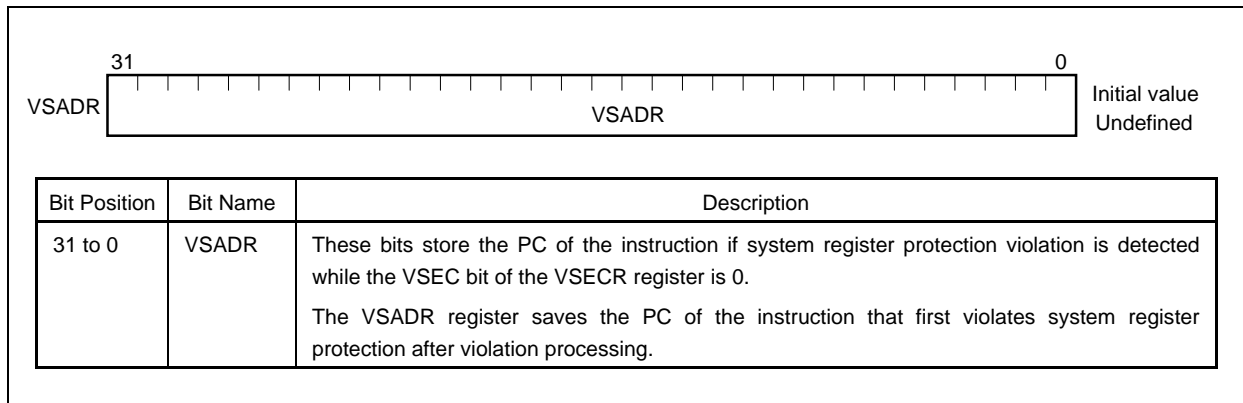
This register saves the contents of the task identifier (TID) when the first instruction that is detected by the system register protection function as violation is executed.



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.



Caution Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of the VSADR register is automatically set to bits 31 to 26.

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

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 V850E2S 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 V850E2S 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 protection setting is enabled for 64 MB of 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	PA0L	Protection area 0 lower-limit address	PA0L		
7	Reserved for function expansion		PA0U	Protection area 0 upper-limit address	PA0U		
8			PA1L	Protection area 1 lower-limit address	PA1L		
9			PA1U	Protection area 1 upper-limit address	PA1U		
10			PA2L	Protection area 2 lower-limit address	PA2L		
11			PA2U	Protection area 2 upper-limit address	PA2U		
12			PA3L	Protection area 3 lower-limit address	PA3L		
13			PA3U	Protection area 3 upper-limit address	PA3U		
14					Reserved for function expansion		Reserved for function expansion
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							

6.1.1 PAnL – Protection area n lower-limit address (n = 0 to 3)

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

Be sure to set bits 3 and 2 to “0”.

PAnL																31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16															
																AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL
																15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0															
																AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	AL	0	0	S	E

Initial value
00000000H

Bit Position	Bit Name	Description
31 to 4	AL31 to AL4	<p>These bits set the lower-limit address of the protection area.</p> <p>Because bits 3 to 0 of the PAnL register are used for the other setting of the protection area, bits 3 to 0 (AL3 to 0) of the lower-limit address are implicitly 0.</p> <p>Specify the 64 MB space from 00000000H to 01FFFFFFH and from FE000000H to FFFFFFFFH.</p> <p>Write to AL31 to AL26 is ignored, and the value of AL25 is used for extension to AL31.</p>
1	S	<p>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.</p> <p>If an instruction that accesses the access-prohibited protection area is executed, data protection violation is detected, and MDP exception is immediately acknowledged.</p> <p>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.</p>
0	E	<p>This bit enables or disable the setting of the protection area.</p> <p>When the E bit is cleared (0), the contents of all the other setting bits are invalid, and no protection area is set.</p> <p>0: Invalid (Protection area n is not used.) 1: Valid (Protection area n is used.)</p>

Remark n = 0 to 3

6.1.2 PAnU – Protection area n upper-limit address (n = 0 to 3)

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

Be sure to set bits 3 to “0”.

<table border="1" style="width: 100%; text-align: center;"> <tr> <td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td> </tr> <tr> <td>AU31</td><td>AU30</td><td>AU29</td><td>AU28</td><td>AU27</td><td>AU26</td><td>AU25</td><td>AU24</td><td>AU23</td><td>AU22</td><td>AU21</td><td>AU20</td><td>AU19</td><td>AU18</td><td>AU17</td><td>AU16</td> </tr> </table>															31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	AU31	AU30	AU29	AU28	AU27	AU26	AU25	AU24	AU23	AU22	AU21	AU20	AU19	AU18	AU17	AU16	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																
AU31	AU30	AU29	AU28	AU27	AU26	AU25	AU24	AU23	AU22	AU21	AU20	AU19	AU18	AU17	AU16																																
<table border="1" style="width: 100%; text-align: center;"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>AU15</td><td>AU14</td><td>AU13</td><td>AU12</td><td>AU11</td><td>AU10</td><td>AU9</td><td>AU8</td><td>AU7</td><td>AU6</td><td>AU5</td><td>AU4</td><td>0</td><td>W</td><td>R</td><td>X</td> </tr> </table>															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	X	Initial value 00000000H
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	X																																
Bit Position	Bit Name	Description																																													
31 to 4	AU31 to AU4	<p>These bits set the upper-limit address of the protection area.</p> <p>Because bits 3 to 0 of the PAnU register are used for the other setting of the protection area, bits 3 to 0 (AU3 to 0) of the upper-limit address are implicitly 1.</p> <p>Specify the 64 MB space from 00000000H to 01FFFFFFH and from FE000000H to FFFFFFFFH.</p>																																													
2	W	<p>This bit enables or disables a write access to the protection area.</p> <p>When the W bit is cleared (0), a write access to the data placed in the protection area of the data area is prohibited.</p> <p>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.</p> <p>0: Disables write access to the protection area. 1: Enables write access to the protection area.</p>																																													
1	R	<p>This bit enables or disables a read access to the protection area.</p> <p>When the R bit is cleared (0), a read access to the data placed in the protection area of the data area is prohibited.</p> <p>If an instruction that reads the access-prohibited protection area is executed, data protection violation is detected and the MDP exception is immediately acknowledged.</p> <p>0: Disables read access to the protection area. 1: Enables read access to the protection area.</p>																																													
0	X	<p>This bit enables or disables instruction execution for the protection area.</p> <p>When the X bit is cleared (0), execution of the program placed in the protection area of the program area is prohibited.</p> <p>If an instruction is executed for the protection area, instruction protection violation is detected and the MIP exception is immediately acknowledged.</p> <p>0: Disables instruction execution for the protection area. 1: Enables instruction execution for the protection area.</p>																																													
<p>Remark n = 0 to 3</p>																																															

6.1.3 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”.

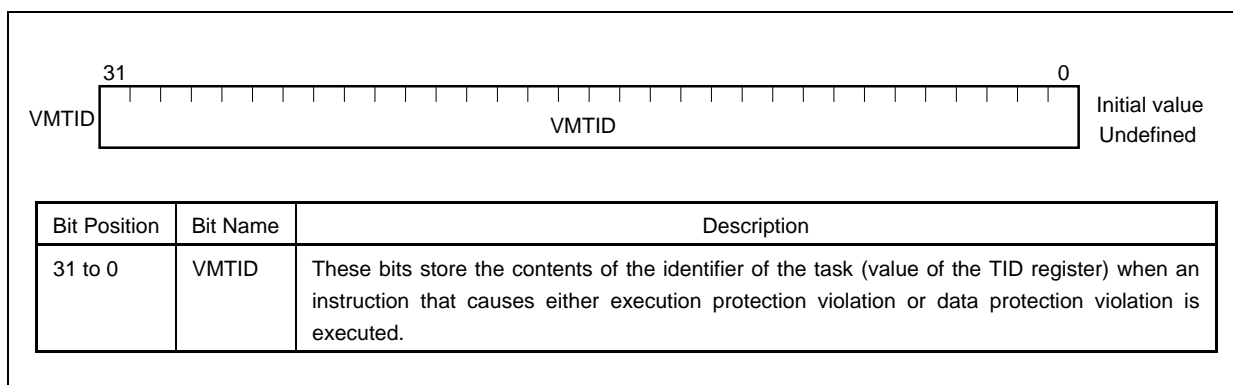
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
VMECR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	0	VMMS	VM RMW	VMS	VMW	VMR	VMX	0	Initial value 00000000H

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 **7.3 Identifying Violation Cause**.

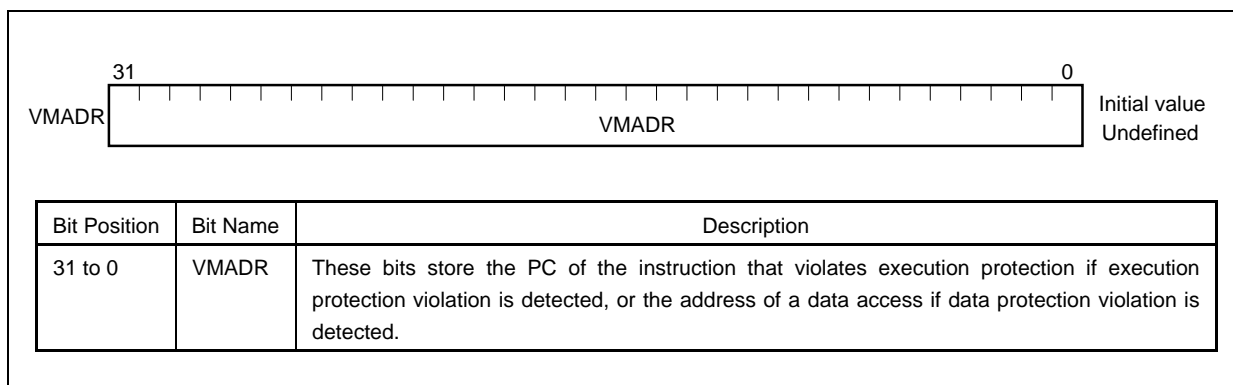
6.1.4 VMTID – Memory protection violation task identifier

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



6.1.5 VMADR – Memory protection violation address

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



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 V850E2S CPU uses the following register as a pair to set a protection area.

- PAnL/PAnU (n = 0 to 3)

Each protection area is set by two combinations of registers: an upper-limit register and a lower-limit register. The registers defined for the V850E2S CPU allow the placement of up to 4 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		PAnU					PAnL			
		Bits 31 to 4	Bit 3	Bit 2	Bit 1	Bit 0	Bits 31 to 4	Bit 3	Bit 1	Bit 0
Field function	Field name	Upper-limit address (mask value)	(RFU)	Write enable	Read enable	Execution enable	Lower-limit address (base address)	(RFU)	sp indirect access enable	Area enable
	AU			W	R	X	AL		S	E
PA0U/L	Instruction protection area setting	Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	(0)	(0)
PA1U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	(0)	(0)
PA2U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	(0)	(0)
PA3U/L		Upper-limit address	0	0	(0)	(0)	Lower-limit address	0	(0)	(0)

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.

The default value of each register is as follows.

Register	Initial Value
PA0L to PA3L	0000 0000H
PA0U to PA3U	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 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.

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

The lower-limit address of the protection area is indicated.

Because the bits 3 to 0 of the PAnL registers are used for the other setting of the protection area, the bits 3 to 0 (AL3 to AL0) of the lower-limit address are implicitly 0 (n = 0 to 3).

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

The upper-limit address of the protection area is indicated.

Because the bits 3 to 0 of the PAnU registers are used for the other setting of the protection area, the bits 3 to 0 (AU3 to AU0) of the upper-limit address are implicitly 1 (n = 0 to 3).

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, setting access control of the crossing part takes precedence.

(1) When used as instruction protection area

If two or more protection areas 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) When used as 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 Special Memory Access Instructions

The V850E2S 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

6.5.1 Load and store instructions executing misaligned access

With the V850E2S 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.

6.5.2 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.

6.5.3 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.

6.5.4 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.

6.6 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 7 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 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.

7.1 Types of Violations

The V850E2S 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 three 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

7.1.1 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**.

7.1.2 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.

7.1.3 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.

7.2 Types of Exceptions

The V850E2S 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.

7.2.1 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.

7.2.2 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.

7.3 Identifying Violation Cause

If protection violation is detected, an exception cause that indicates which exception, MIP or MDP, 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 7-1, auxiliary information indicating the cause of each exception is stored in a specific system register of the MPV bank.

Table 7-1. 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
Other exceptions	–	–	–	–

7.3.1 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 7-2. VMECR Set Value When MIP Exception Occurs

Register	VMECR						
	6	5	4	3	2	1	0
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

7.3.2 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 7-3. VMECR Set Value When MDP Exception Occurs

Register	VMECR						
	6	5	4	3	2	1	0
Bit name	VMMS	VMRMW	VMS	VMW	VMR	VMX	–
Read instruction (aligned) ^{Note 1}	0	0	0/1 ^{Note 5}	0	1	0	0
Write instruction (aligned) ^{Note 2}	0	0	0/1 ^{Note 5}	1	0	0	0
Read instruction (misaligned) ^{Note 3}	1	0	0/1 ^{Note 5}	0	1	0	0
Write instruction (misaligned) ^{Note 2}	1	0	0/1 ^{Note 5}	1	0	0	0
CAXI/SET1/NOT1/CLR1 instruction ^{Note 4}	0	1	0/1 ^{Note 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.

CHAPTER 8 SPECIFAL FUNCTON

This chapter explains the special function related to the processor protection function.

8.1 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.

- PAnL.E bit (n = 0 to 3)

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

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	0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0				
SYNCE		I	0 0 0 0 0	0 0 0 0 0 0 0	1 1 1 0 1				
SYNCM		I	0 0 0 0 0	0 0 0 0 0 0 0	1 1 1 1 0				
SYNCP		I	0 0 0 0 0	0 0 0 0 0 0 0	1 1 1 1 1				
MOV	reg1, reg2	I	r r r r r	0 0 0 0 0 0 0	R R R R R				rrrrr ≠ 00000
NOT	reg1, reg2	I	r r r r r	0 0 0 0 0 0 1	R R R R R				
RIE		I	0 0 0 0 0	0 0 0 1 0	0 0 0 0 0				
SWITCH	reg1	I	0 0 0 0 0	0 0 0 0 1 0	R R R R R				
FETRAP	vector4	I	0 i i i i	0 0 0 0 1 0	0 0 0 0 0				iiii ≠ 0000
DIVH	reg1, reg2	I	r r r r r	0 0 0 0 1 0	R R R R R				rrrrr ≠ 00000, RRRRR ≠ 00000
JMP	[reg1]	I	0 0 0 0 0 0	0 0 0 0 1 1	R R R R R				
SLD.BU	disp4 [ep], reg2	IV	r r r r r	0 0 0 0 1 1	0 d d d d				rrrrr ≠ 00000
SLD.HU	disp5 [ep], reg2	IV	r r r r r	0 0 0 0 1 1	1 d d d d				rrrrr ≠ 00000
ZXB	reg1	I	0 0 0 0 0	0 0 0 1 0 0	R R R R R				
SXB	reg1	I	0 0 0 0 0	0 0 0 1 0 1	R R R R R				
ZXH	reg1	I	0 0 0 0 0	0 0 0 1 1 0	R R R R R				
SXH	reg1	I	0 0 0 0 0	0 0 0 1 1 1	R R R R R				
SATSUBR	reg1, reg2	I	r r r r r	0 0 0 1 0 0	R R R R R				rrrrr ≠ 00000
SATSUB	reg1, reg2	I	r r r r r	0 0 0 1 0 1	R R R R R				rrrrr ≠ 00000
SATADD	reg1, reg2	I	r r r r r	0 0 0 1 1 0	R R R R R				rrrrr ≠ 00000
MULH	reg1, reg2	I	r r r r r	0 0 0 1 1 1	R R R R R				rrrrr ≠ 00000
OR	reg1, reg2	I	r r r r r	0 0 1 0 0 0	R R R R R				
XOR	reg1, reg2	I	r r r r r	0 0 1 0 0 1	R R R R R				
AND	reg1, reg2	I	r r r r r	0 0 1 0 1 0	R R R R R				
TST	reg1, reg2	I	r r r r r	0 0 1 0 1 1	R R R R R				
SUBR	reg1, reg2	I	r r r r r	0 0 1 1 0 0	R R R R R				
SUB	reg1, reg2	I	r r r r r	0 0 1 1 0 1	R R R R R				
ADD	reg1, reg2	I	r r r r r	0 0 1 1 1 0	R R R R R				
CMP	reg1, reg2	I	r r r r r	0 0 1 1 1 1	R R R R R				
MOV	imm5, reg2	I	r r r r r	0 1 0 0 0 0	i i i i i				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	rrrrrr	010001	iiiiii				rrrrrr ≠ 00000
ADD	imm5, reg2	I	rrrrrr	010010	iiiiii				
CMP	imm5, reg2	I	rrrrrr	010011	iiiiii				
CALLT	imm6	II	00000	01000i	iiiiii				
SHR	imm5, reg2	II	rrrrrr	010100	iiiiii				
SAR	imm5, reg2	II	rrrrrr	010101	iiiiii				
SHL	imm5, reg2	II	rrrrrr	010110	iiiiii				
MULH	imm5, reg2	II	rrrrrr	010111	iiiiii				rrrrrr ≠ 00000
JR	disp32	VI	00000	010111	00000	dddd	dddddd	dddd0	See Table B-2
JARL	disp32, reg1	VI	00000	010111	RRRRR	dddd	dddddd	dddd0	See Table B-2 RRRRR ≠ 00000
SLD.B	disp7 [ep], reg2	IV	rrrrrr	0110dd	dddd				
SST.B	reg2, disp7 [ep]	IV	rrrrrr	0111dd	dddd				
SLD.H	disp8 [ep], reg2	IV	rrrrrr	1000dd	dddd				
SST.H	reg2, disp8 [ep]	IV	rrrrrr	1001dd	dddd				
SLD.W	disp8 [ep], reg2	IV	rrrrrr	1010dd	dddd0				
SST.W	reg2, disp8 [ep]	IV	rrrrrr	1010dd	dddd1				
Bcond	disp9	III	dddd	1011dd	dcccc				
ADDI	imm16, reg1, reg2	VI	rrrrrr	110000	RRRRR	iiiiii	iiiiiii	iiiiii	
MOV	imm32, reg1	VI	00000	110001	RRRRR	IIIII	IIIIII	IIIII	See Table B-2
MOVEA	imm16, reg1, reg2	VI	rrrrrr	110001	RRRRR	iiiiii	iiiiiii	iiiiii	rrrrrr ≠ 00000
MOVHI	imm16, reg1, reg2	VI	rrrrrr	110010	RRRRR	iiiiii	iiiiiii	iiiiii	rrrrrr ≠ 00000
SATSUBI	imm16, reg1, reg2	VI	rrrrrr	110011	RRRRR	iiiiii	iiiiiii	iiiiii	rrrrrr ≠ 00000
DISPOSE	imm5, list12	XIII	00000	11001i	iiiiL	LLLLL	LLLLLL	00000	
DISPOSE	imm5, list12, [reg1]	XIII	00000	11001i	iiiiL	LLLLL	LLLLLL	RRRRR	RRRRR ≠ 00000
ORI	imm16, reg1, reg2	VI	rrrrrr	110100	RRRRR	iiiiii	iiiiiii	iiiiii	
XORI	imm16, reg1, reg2	VI	rrrrrr	110101	RRRRR	iiiiii	iiiiiii	iiiiii	
ANDI	imm16, reg1, reg2	VI	rrrrrr	110110	RRRRR	iiiiii	iiiiiii	iiiiii	
MULHI	imm16, reg1, reg2	VI	rrrrrr	110111	RRRRR	iiiiii	iiiiiii	iiiiii	rrrrrr ≠ 00000
JMP	imm32 [reg1]	VI	00000	110111	RRRRR	dddd	dddddd	dddd0	See Table B-2
LD.B	disp16 [reg1], reg2	VII	rrrrrr	111000	RRRRR	dddd	dddddd	dddd	
LD.H	disp16 [reg1], reg2	VII	rrrrrr	111001	RRRRR	dddd	dddddd	dddd0	
LD.W	disp16 [reg1], reg2	VII	rrrrrr	111001	RRRRR	dddd	dddddd	dddd1	
ST.B	reg2, disp16 [reg1]	VII	rrrrrr	111010	RRRRR	dddd	dddddd	dddd	
ST.H	reg2, disp16 [reg1]	VII	rrrrrr	111011	RRRRR	dddd	dddddd	dddd0	
ST.W	reg2, disp16 [reg1]	VII	rrrrrr	111011	RRRRR	dddd	dddddd	dddd1	
PREPARE	list12, imm5	XIII	00000	11110i	iiiiL	LLLLL	LLLLLL	00001	
PREPARE	list12, imm5, sp/imm	XIII	00000	11110i	iiiiL	LLLLL	LLLLLL	ff011	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 w	d 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 w	d 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 w	d 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 w	d 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 w	d 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 w	d 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 w	d 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 w	d 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 d d 0	
JARL	disp22, reg2	V	r 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 d d 0	rrrrr ≠ 00000
LD.BU	disp16 [reg1], reg2	VII	r r r r r r	1 1 1 1 0 b	R R R R R R	d d 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 R	d d d d d d	d d d d d d d	d d d d d	
NOT1	bit3#, disp16 [reg1]	VIII	0 1 b b b	1 1 1 1 1 0	R R R R R R	d d 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 R	d d 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 R	d d 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 r	1 1 1 1 1 1	R R R R R R	d d d d d d	d d d d d d d	d d d d 1	rrrrr ≠ 00000
SETF	cond, reg2	IX	r 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, regD	IX	r r r r r r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R R R R R R	w 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R R R R R R	w 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R R R R R R	w 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R 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 r	1 1 1 1 1 1	R R R R R R	w 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	1 1 0 1 0	1 1 1 1 1 1 1	v v v v v v	0 0 V V V V	0 0 1 0 1 1	0 0 0 0 0	
SASF	cccc, reg2	IX	r r r r r r	1 1 1 1 1 1 1	0 c c c c c	0 0 0 0 0	0 1 0 0 0 0	0 0 0 0 0	
MUL	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 0 0 1	0 0 0 0 0	
MULU	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 0 0 1	0 0 0 1 0	
MUL	imm9, reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	i i i i i i	w w w w w w	0 1 0 0 1 I	I I I I 0 0	
MULU	imm9, reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	i i i i i i	w w w w w w	0 1 0 0 1 I	I I I I 1 0	
DIVH	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 0 0	0 0 0 0 0	
DIVHU	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 0 0	0 0 0 1 0	
DIV	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 1 0	0 0 0 0 0	
DIVQ	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 1 1	1 1 1 0 0	
DIVU	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 1 0	0 0 0 1 0	
DIVQU	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 0 1 1 1	1 1 1 1 0	
CMOV	cccc, imm5, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	i i i i i i	w w w w w w	0 1 1 0 0 0	c c c c 0	
CMOV	cccc, reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 1 0 0 1	c c c c 0	
BSW	reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 0	0 0 0 0 0	
BSH	reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 0	0 0 0 1 0	
HSW	reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 0	0 0 1 0 0	
HSH	reg2, reg3	XII	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 0	0 0 1 1 0	
SCH0R	reg2, reg3	IX	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 1	0 0 0 0 0	
SCH1R	reg2, reg3	IX	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 1	0 0 0 1 0	
SCH0L	reg2, reg3	IX	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 1	0 0 1 0 0	
SCH1L	reg2, reg3	IX	r r r r r r	1 1 1 1 1 1 1	0 0 0 0 0	w w w w w w	0 1 1 0 1 1	0 0 1 1 0	
SBF	cccc, reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 1 1 0 0	c c c c 0	cccc ≠ 1101
SATSUB	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 1 1 0 0	1 1 0 1 0	
ADF	cccc, reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 1 1 0 1	c c c c 0	cccc ≠ 1101
SATADD	reg1, reg2, reg3	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w w w	0 1 1 1 0 1	1 1 0 1 0	
MAC	reg1, reg2, reg3, reg4	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w 0	0 1 1 1 1 0	m m m m 0	
MACU	reg1, reg2, reg3, reg4	XI	r r r r r r	1 1 1 1 1 1 1	R R R R R R	w w w w 0	0 1 1 1 1 1	m m m m 0	

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	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 d 0	D D D D D D	D D D D D D D D	D D D D D D
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 d 0	D D D D D D	D D D D D D D D	D D D D D D
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 i	I I I I I I	I I I I I I I I	I I I I I I
JMP	disp32 [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 d 0	D D D D D D	D D D D D D D D	D D D D D D
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	ff ^{Note} 011	I I I I I I	I I I I I I I I	I I I I I I
LD.B	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R	w w w w w	d d d d d d	d 0 1 0 1	D D D D D D	D D D D D D D D	D D D D D D
LD.H	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R	w w w w w	d d d d d d	0 0 1 1 1	D D D D D D	D D D D D D D D	D D D D D D
LD.W	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R	w w w w w	d d d d d d	0 1 0 0 1	D D D D D D	D D D D D D D D	D D D D D D
ST.B	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R	w w w w w	d d d d d d	d 1 1 0 1	D D D D D D	D D D D D D D D	D D D D D D
ST.W	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 0	R R R R R	w w w w w	d d d d d d	0 1 1 1 1	D D D D D D	D D D D D D D D	D D D D D D
LD.BU	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R	w w w w w	d d d d d d	d 0 1 0 1	D D D D D D	D D D D D D D D	D D D D D D
LD.HU	disp23[reg1], reg3	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R	w w w w w	d d d d d d	0 0 1 1 1	D D D D D D	D D D D D D D D	D D D D D D
ST.H	reg3, disp23[reg1]	XIV	0 0 0 0 0	1 1 1 1 0 1	R R R R R	w w w w w	d d d d d d	0 1 1 0 1	D D D D D D	D D D D D D D D	D D D D D D

Note ff = 01, 10

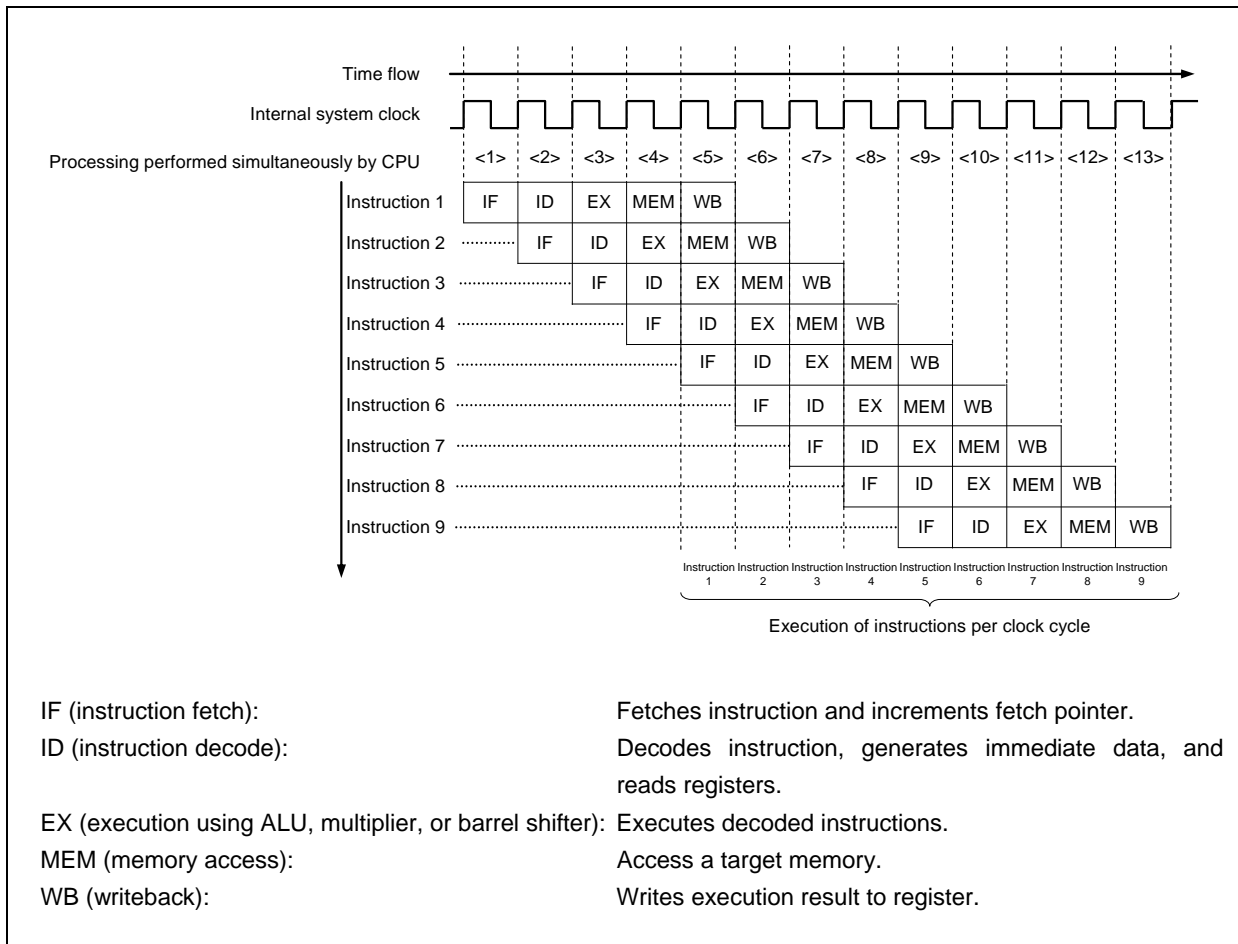
Table B-3. Basic Instruction Opcode Map (64-bit Instruction)

Mnemonic	Operand	Format	opcode						Remark				
			15 11	10 5	4 0	31 27	26 21	20 16					
PREPARE	list12, imm5, sp/imm	XIII	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	1 1 0 1 1					
			47			32				63		48	
			I I I I I	I I I I I I	I I I I I	I I I I I	I I I I I I	I I I I I					

APPENDIX C PIPELINES

The V850E2S CPU, which is based on RISC architecture, uses five-stage pipeline control to execute almost all types of instructions in just one clock cycle. The instruction execution sequence normally includes five 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 9 typical instructions are executed consecutively.

Figure C-1. Example of Consecutive Execution of 9 Typical Instructions

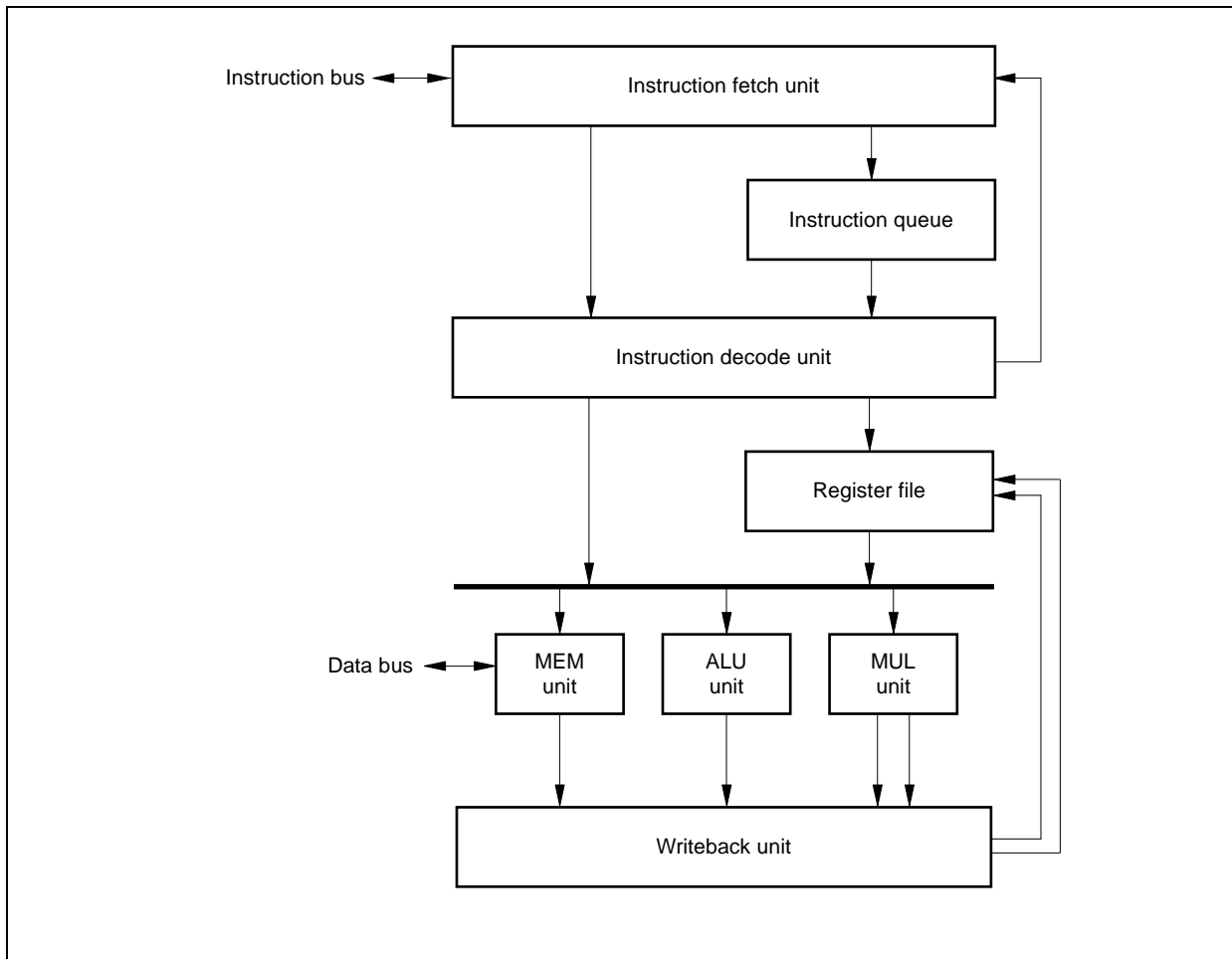


<1> to <13> are CPU states.

C.1 Features

Figure C-2 shows the pipeline configuration of the CPU assumed by the V850E2S CPU.

Figure C-2. Pipeline Configuration



This pipeline includes the following functions.

- (a) Instruction fetch unit
This unit fetches one instruction per cycle using a 32-bit fetch bus (CPU fetch bus).
- (b) Instruction decode unit
This unit decodes instructions issued from the instruction fetch unit.
- (c) ALU unit
This unit issues instructions that perform integer operations and/or logic operations.
- (d) MEM unit
This unit executes instructions (such as load and store instructions) that perform memory access.
- (e) MUL unit
This unit executes instructions that perform integer multiplications.
- (f) 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		
				issue	repeat	latency
Load instructions	LD.B	disp16 [reg1] , reg2	4	1	1	2 ^{Note 1}
	LD.B	disp23 [reg1] , reg3	6	1	1	2 ^{Note 1}
	LD.BU	disp16 [reg1] , reg2	4	1	1	2 ^{Note 1}
	LD.BU	disp23 [reg1] , reg3	6	1	1	2 ^{Note 1}
	LD.H	disp16 [reg1] , reg2	4	1	1	2 ^{Note 1}
	LD.H	disp23 [reg1] , reg3	6	1	1	2 ^{Note 1}
	LD.HU	disp16 [reg1] , reg2	4	1	1	2 ^{Note 1}
	LD.HU	disp23 [reg1] , reg3	6	1	1	2 ^{Note 1}
	LD.W	disp16 [reg1] , reg2	4	1	1	2 ^{Note 1}
	LD.W	disp23 [reg1] , reg3	6	1	1	2 ^{Note 1}
	SLD.B	disp7 [ep] , reg2	2	1	1	2 ^{Note 1}
	SLD.BU	disp4 [ep] , reg2	2	1	1	2 ^{Note 1}
	SLD.H	disp8 [ep] , reg2	2	1	1	2 ^{Note 1}
	SLD.HU	disp5 [ep] , reg2	2	1	1	2 ^{Note 1}
	SLD.W	disp8 [ep] , reg2	2	1	1	2 ^{Note 1}
Store instructions	ST.B	reg2, disp16 [reg1]	4	1	1	1
	ST.B	reg3, disp23 [reg1]	6	1	1	1
	ST.H	reg2, disp16 [reg1]	4	1	1	1
	ST.H	reg3, disp23 [reg1]	6	1	1	1
	ST.W	reg2, disp16 [reg1]	4	1	1	1
	ST.W	reg3, disp23 [reg1]	6	1	1	1
	SST.B	reg2, disp7 [ep]	2	1	1	1
	SST.H	reg2, disp8 [ep]	2	1	1	1
	SST.W	reg2, disp8 [ep]	2	1	1	1
Multiply instructions	MUL	reg1, reg2, reg3	4	1	4	4
	MUL	imm9, reg2, reg3	4	1	4	4
	MULH	reg1, reg2	2	1	1	1
	MULH	imm5, reg2	2	1	1	1
	MULHI	imm16, reg1, reg2	4	1	1	1
	MULU	reg1, reg2, reg3	4	1	4	4
	MULU	imm9, reg2, reg3	4	1	4	4
Multiply-accumulate instructions	MAC	reg1, reg2, reg3, reg4	4	2	5	5
	MACU	reg1, reg2, reg3, reg4	4	2	5	5

Table C-1. Clock Requirements for Basic Instructions (2/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks		
				issue	repeat	latency
Arithmetic operation instructions	ADD	reg1, reg2	2	1	1	1
	ADD	imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
	CMP	imm5, reg2	2	1	1	1
	MOV	reg1, reg2	2	1	1	1
	MOV	imm5, reg2	2	1	1	1
	MOV	imm32, reg1	4	1	1	1
	MOVEA	imm16, reg1, reg2	4	1	1	1
	MOVHI	imm16, reg1, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
Conditional operation instructions	ADF	cccc, reg1, reg2, reg3	4	1	1	1
	SBF	cccc, reg1, reg2, reg3	4	1	1	1
Saturated operation instructions	SATADD	reg1, reg2	2	1	1	1
	SATADD	imm5, reg2	2	1	1	1
	SATADD	reg1, reg2, reg3	4	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
	SATSUB	reg1, reg2, reg3	4	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
Logic operation instructions	SATSUBR	reg1, reg2	2	1	1	1
	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
Data manipulation instructions	XORI	imm16, reg1, reg2	4	1	1	1
	BSH	reg2, reg3	4	1	1	1
	BSW	reg2, reg3	4	1	1	1
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
	CMOV	cccc, imm5, reg2, reg3	4	1	1	1
	HSH	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
	SAR	imm5, reg2	2	1	1	1
	SAR	reg1, reg2, reg3	4	1	1	1
	SASF	cccc, reg2	4	1	1	1
	SETF	cccc, reg2	4	1	1	1
	SHL	reg1, reg2	4	1	1	1
	SHL	imm5, reg2	2	1	1	1
SHL	reg1, reg2, reg3	4	1	1	1	

Table C-1. Clock Requirements for Basic Instructions (3/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks		
				issue	repeat	latency
Data manipulation instructions	SHR	reg1, reg2	4	1	1	1
	SHR	imm5, reg2	2	1	1	1
	SHR	reg1, reg2, reg3	4	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1
Bit search instructions	SCH0L	reg2, reg3	4	1	1	1
	SCH0R	reg2, reg3	4	1	1	1
	SCH1L	reg2, reg3	4	1	1	1
	SCH1R	reg2, reg3	4	1	1	1
Divide instructions	DIV	reg1, reg2, reg3	4	36	36	36
	DIVH	reg1, reg2	2	36	36	36
	DIVH	reg1, reg2, reg3	4	36	36	36
	DIVHU	reg1, reg2, reg3	4	35	35	35
	DIVU	reg1, reg2, reg3	4	35	35	35
High-speed divide instructions	DIVQ	reg1, reg2, reg3	4	N+7 ^{Note 4}	N+7 ^{Note 4}	N+7 ^{Note 4}

Table C-1. Clock Requirements for Basic Instructions (4/4)

Instruction Type	Mnemonic	Operand	Byte Count	No. of Execution Clocks		
				issue	repeat	latency
Special instruction	FETRAP	vector	2	3	3	3
	HALT	–	4	Undefined	Undefined	Undefined
	LDSR	reg2, regID	4	1	1	1
	NOP	–	2	1	1	1
	PREPARE	list12, imm5	4	$n+1$ ^{Note 3}	$n+1$ ^{Note 3}	$n+1$ ^{Note 3}
	PREPARE	list12, imm5, sp	4	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}
	PREPARE	list12, imm5, imm16	4	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}
	PREPARE	list12, imm5, imm16<<16	4	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}
	PREPARE	list12, imm5, imm32	4	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}	$n+2$ ^{Note 3}
	RETI	–	4	3	3	3
	RIE	–	2	3	3	3
	RIE	–	4	3	3	3
	STSR	regID, reg2	4	1	1	1
	SWITCH	reg1	2	5	5	5
	SYNCE	–	2	Undefined	Undefined	Undefined
	SYNCM	–	2	Undefined	Undefined	Undefined
	SYNCP	–	2	Undefined	Undefined	Undefined
	SYSCALL	vector8	4	5	5	5
	TRAP	vector5	4	3	3	3
	Undefined instruction code (operates as RIE instruction)			4	3	3

- Notes**
1. When there are wait states (+ number of read access wait states)
 2. Add one (+ 1) when an instruction replaces the previous contents of PSW register.
 3. n is the total number of registers specified in list x (depends on the number of wait states. When there are no wait states, n matches with the number of registers specified in list x).
 4. $N = (\text{Number of valid bits of dividend}) - (\text{Number of valid bits of divisor})$
However, if N is 0 or below, then $N = 1$.

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

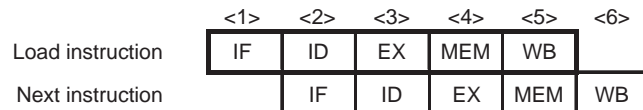
Symbol	Description
issue	When next instruction is executed immediately after previous instruction
repeat	When same instruction is executed again immediately after its first execution
latency	When execution result of the current instruction is used by the immediate next instruction

C.3 Pipeline for Basic Instructions

C.3.1 Load instructions

[Target instructions] LD.B, LD.H, LD.W, LD.BU, LD.HU, SLD.B, SLD.BU, SLD.H, SLD.HU, and SLD.W

[Pipeline]

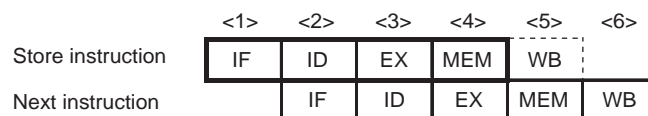


[Description] The pipeline has five stages: the IF, ID, EX, MEM, and WB stages. However, if an instruction that uses the execution result is placed immediately after the load instruction, a data wait period may be generated.

C.3.2 Store instructions

[Target instructions] ST.B, ST.H, ST.W, SST.B, SST.H, and SST.W

[Pipeline]



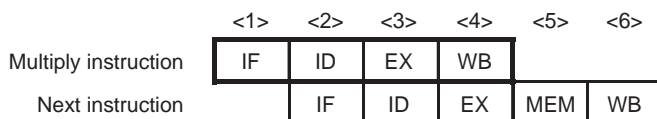
[Description] This pipeline has five stages: the IF, ID, EX, MEM, and WB stages. However, since data cannot be written to registers, nothing is done at the WB stage.

C.3.3 Multiply instructions

(1) Halfword data multiply instruction

[Target instructions] MULH and MULHI

[Pipeline]



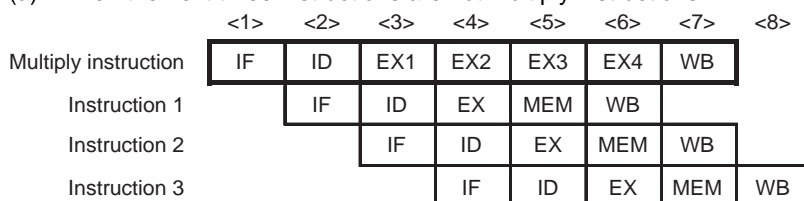
[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

(2) Word data multiply instruction

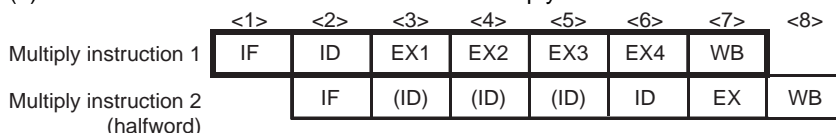
[Target instructions] MUL and MULU

[Pipeline]

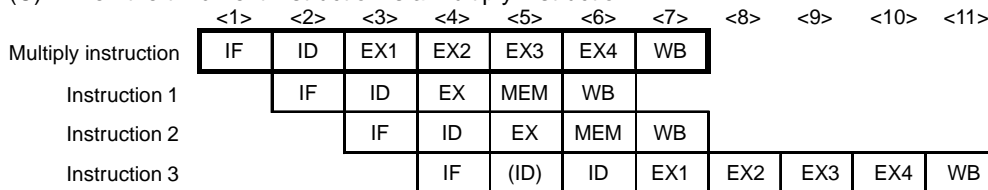
(a) When the next three instructions are not multiply instructions



(b) When the next instruction is a halfword multiply instruction



(c) When the third next instruction is a multiply instruction



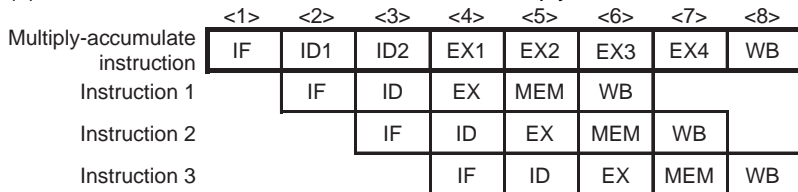
[Description] The pipeline has seven stages: the IF, ID, EX1, EX2, EX3, EX4, and WB stages. If an instruction that uses the execution result is placed immediately after the multiply instruction, a data wait period may be generated.

C.3.4 Multiply-accumulate instructions

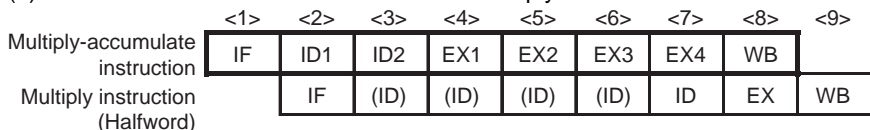
[Target instructions] MAC and MACU

[Pipeline]

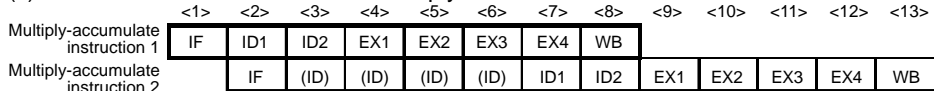
(a) When the next three instructions are not multiply-accumulate instructions



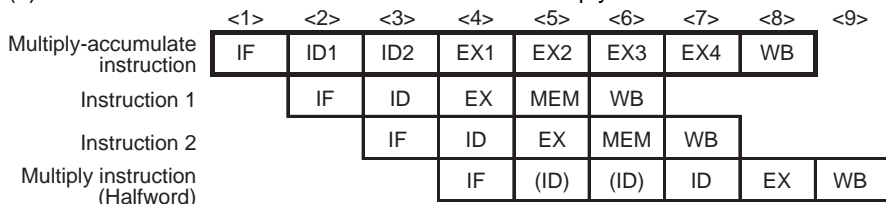
(b) When the next instruction is a halfword multiply instruction



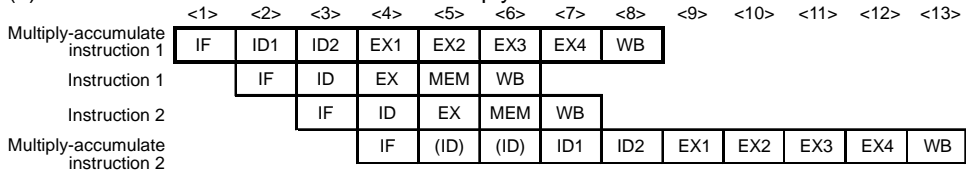
(c) When the next instruction is a multiply-accumulate instruction



(d) When the third next instruction is a halfword multiply instruction



(e) When the third next instruction is a multiply-accumulate instruction

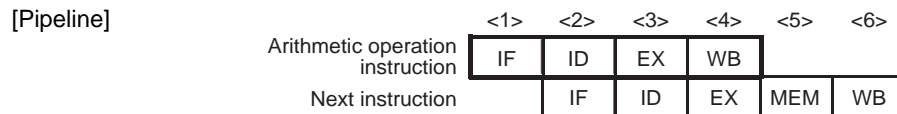


[Description]

The pipeline has eight stages: the IF, ID1, ID2, EX1, EX2, EX3, EX4, and WB stages. If an instruction that uses the execution result is placed immediately after the multiply instruction, a data wait period may be generated.

C.3.5 Arithmetic operation instructions

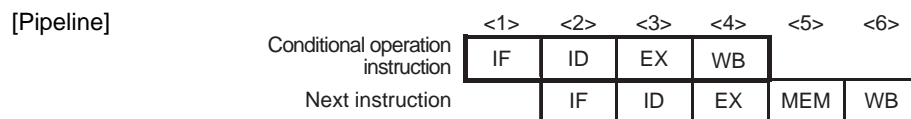
[Target instructions] ADD, ADDI, CMP, MOV, MOVEA, MOVHI, SUB, and SUBR



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.6 Conditional operation instructions

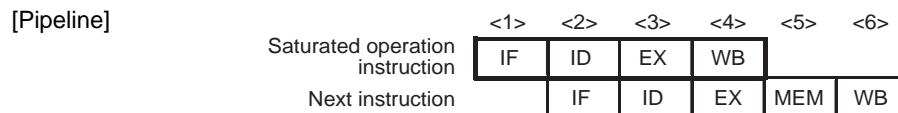
[Target instructions] ADF and SBF



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.7 Saturated operation instructions

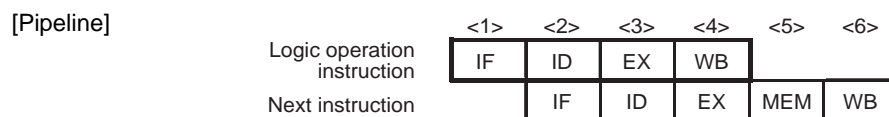
[Target instructions] SATADD, SATSUB, SATSUBI, and SATSUBR



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.8 Logic operation instructions

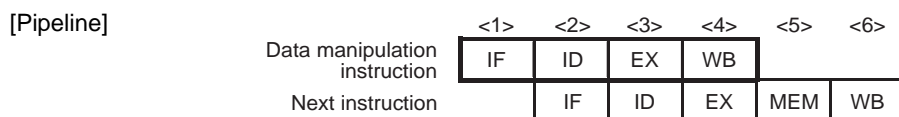
[Target instructions] AND, ANDI, NOT, OR, ORI, TST, XOR, and XORI



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.9 Data manipulation instructions

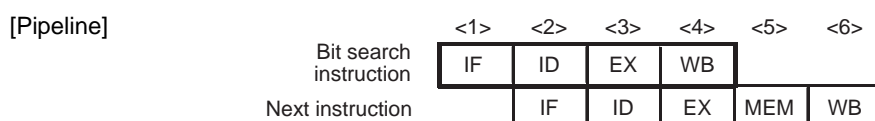
[Target instructions] BSH, BSW, CMOV, HSH, HSW, SAR, SASF, SETF, SHL, SHR, SXB, SXH, ZXB, and ZXH



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.10 Bit search instructions

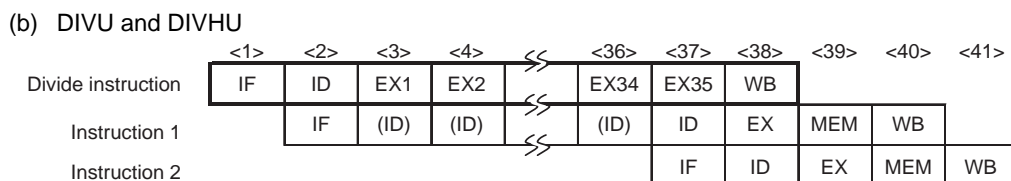
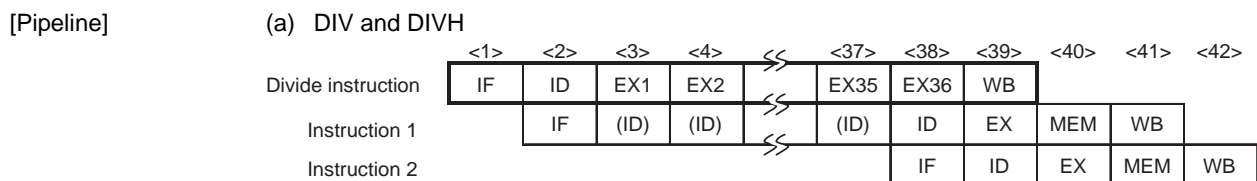
[Target instructions] SCH0L, SCH0R, SCH1L, and SCH1R



[Description] The pipeline has four stages: the IF, ID, EX, and WB stages.

C.3.11 Divide instructions

[Target instructions] DIV, DIVH, DIVHU, and DIVU



[Description] For a DIV or DIVH instruction, the pipeline has 39 stages: IF, ID, EX1 to EX36 (ordinary EX stage), and WB, and for the DIVU and DIVHU instructions, it has 38 stages: IF, ID, EX1 to EX35 (ordinary EX stage), and WB.

Remark If an interrupt occurs during execution of a divide instruction, execution is halted and the interrupt is handled using the start address of this instruction as the return address. After the interrupt has been handled, this instruction is restarted. In such cases, the values prior to execution of this instruction are retained in general-purpose register reg1 and general-purpose register reg2.

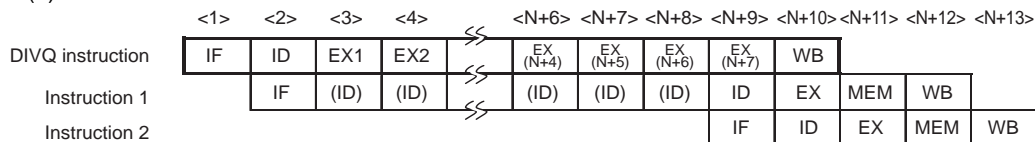
C.3.12 High-speed divide instructions

This instruction automatically determines the minimum number of steps required for the operation.

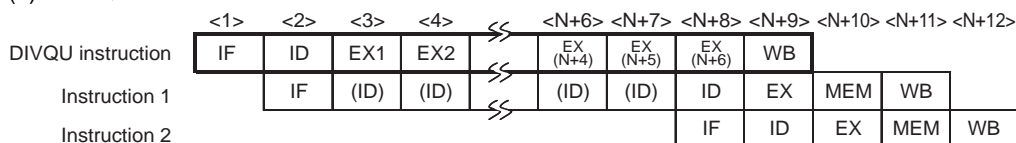
[Target instructions] DIVQ and DIVQU

[Pipeline]

(a) DIVQ



(b) DIVQU



[Description]

For the DIVQ instruction, the pipeline has N + 10 stages: IF, ID, EX1 to EX(N + 7), and WB, and for the DIVQU instruction, it has N + 9 stages: IF, ID, EX1 to EX(N+6), and WB.

Remark N = (Number of valid bits of dividend) – (Number of valid bits of divisor)

However, if N is negative, it is assumed that N = 1.

In addition, if an interrupt occurs during execution of a divide instruction, execution is halted and the interrupt is handled using the start address of this instruction as the return address. After the interrupt has been handled, this instruction is restarted. In such cases, the values prior to execution of this instruction are retained in general-purpose register reg1 and general-purpose register reg2.

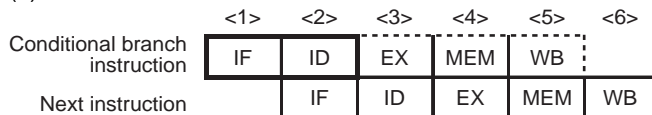
C.3.13 Branch instructions

(1) Conditional branch instruction (except BR instruction)

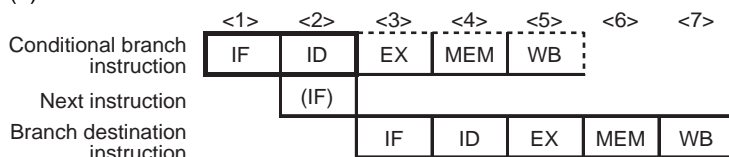
[Target instructions] Bcond instruction

[Pipeline]

(a) When condition has not been met



(b) When condition has been met



(IF): Invalid instruction fetch

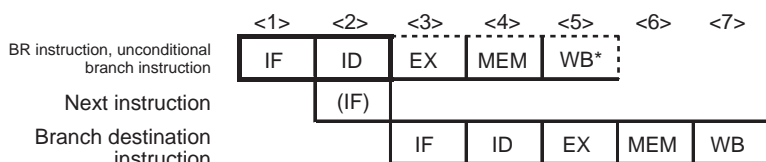
[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. Since the branch destination is set at the ID stage, nothing occurs at the EX stage, MEM stage, or WB stage.

(2) BR instruction and unconditional branch instructions (except JMP instruction)

[Target instructions] BR, JARL, and JR instruction

[Pipeline]



(IF): Invalid instruction fetch

WB*: No operation if JR or BR instruction, but writes back to return PC if JARL instruction.

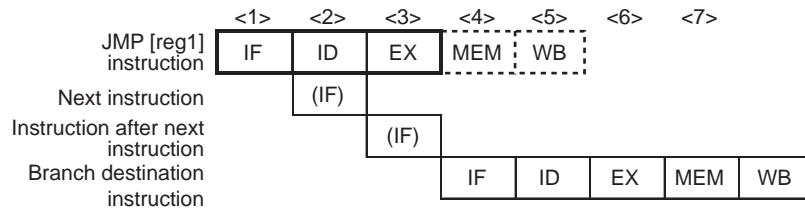
[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. Since the branch destination is set at the ID stage, nothing occurs at the EX stage, MEM stage, or WB stage. However, in the case of the JARL instruction, a writeback to return PC occurs at the WB stage. Also, the IF is invalid for the instruction after the branch instruction.

(3) JMP instructions

(a) JMP [reg1] instruction

[Pipeline]



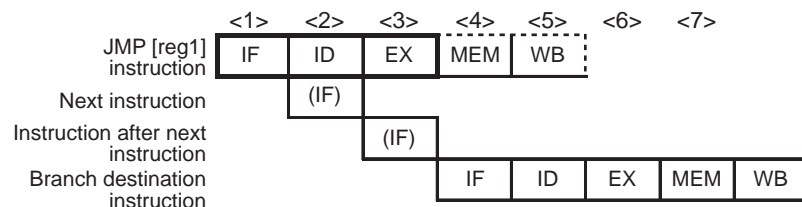
(IF): Invalid instruction fetch

[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. Since the branch destination is set at the EX stage, nothing occurs at the MEM stage or WB stage.

(b) JMP dip32 [reg1] instruction

[Pipeline]



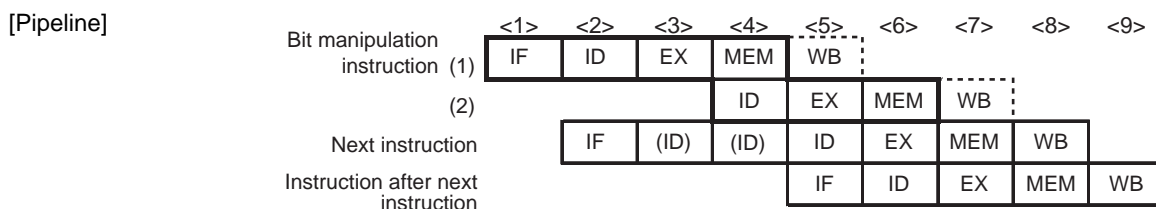
(IF): Invalid instruction fetch

[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. Since the branch destination is set at the EX stage, nothing occurs at the MEM stage or WB stage.

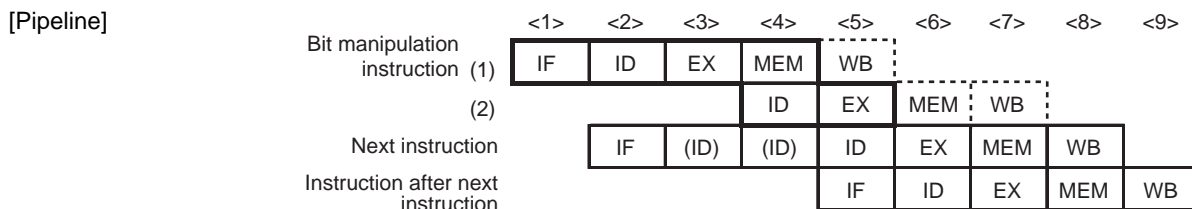
C.3.14 Bit manipulation instructions

(1) CLR1, NOT1, and SET1 instructions



[Description] The pipeline has seven stages: IF, ID, EX1, MEM, EX2 (ordinary stage), MEM, and WB stages. Since no data is written to registers, nothing occurs at the WB stage. This instruction executes memory access as read-modify-write operations, and requires a total of two clock cycles at the EX stage and two clock cycles at the MEM stage.

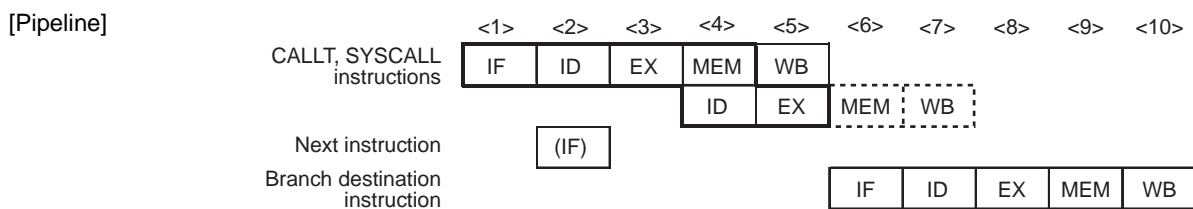
(2) TST1 instruction



[Description] The pipeline has seven stages: IF, ID, EX1, MEM, EX2 (ordinary stage), MEM, and WB stages. Since no data is written to registers and there is no second memory access, nothing occurs at the second MEM stage and WB stage. This instruction requires a total of two clock cycles.

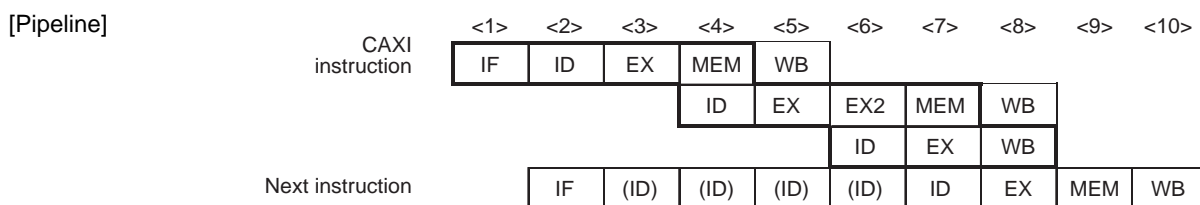
C.3.15 Special instructions

(1) CALLT and SYSCALL instruction



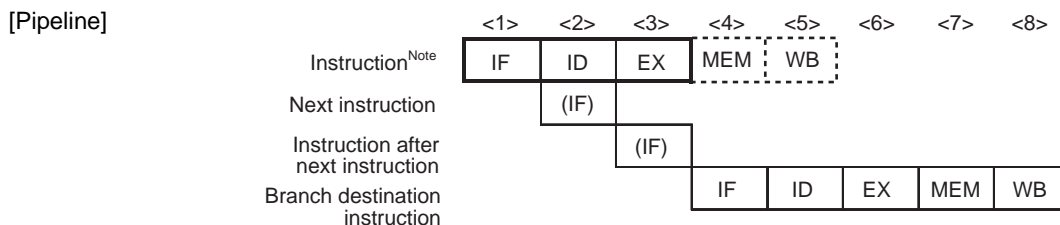
[Description] The pipeline has seven stages: IF, ID, EX1, MEM, EX2, MEM, and WB stages. Since there is no memory access at the second MEM and WB stages, no data is written to registers, and so nothing occurs.

(2) CAXI instruction



[Description] The pipeline has eight stages: IF, ID, EX1, MEM, EX2, EX3, MEM, and WB stages.

(3) CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions



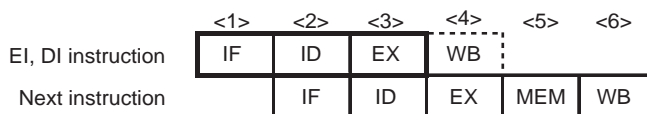
(IF): Invalid instruction fetch

Note CTRET, EIRET, FERET, FETRAP, RETI, RIE, and TRAP instructions

[Description] The pipeline has five stages: IF, ID, EX, MEM, and WB stages. The branch destination is determined at the EX stage, and nothing occurs at the MEM stage and WB stage.

(4) DI, EI, and LDSR instructions

[Pipeline]

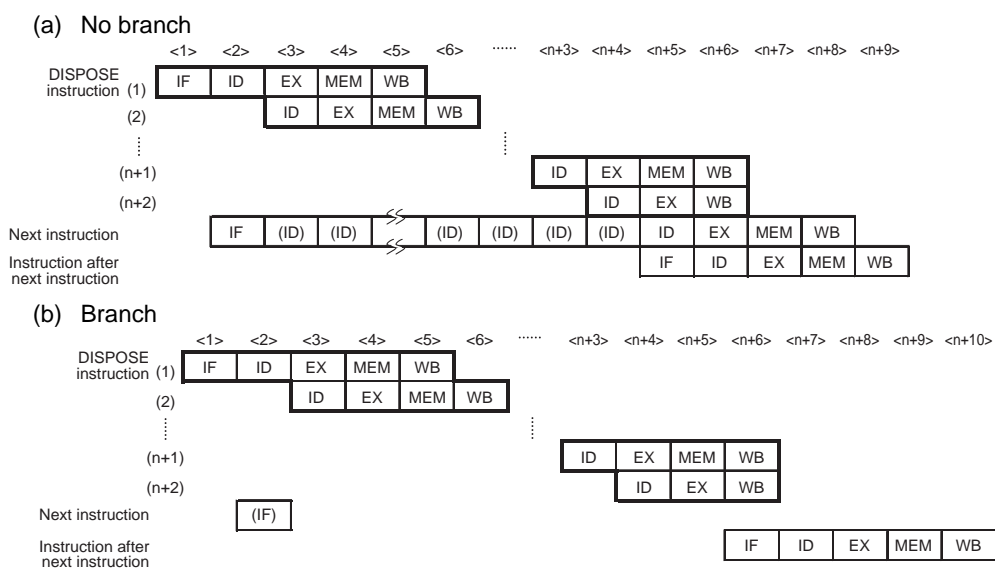


[Description]

The pipeline has four stages: IF, ID, EX, and WB stages. However, there is no memory access at the WB stage, so no data is written to registers and nothing occurs.

(5) DISPOSE instruction

[Pipeline]



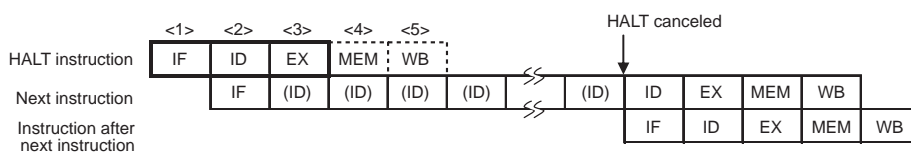
Remark n is the number of registers specified by the register list (list12).

[Description]

The pipeline has “n + 5” stages: IF, ID, EX, MEM (n + 1 time), and WB stages (n: register list number). The MEM stage requires n + 1 cycles.

(6) HALT instruction

[Pipeline]

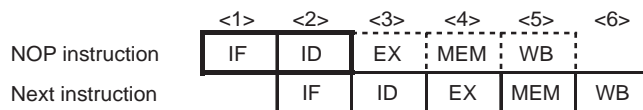


[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. Since there is no memory access or writing of data to registers, nothing occurs at the MEM stage and WB stage. Also, the ID stage is delayed at the next instruction until HALT mode is cleared.

(7) NOP instruction

[Pipeline]

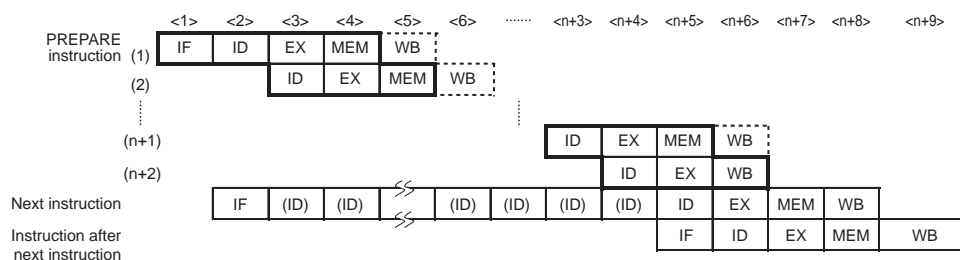


[Description]

The pipeline has five stages: IF, ID, EX, MEM, and WB stages. However, since no operations, memory access, or writing of data to registers is performed, nothing occurs at the EX stage, MEM stage, and WB stage.

(8) PREPARE instruction

[Pipeline]



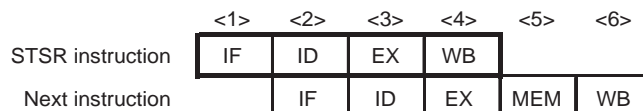
Remark n is the number of registers specified in the register list (list12).

[Description]

The pipeline has “n + 5” stages: IF, ID, EX, MEM (n + 1 time), and WB stages (n: register list number). The MEM stage requires n + 1 cycles. However, since no data is written to registers at the WB stage, nothing occurs.

(9) STSR instruction

[Pipeline]

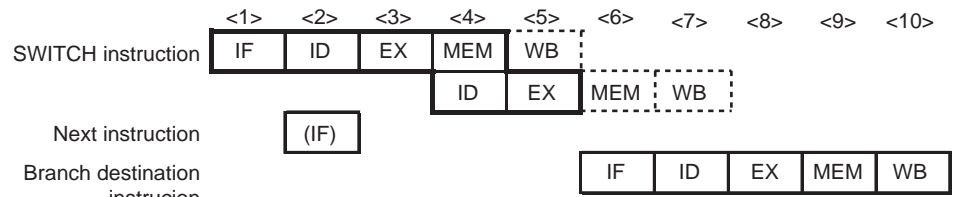


[Description]

The pipeline has four stages: IF, ID, EX, and WB stages. If an STSR instruction that uses the same register is placed immediately after the LDSR instruction, a wait for data alignment will occur.

(10) SWITCH instruction

[Pipeline]



(IF): Invalid instruction fetch

[Description]

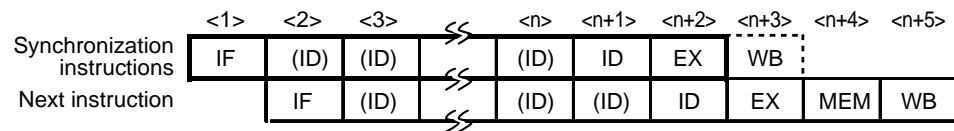
The pipeline has seven stages: IF, ID, EX1 (ordinary EX stage), MEM, EX2, MEM, and WB stages. However, there is no memory access or writing of data to registers at the second MEM and WB stages, so nothing occurs.

(11) Synchronization instructions

[Target instructions]

SYNCP and SYNCP

[Pipeline]



Note n is an undefined value.

[Description]

The synchronization instructions are not issued until processing of all instructions held pending by the CPU has been completed. Since no data is written to registers at the WB stage, nothing occurs.

APPENDIX D DIFFERENCES BETWEEN V850E2S CPU AND OTHER CPUS

D.1 Difference Between V850E2 and V850E2M

(1/3)

Item		V850E2S	V850E2M	V850E2
Instructions (including operands)	ADF cccc, reg1, reg2, reg3	Provided		
	HSH reg2, reg3			
	JARL disp32, reg1			
	JMP disp32, [reg1]			
	JR disp32			
	MAC reg1, reg2, reg3, reg4			
	MACU reg1, reg2, reg3, reg4			
	SAR reg1, reg2, reg3			
	SATADD reg1, reg2, reg3			
	SATSUB reg1, reg2, reg3			
	SBF cccc, reg1, reg2, reg3			
	SCH0L reg1, reg2			
	SCH0R reg1, reg2			
	SCH1L reg1, reg2			
	SCH1R reg1, reg2			
	SHL reg1, reg2, reg3			
	SHR reg1, reg2, reg3			
	CAXI [reg1], reg2, reg3	Provided		Not provided
	DIVQ reg1, reg2, reg3			
	DIVQU reg1, reg2, reg3			
	EIRET			
	FERET			
	FETRAP vector4			
	RIE			
	SYNCM			
	SYNCP			
	SYNCE			
	SYSCALL vector8			
	LD.B disp23 [reg1], reg3			
	LD.BU disp23 [reg1], reg4			
	LD.H disp23 [reg1], reg3			
	LD.HU disp23 [reg1], reg3			
	LD.W disp23 [reg1], reg3			
ST.B reg3, disp23 [reg1]				
ST.H reg3, disp23 [reg2]				
ST.W reg3, disp23 [reg3]				

(2/3)

Item		V850E2S	V850E2M	V850E2
Instructions (including operands)	Floating to point operation exception	Not provided	Provided	Not provided
Number of instruction execution clocks		Varies among certain instructions.		
Program area		64 MB	4 GB ^{Note 2}	512 MB
Valid bits in program counter (PC)		32 bits ^{Note 1}	32 bits ^{Note 2}	Lower 29 bits
Data area		64 MB	4 GB	
System register bank		Provided		Not provided
Main bank		Provided		Provided ^{Note 3}
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		Not Provided	Provided	Not Provided
FPEC				
User 0 bank		Provided		Not Provided

- Notes**
1. Instruction addressing range is a 64 MB. A value resulting from a sign-extension of bit 25 of EIPC is automatically set to bits 31 to 26.
 2. 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.
 3. Bank configuration is not employed and only system registers equivalent to the main bank are available.

(3/3)

Item		V850E2S	V850E2M	V850E2
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		
	Memory protection exception	Provided (30H)		Not provided
	Floating to point operation exception	Not provided	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		5 stages	7 stages	
		Pipeline flow varies for each instruction.		
Debug functions		Functions differ.		

Note Some specifications such as exception handler addresses and exception code are different.

APPENDIX E INSTRUCTION INDEX

E.1 Basic Instructions

[A]	[F]	[N]	
ADD 65	FERET.....94	NOP 118	SLD.BU151
ADDI 66	FETRAP95	NOT 119	SLD.H.....152
ADF..... 67		NOT1 120	SLD.HU153
AND 68	[H]		SLD.W154
ANDI 69	HALT96	[O]	SST.B.....155
	HSH.....97	OR 122	SST.H.....156
[B]	HSW98	ORI 123	SST.W157
Bcond..... 70			ST.B158
BSH 72	[J]	[P]	ST.H159
BSW..... 73	JARL.....99	PREPARE..... 124	ST.W160
	JMP 101		STSR.....161
[C]	JR.....102	[R]	SUB162
CALLT..... 74		RETI..... 126	SUBR163
CAXI 75	[L]	RIE..... 128	SWITCH164
CLR1..... 76	LD.B103		SXB165
CMOV 78	LD.BU.....104	[S]	SXH.....166
CMP 80	LD.H105	SAR 129	SYNCE167
CTRET..... 81	LD.HU.....106	SASF 131	SYNCM168
	LD.W107	SATADD 132	SYNCP169
[D]	LDSR.....108	SATSUB..... 134	SYSCALL170
DI 82		SATSUBI..... 135	
DISPOSE 83	[M]	SATSUBR..... 136	[T]
DIV 85	MAC109	SBF 137	TRAP.....172
DIVH 86	MACU.....110	SCH0L 138	TST.....173
DIVHU..... 88	MOV 111	SCH0R..... 139	TST1.....174
DIVQ 89	MOVEA112	SCH1L 140	
DIVQU 90	MOVHI.....113	SCH1R..... 141	[X]
DIVU 91	MUL.....114	SET1..... 142	XOR.....175
	MULH 115	SETF..... 144	XORI.....176
[E]	MULHI116	SHL..... 146	ZXB177
EI 92	MULU 117	SHR 148	ZXH178
EIRET 93		SLD.B 150	

REVISION HISTORY	V850E2S User's Manual: Architecture
------------------	-------------------------------------

Rev.	Date	Description	
		Page	Summary
0.01	Dec 26, 2011	—	First Edition issued
1.00	May 29, 2014	Throughout	Changed symbols of EI level software exception, Reserved instruction exception, FE level software exception and System call exception
		PART 2 CHAPTER 2 REGISTER SET	
		25 to 26	2. 3. 4 PSW, changed the Note, register figure and table
		33	Table 2-3. System Register Bank, changed
		PART 3 CHAPTER 5 SYSTEM REGISTER PROTECTION	
		225	Changed the description
		PART 3 CHAPTER 6 MEMORY PROTECTION	
		233	6. 1. 1 PAnL, changed
		APPENDIX C PIPELINES	
		267	C. 3. 3 (2) Word data multiply instruction - (b) and (c), changed
		268	C. 3. 4 Multiply-accumulate instructions - (b) to (e) , changed
		277	C. 3. 15 (8) PREPARE instruction , changed
		278	C. 3. 15 (11) Synchronization instructions, changed

V850E2S User's Manual: Architecture

Publication Date: Rev.0.01 Dec 26, 2011
Rev.1.00 May 29, 2014

Published by: Renesas Electronics Corporation

**SALES OFFICES****Renesas Electronics Corporation**<http://www.renesas.com>Refer to "<http://www.renesas.com/>" for the latest and detailed information.**Renesas Electronics America Inc.**2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130**Renesas Electronics Canada Limited**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: +44-1628-585-100, Fax: +44-1628-585-900**Renesas Electronics Europe GmbH**Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327**Renesas Electronics (China) Co., Ltd.**Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679**Renesas Electronics (Shanghai) Co., Ltd.**Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999**Renesas Electronics Hong Kong Limited**Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2686-9022/9044**Renesas Electronics Taiwan Co., Ltd.**13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670**Renesas Electronics Singapore Pte. Ltd.**80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300**Renesas Electronics Malaysia Sdn.Bhd.**Unit 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.**12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

V850E2S



Renesas Electronics Corporation

R01US0037EJ0100