

To our customers,

---

## Old Company Name in Catalogs and Other Documents

---

On April 1<sup>st</sup>, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: <http://www.renesas.com>

April 1<sup>st</sup>, 2010  
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (<http://www.renesas.com>)

Send any inquiries to <http://www.renesas.com/inquiry>.

## Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. 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 categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
  - “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.
  - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
  - “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. 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.
9. 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 system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. 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.
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.

# **Preliminary User's Manual**

## **V850E2**

### **32-bit Microprocessor Core**

### **Architecture**

---

[MEMO]

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.

These commodities, technology or software, must be exported in accordance with the export administration regulations of the exporting country. Diversion contrary to the law of that country is prohibited.

- **The information contained in this document is being issued in advance of the production cycle for the product. The parameters for the product may change before final production or NEC Electronics Corporation, at its own discretion, may withdraw the product prior to its production.**
- **Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific". The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics products depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  - "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.
  - "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  - "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

- (1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
- (2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

# Regional Information

Some information contained in this document may vary from country to country. Before using any NEC Electronics product in your application, please contact the NEC Electronics office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

## [GLOBAL SUPPORT]

<http://www.necel.com/en/support/support.html>

### **NEC Electronics America, Inc. (U.S.)**

Santa Clara, California  
Tel: 408-588-6000  
800-366-9782

### **NEC Electronics (Europe) GmbH**

Duesseldorf, Germany  
Tel: 0211-65030

- **Sucursal en España**

Madrid, Spain  
Tel: 091-504 27 87

- **Succursale Française**

Vélizy-Villacoublay, France  
Tel: 01-30-67 58 00

- **Filiale Italiana**

Milano, Italy  
Tel: 02-66 75 41

- **Branch The Netherlands**

Eindhoven, The Netherlands  
Tel: 040-244 58 45

- **Tyskland Filial**

Taeby, Sweden  
Tel: 08-63 80 820

- **United Kingdom Branch**

Milton Keynes, UK  
Tel: 01908-691-133

### **NEC Electronics Hong Kong Ltd.**

Hong Kong  
Tel: 2886-9318

### **NEC Electronics Hong Kong Ltd.**

Seoul Branch  
Seoul, Korea  
Tel: 02-558-3737

### **NEC Electronics Shanghai Ltd.**

Shanghai, P.R. China  
Tel: 021-5888-5400

### **NEC Electronics Taiwan Ltd.**

Taipei, Taiwan  
Tel: 02-2719-2377

### **NEC Electronics Singapore Pte. Ltd.**

Novena Square, Singapore  
Tel: 6253-8311

J04.1

# PREFACE

<b>Target Readers</b>	This manual is intended for users who wish to understand the functions of the V850E2 CPU core for designing application systems using the V850E2 CPU core.	
<b>Purpose</b>	This manual is intended for users to understand the architecture of the V850E2 CPU core described in the Organization below.	
<b>Organization</b>	This manual contains the following information: <ul style="list-style-type: none"><li>• Register set</li><li>• Data type</li><li>• Instruction format and instruction set</li><li>• Interrupts and exceptions</li><li>• Pipeline operations</li><li>• Shifting to debug mode</li></ul>	
<b>How to Use this Manual</b>	It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.  To learn about the hardware functions, → Read Hardware User's Manual of each product.  To learn about the functions of a specific instruction in detail, → Read <b>CHAPTER 5 INSTRUCTIONS</b> .	
<b>Conventions</b>	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)
	<b>Note:</b>	Footnote for item marked with <b>Note</b> in the text
	<b>Caution:</b>	Information requiring particular attention
	<b>Remark:</b>	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): $2^{30} = 1,024^3$



# Contents

<b>CHAPTER 1 OVERVIEW .....</b>	<b>13</b>
<b>1.1 Features .....</b>	<b>14</b>
<b>1.2 Internal Configuration .....</b>	<b>15</b>
<b>CHAPTER 2 REGISTER SET .....</b>	<b>16</b>
<b>2.1 Program Registers .....</b>	<b>17</b>
<b>2.2 System Registers .....</b>	<b>19</b>
2.2.1 Interrupt Status-saving Registers (EIPC, EIPSW) .....	21
2.2.2 NMI Status-saving Registers (FEPC and FEPSW) .....	22
2.2.3 Exception Cause Register (ECR) .....	22
2.2.4 Program status word (PSW) .....	23
2.2.5 CALLT Status-saving Registers (CTPC and CTPSW) .....	25
2.2.6 Exception/Debug Trap Status-saving Registers (DBPC and DBPSW) .....	26
2.2.7 CALLT base pointer (CTBP) .....	27
2.2.8 Debug Interface register (DIR) .....	27
2.2.9 Breakpoint Control registers 0 to 3 (BPC0 to BPC3) .....	31
2.2.10 Program ID register (ASID) .....	34
2.2.11 Breakpoint Address Setup registers 0 to 3 (BPAV0 to BPAV3) .....	34
2.2.12 Breakpoint Address Mask registers 0 to 3 (BPAM0 to BPAM3) .....	35
2.2.13 Breakpoint Data Setup registers 0 to 3 (BPDV0 to BPDV3) .....	36
2.2.14 Breakpoint Data Mask Registers 0 to 3 (BPDM0 to BPDM3) .....	37
<b>CHAPTER 3 DATA TYPE .....</b>	<b>38</b>
<b>3.1 Data Format .....</b>	<b>38</b>
<b>3.2 Data Representation .....</b>	<b>40</b>
3.2.1 Integer .....	40
3.2.2 Unsigned integer .....	40
3.2.3 Bit .....	40
<b>3.3 Data Alignment .....</b>	<b>41</b>

<b>CHAPTER 4 ADDRESS SPACE.....</b>	<b>42</b>
<b>4. 1 Memory Map .....</b>	<b>43</b>
<b>4. 2 Addressing Modes .....</b>	<b>44</b>
4. 2. 1 Instruction address.....	44
4. 2. 2 Operand address .....	47
<b>CHAPTER 5 INSTRUCTIONS.....</b>	<b>49</b>
<b>5. 1 Instruction Formats .....</b>	<b>49</b>
<b>5. 2 Outline of Instructions.....</b>	<b>53</b>
<b>5. 3 Instruction Set .....</b>	<b>58</b>
ADD .....	61
ADDI .....	62
ADF .....	63
AND .....	64
ANDI .....	65
Bcond .....	66
BSH .....	68
BSW .....	69
CALLT .....	70
CLR1 .....	71
CMOV .....	72
CMP .....	74
CTRET .....	75
DBRET .....	76
DBTRAP .....	77
DI .....	78
DISPOSE .....	79
DIV .....	81
DIVH .....	82
DIVHU .....	84
DIVU .....	85
EI .....	86
HALT .....	87
HSH .....	88
HSW .....	89
JARL .....	90

JMP .....	91
JR .....	92
LD.B .....	93
LD.BU .....	94
LD.H .....	95
LD.HU .....	96
LD.W .....	97
LDSR .....	98
MAC .....	99
MACU .....	100
MOV .....	101
MOVEA .....	102
MOVHI .....	103
MUL .....	104
MULH .....	106
MULHI .....	107
MULU .....	108
NOP .....	110
NOT .....	111
NOT1 .....	112
OR .....	113
ORI .....	114
PREPARE .....	115
RETI .....	117
SAR .....	119
SASF .....	120
SATADD .....	121
SATSUB .....	123
SATSUBI .....	124
SATSUBR .....	125
SBF .....	126
SCH0L .....	127
SCH0R .....	128
SCH1L .....	129
SCH1R .....	130
SET1 .....	131
SETF .....	132
SHL .....	134
SHR .....	135

SLD.B .....	136
SLD.BU .....	137
SLD.H .....	138
SLD.HU .....	139
SLD.W .....	140
SST.B .....	141
SST.H .....	142
SST.W .....	143
ST.B .....	144
ST.H .....	145
ST.W .....	146
STSR .....	147
SUB .....	148
SUBR .....	149
SWITCH .....	150
SXB .....	151
SXH .....	152
TRAP .....	153
TST .....	154
TST1 .....	155
XOR .....	156
XORI .....	157
ZXB .....	158
ZXH .....	159

<b>5. 4 Number of Instruction Execution Clock Cycle .....</b>	<b>160</b>
---	------------

## **CHAPTER 6 INTERRUPTS AND EXCEPTIONS..... 165**

<b>6. 1 Interrupt Servicing .....</b>	<b>166</b>
6. 1. 1 Maskable interrupt.....	166
6. 1. 2 Non-maskable interrupt.....	168
<b>6. 2 Exception Processing.....</b>	<b>169</b>
6. 2. 1 Software exception.....	169
6. 2. 2 Exception trap .....	170
6. 2. 3 Debug traps and debug breaks.....	171

6. 3 Interrupt/Exception Processing Return .....	172
6. 3. 1 Interrupt/software exception return .....	172
6. 3. 2 Exception trap, debug trap, and debug break return .....	173
<b>CHAPTER 7 RESET .....</b>	<b>174</b>
7. 1 Post-Reset Register Status .....	174
7. 2 Post-Reset Initialization .....	175
<b>CHAPTER 8 PIPELINE OPERATIONS .....</b>	<b>176</b>
8. 1 Features .....	178
8. 2 Pipeline Flow during Execution of Instructions .....	180
8. 2. 1 Load instructions .....	180
8. 2. 2 Store instructions .....	180
8. 2. 3 Multiplication instructions .....	181
8. 2. 4 Multiplication with addition instructions .....	182
8. 2. 5 Arithmetic operation instructions .....	182
8. 2. 6 Conditional arithmetic instructions .....	183
8. 2. 7 Saturation instructions .....	183
8. 2. 8 Logical operation instructions .....	184
8. 2. 9 Data operation instructions .....	184
8. 2. 10 Bit search instructions .....	185
8. 2. 11 Division instructions .....	185
8. 2. 12 Branch instructions .....	186
8. 2. 13 Bit manipulation instructions .....	188
8. 2. 14 Special instructions .....	189
8. 2. 15 Instructions for debug function .....	194
<b>CHAPTER 9 SHIFTING TO DEBUG MODE .....</b>	<b>195</b>
9. 1 Methods for Shifting to Debug Mode .....	195
9. 2 Caution Points .....	202
<b>APPENDIX A LIST OF INSTRUCTION .....</b>	<b>203</b>
<b>APPENDIX B INSTRUCTION OPCODE MAP .....</b>	<b>219</b>

<b>APPENDIX C</b>	<b>DIFFERENCES WITH ARCHITECTURE OF V850 CPU AND V850E1 CPU.....</b>	<b>224</b>
<b>APPENDIX D</b>	<b>INSTRUCTIONS ADDED FOR V850E2 CPU COMPARED WITH V850 CPU AND V850E1 CPU.....</b>	<b>227</b>
<b>APPENDIX E</b>	<b>LIST OF CAUTION POINTS .....</b>	<b>230</b>
<b>APPENDIX F</b>	<b>INSTRUCTION INDEX.....</b>	<b>234</b>

# CHAPTER 1 OVERVIEW

Real-time control systems are used in a wide range of applications including:

- office equipment, such as HDDs (Hard Disk Drives), PPCs (Plain Paper Copiers), printers, and facsimiles;
- automobile electronics, such as engine control systems and ABSs (Antilock Braking Systems); and
- factory automation equipment, such as NC (Numerical Control) machine tools and various controllers.

The great majority of these systems conventionally employ 8-bit or 16-bit microcontrollers. However, the performance level of these microcontrollers has become inadequate in recent years as control operations have risen in complexity, leading to the development of increasingly complicated instruction sets and hardware designs. As a result the need has arisen for a new generation of microcontrollers operable at much higher frequencies to achieve an acceptable level of performance under today's more demanding requirements.

The V850 Series of microcontrollers was developed to satisfy this need. This series uses RISC architecture that can provide maximum performance with simpler hardware. It allows users to obtain a performance approximately 15 times higher than that of the existing 78K/III Series and 78K/IV Series of CISC single-chip microcontrollers at a lower total cost.

In addition to the basic instructions of conventional RISC CPUs, the V850 Series is provided with special instructions, such as saturate, bit manipulate, and multiply/divide (executed by a hardware multiplier) instructions, which are especially suited for digital servo control systems. Moreover, instruction formats are designed for maximum compiler coding efficiency, allowing the reduction of object code sizes.

The V850E2 CPU has strengthened further the performance of the V850E1 CPU which is the succeeding product of this V850 CPU and the interruption response ability. Furthermore, the pipeline was increase to 7-stage in order to raise the clock frequency of CPU. To increase the throughput of CPU, parallelization of pipeline processing was performed, and the program area has been extended to 512M bytes. Moreover, in order to enable high-speed memory access, the instruction/data cache interface has been built in. Because the instruction codes are upwardly compatible with the V850 CPU or the V850E1 CPU at the object code level, the software resources of common systems can be used unchanged.

## 1.1 Features

### (1) High-performance 32-bit architecture for embedded control

- Number of instructions: 93
- 32-bit general registers: 32
- Load/store instructions in long/short formats
- 3-operand instructions
- 7-stage pipeline of 1 clock cycle per stage
- Hardware interlock on register/flag hazards
- Memory space: 512 M Bytes linear for program space  
4 G Bytes linear for data space

### (2) Special instructions

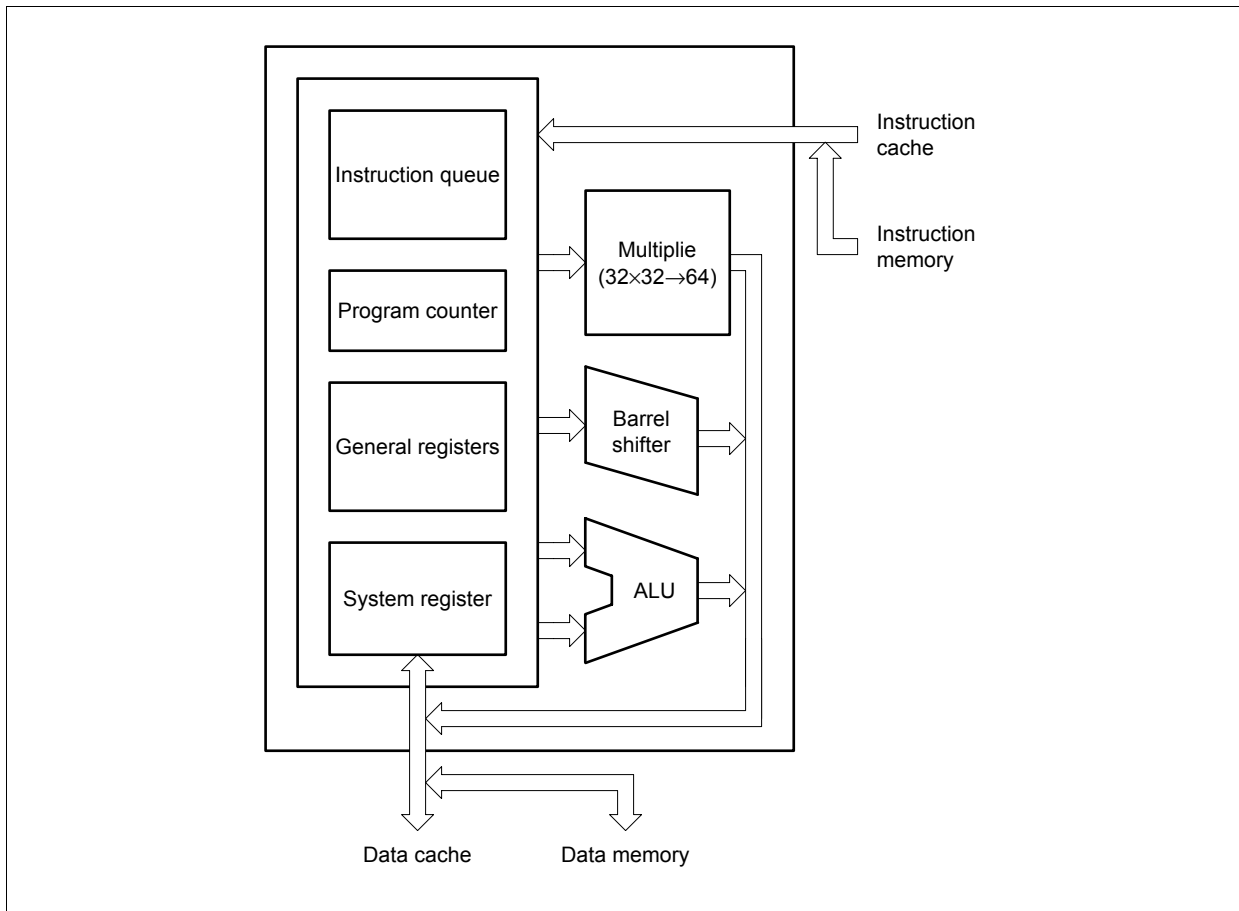
- Saturate operation instructions
- Bit manipulation instructions
- Multiply instructions (On-chip hardware multiplier executing multiplication in 1 clock)
  - 16 bits  $\times$  16 bits  $\rightarrow$  32 bits
  - 32 bits  $\times$  32 bits  $\rightarrow$  32 bits or 64 bits
- MAC operation instructions
  - 32 bits  $\times$  32 bits + 64 bits  $\rightarrow$  64 bits



## 1.2 Internal Configuration

The V850E2 CPU uses a 7-stage pipeline to execute nearly all instructions in one clock cycle. It includes address calculations, arithmetic operations, logical operations, and data transfers. It also contains additional dedicated high-speed features, such as a multiplier (32 × 32 bits) and a barrel shifter (32 bits/clock), to execute complex instructions. Figure 1-1 shows the internal configuration.

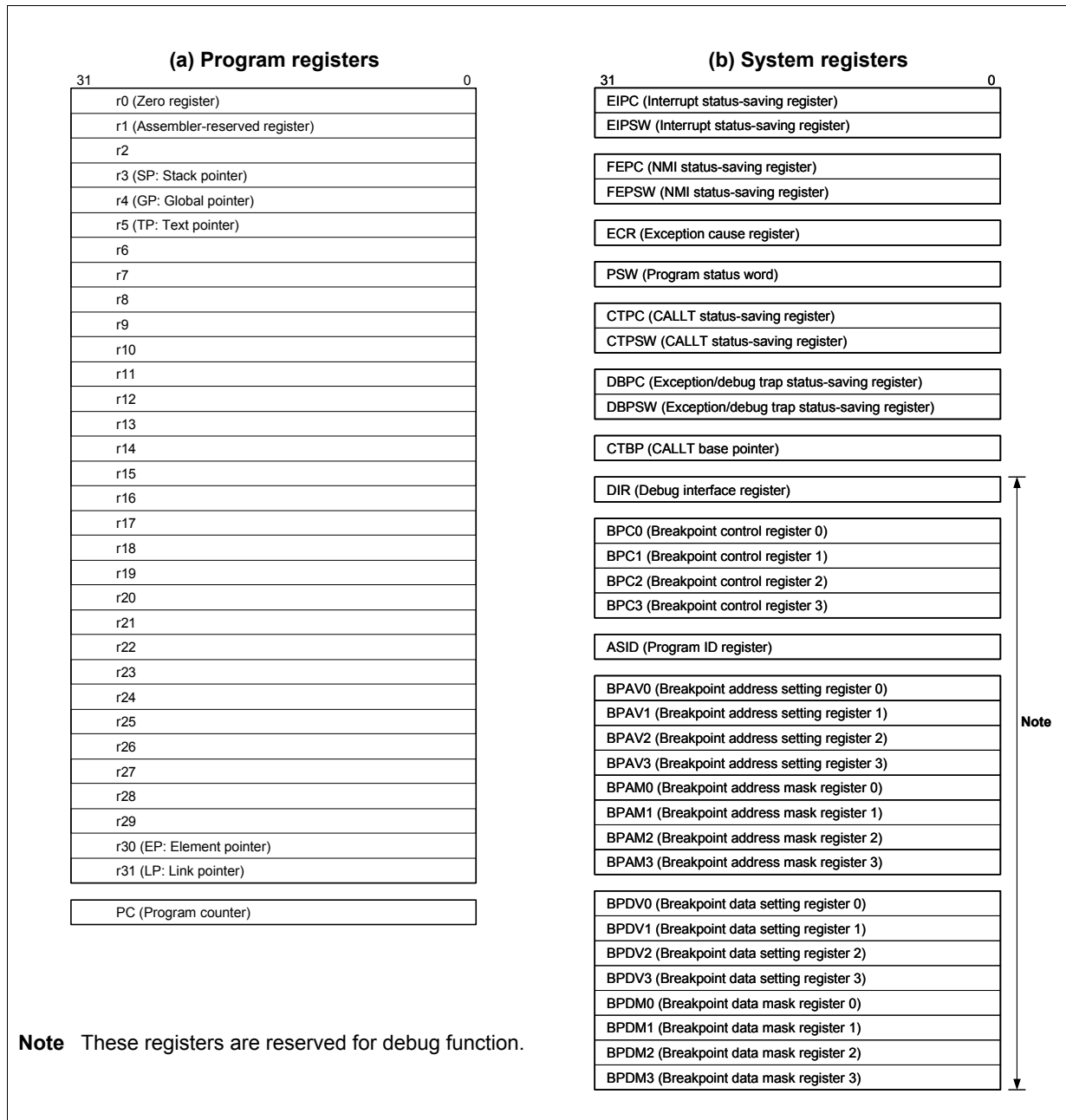
**Figure 1-1. V850E2 CPU Internal Configuration**



# CHAPTER 2 REGISTER SET

The registers can be classified into two types: program registers that can be used for general programming, and system registers that can control the execution environment. All registers are 32-bit wide.

**Figure 2-1. Registers**



## 2. 1 Program Registers

There are general registers (r0 to r31) and program counter (PC) in the program registers.

**Table 2-1. Program Registers**

Program Register	Name	Function	Description
General register	r0	Zero register	Always holds "0."
	r1	Assembler-reserved register	Working register for address generation.
	r2	Address/data variable register (when the real-time OS to be used is not using r2)	
	r3	Stack pointer (SP)	Stack frame generation when function is called.
	r4	Global pointer (GP)	Access global variables in data area.
	r5	Text pointer (TP)	Register for pointing start address of the text area where program code is placed.
	r6 to r29	Address/data variable registers	
	r30	Element pointer (EP)	Base pointer for address generation when memory is accessed.
	r31	Link pointer (LP)	Used when compiler calls function.
Program counter	PC	Holds instruction address during program execution.	

**Remark** For detailed descriptions of r1, r3 to r5, and r31 used by an assembler or C compiler, refer to the **CA850 (C Compiler Package) Assembly Language User's Manual**.

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

Thirty-two general-purpose registers, r0 to r31, are provided. All these registers can be used for data variables or address variables.

However, care must be exercised as follows in using the r0 to r5, r30, and r31 registers.

#### (a) r0, r30

r0 and r30 are implicitly used by instructions.

r0 is a register that always holds 0, and is used for operations using 0 and offset 0 addressing. r30 is used as a base pointer when accessing memory using the SLD and SST instructions.

#### (b) r1, r3 to r5, r31

r1, r3 to r5, and r31 are implicitly used by the assembler and C compiler.

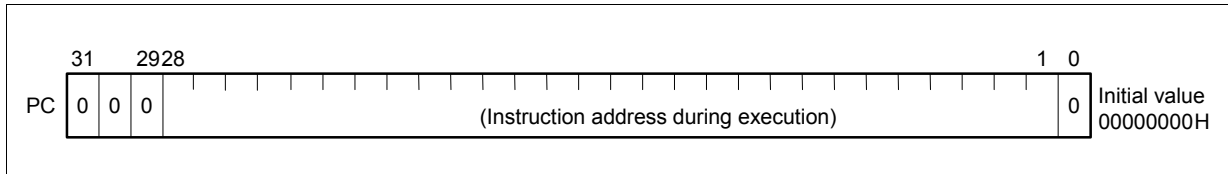
Before using these registers, therefore, their contents must be saved so that they are not lost. The contents must be restored to the registers after the registers have been used.

#### (c) r2

r2 is sometimes used by a real-time OS. When the real-time OS to be used is not using r2, r2 can be used as an address variable register or a data variable register.

**(2) Program counter (PC)**

This register holds an instruction address during program execution. The lower 29 bits of this register are valid, and bits 31 to 29 are reserved for future function expansion (fixed to 0). If a carry occurs from bit 28 to bit 29, it is ignored. Bit 0 is always fixed to 0, and execution cannot branch to an odd address.

**Figure 2-2. Program Counter (PC)**

## 2.2 System Registers

The system registers control the CPU status and hold information on interrupts.

System registers can be read or written by specifying the relevant system register number from the following list using a system register load/store instruction (LDSR or STSR instruction).

**Table 2-2. System Register Numbers**

System register No.	Register name	Operand Specifiability	
		LDSR instruction	STSR instruction
0	Interrupt status-saving register (EIPC)	Yes	Yes
1	Interrupt status-saving register (EIPSW)	Yes	Yes
2	NMI status-saving register (FEPC)	Yes	Yes
3	NMI status-saving register (FEPSW)	Yes	Yes
4	Exception cause register (ECR)	No	Yes
5	Program status word (PSW)	Yes	Yes
6-15	(Nos. reserved for future expansion (operation not guaranteed when accessed))	No	No
16	CALLT status-saving register (CTPC)	Yes	Yes
17	CALLT status-saving register (CTPSW)	Yes	Yes
18	Exception/Debug trap status-saving register (DBPC)	Yes	Yes
19	Exception/Debug trap status-saving register (DBPSW)	Yes	Yes
20	CALLT base pointer (CTBP)	Yes	Yes
21	Debug Interface register (DIR)	Yes <sup>Note</sup>	Yes
22-27	Varies according to channel set via DIR register (See <b>Table 2-3</b> ).	-	-
28-31	(Nos. reserved for future expansion (operation not guaranteed when accessed))	No	No

**Note** Some bits are undefined when read while in user mode (see **2.2.8 Debug Interface register (DIR)**).

**Caution** After bit 0 in the EIPC, FEPC, or CTPC register is set (= 1) by the LDSR instruction, if interrupt processing occurs and a RETI instruction is used to recover, the value of bit 0 is ignored (since the PC bit 0 value is fixed to 0). When setting values to the EIPC, FEPC, or CTPC register, always set an even number (bit 0 = 0) unless there is a particular reason to do otherwise.

**Remark** Yes: Access enabled  
No: Access prohibited

**Table 2-3. System Register List (System Register Nos.: 22 to 27)**

System register No.	Register name	Operand Specifiability	
		LDSR instruction	STSR instruction
22	Breakpoint control register n (BPCn)	Yes	Yes
23	Program ID register (ASID)	Yes	Yes
24	Breakpoint address setting register n (BPAVn)	Yes	Yes
25	Breakpoint address mask register n (BPAMn)	Yes	Yes
26	Breakpoint data setting register n (BPDVn)	Yes	Yes
27	Breakpoint data mask register n (BPDm)	Yes	Yes

**Remark** n = 0 to 3

Yes: Access enabled

## 2. 2. 1 Interrupt Status-saving Registers (EIPC, EIPSW)

Interrupt status-saving registers include the EIPC and EIPSW registers.

When a software exception, maskable interrupt, or runtime error exception has occurred, the value in the program counter (PC) is saved in the EIPC register and the value in the program status word (PSW) is saved in the EIPSW register (when a non-maskable interrupt (NMI) occurs, the value is saved in the NMI status-saving registers (FEPC and FEPSW)).

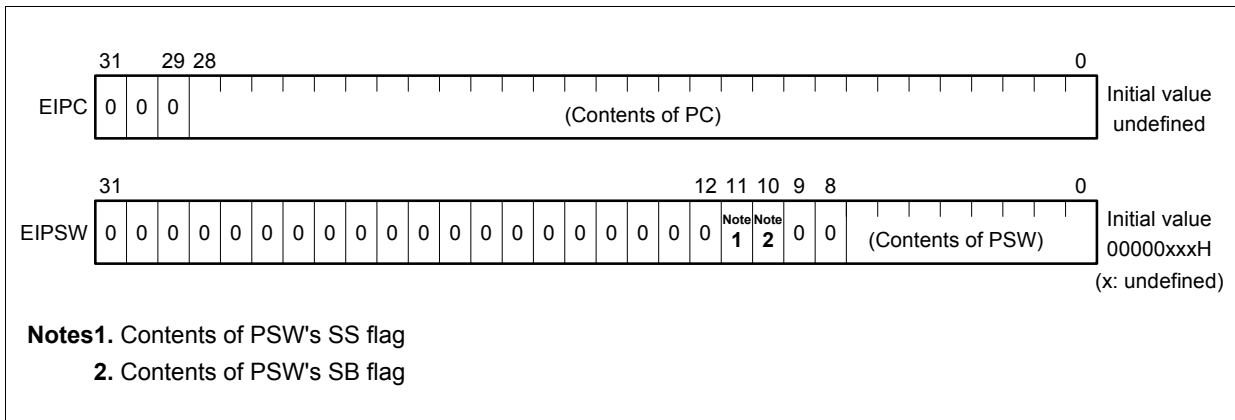
Except for certain instructions, when a software exception, maskable interrupt, or runtime error exception occurs, the address of the instruction after the instruction where the exception or interrupt occurs is stored in the EIPC register (See **Table 6-1. Interrupt/Exception Codes**).

The current PSW value is saved in the EIPSW register.

Because only one pair of interrupt status-saving registers is provided, the contents of these registers must be saved by program when multiple interrupts are enabled.

EIPC register bits 31 to 29 and EIPSW register bits 31 to 12, 9 and 8 are reserved (fixed to 0) for future expanded functions.

**Figure 2-3. Interrupt Status-saving Registers (EIPC and EIPSW)**



## 2. 2. 2 NMI Status-saving Registers (FEPC and FEPSW)

The NMI status-saving registers include the FEPC and FEPSW registers.

When a non-maskable interrupt (NMI) or runtime error exception occurs, the value in the program counter (PC) is saved in the FEPC register and the value in the program status word (PSW) is saved in the FEPSW register. Except for certain instructions, when an NMI or runtime error exception occurs, the address of the instruction after the instruction where the exception or interrupt occurs is stored in the FEPC register (See **Table 6-1. Interrupt/Exception Codes**).

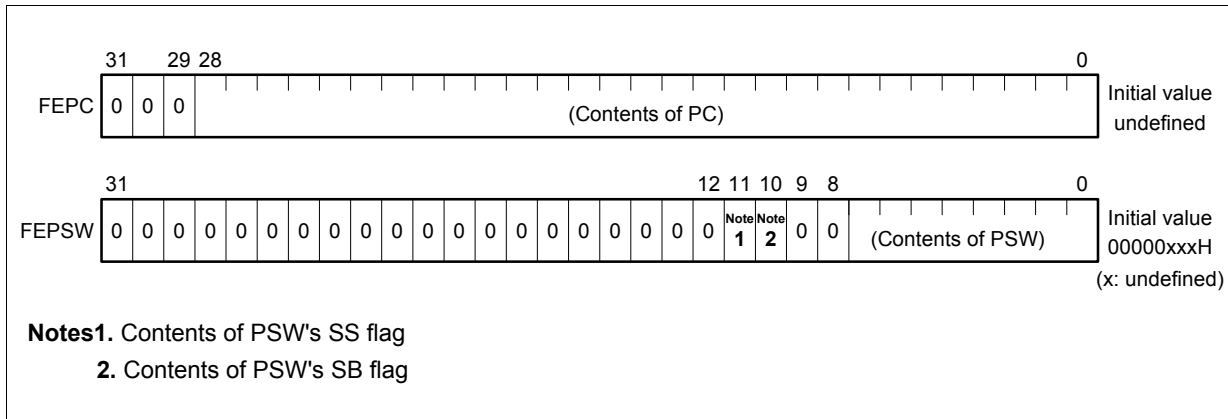
**Interrupt/Exception Codes**.

The current PSW value is saved in the FEPSW register.

Because only one pair of NMI status saving registers is provided, the contents of these registers must be saved by program when multiple interrupts are enabled.

Bits 31 to 29 of FEPC register and bits 31 to 12, 9, and 8 of FEPSW register are reserved for future function expansion (fixed to 0).

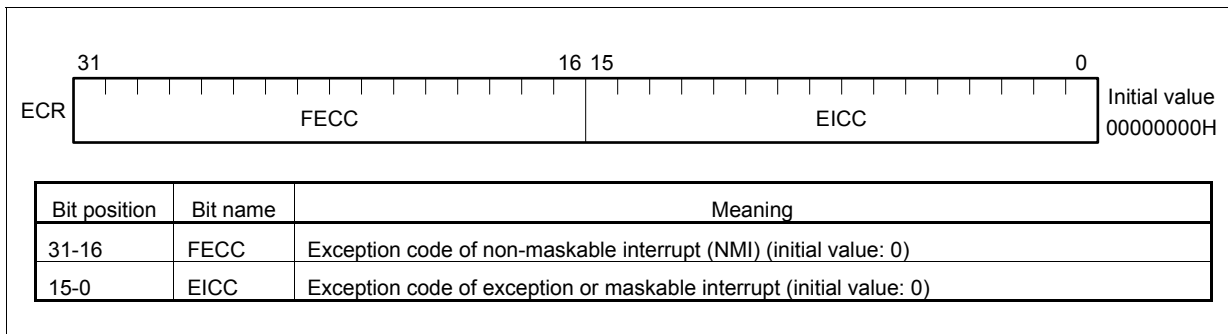
**Figure 2-4. NMI Status-saving Registers (FEPC and FEPSW)**



## 2. 2. 3 Exception Cause Register (ECR)

The Exception Cause register (ECR) stores the factor (code) that has caused an exception or interrupt to occur. The values saved in the ECR register are exception code that are encoded for each interrupt factor (See **Table 6-1. Interrupt/Exception Codes**). Since the ECR register is a read-only register, the LDSR instruction cannot be used to write data to this register.

**Figure 2-5. Exception Cause Register (ECR)**





Bits 31 to 12, 9, and 8 of this register are reserved for future expansion of functions, and writing values other than 0 to them is prohibited. Their values when read is undefined.

### Figure 2-6. Program Status Word (PSW) (1 of 2)

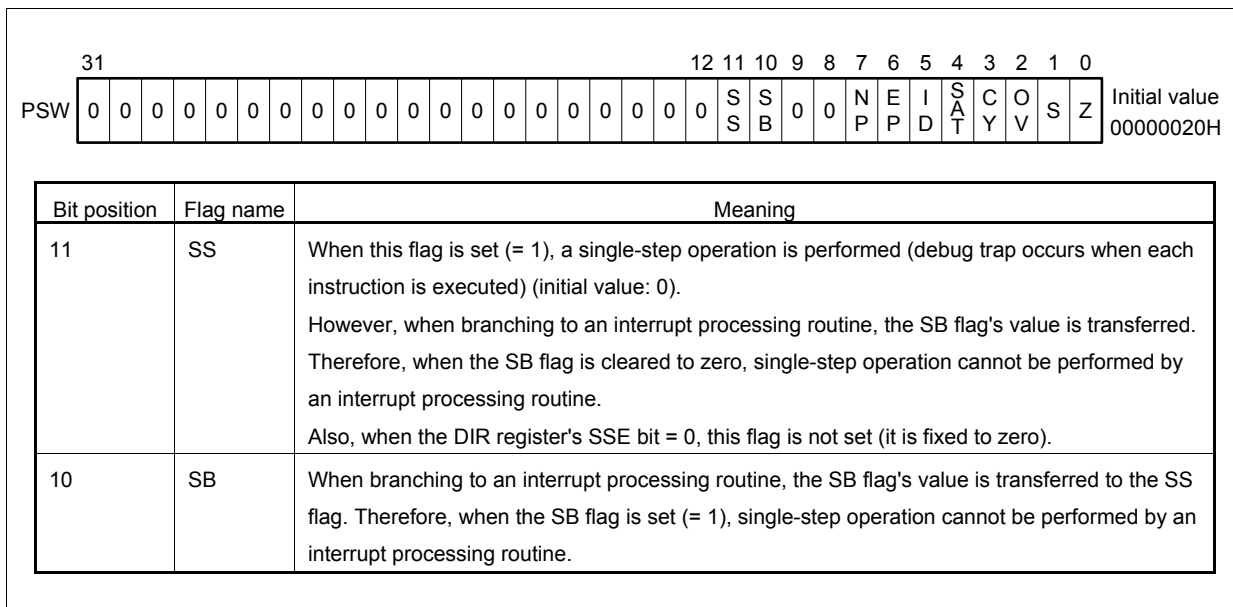


Figure 2-6. Program Status Word (PSW) (2 of 2)

Bit position	Flag name	Meaning
7	NP	This bit indicates when non-maskable interrupt (NMI) processing is in progress. When an NMI request is received, this bit is set (= 1) and multiple interrupts are prohibited. 0: NMI processing not in progress (initial value) 1: NMI processing in progress
6	EP	This bit indicates when exception processing is in progress. When an exception has occurred, this bit is set (= 1). Interrupts can be received when this bit = 1. 0: Exception processing not in progress (initial value) 1: Exception processing in progress
5	ID	This bit indicates whether or not maskable interrupt requests can be received. 0: Enable interrupts (EI) 1: Disable interrupts (DI) (Initial value)
4	SAT <sup>Note</sup>	This indicates when a saturation instruction's result overflows, causing the result of the operation to be saturated. Since this flag is a cumulative flag, it is set (= 1) when saturation instruction's result is saturated and is not cleared to zero even when the subsequent instruction's result is not saturated. The LDSR instruction must be used to clear this bit. 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	Indicates when a carry or borrow occurs in an operation result. 0: Carry or borrow has not occurred (initial value) 1: Carry or borrow has occurred
2	OV <sup>Note</sup>	Indicates when an overflow occurs in an operation result. 0: Overflow has not occurred (initial value) 1: Overflow has occurred
1	S <sup>Note</sup>	Indicates when operation result is a negative value. 0: Operation result is a positive value or 0 (initial value) 1: Operation result is a negative value.
0	Z	Indicates whether or not the operation result is "0". 0: The operation result is not "0". (initial value) 1: The operation result is "0".

**Note** The value in the OV flag and S flag for the saturation operation determines the operation result of saturation processing. If the OV flag is set (= 1) during saturation processing, the SAT flag is set (= 1).

Operation result status	Flag's status			Operation result after saturation processing
	SAT	OV	S	
When beyond maximum positive value	1	1	0	7FFFFFFH
When beyond maximum negative value	1	1	1	80000000H
Positive (Not exceeding maximum value)	Holds the value before operation	0	0	Operation result
Negative (Not exceeding maximum value)			1	

### 2. 2. 5 CALLT Status-saving Registers (CTPC and CTPSW)

The CALLT status-saving registers include the CTPC register and CTPSW register.

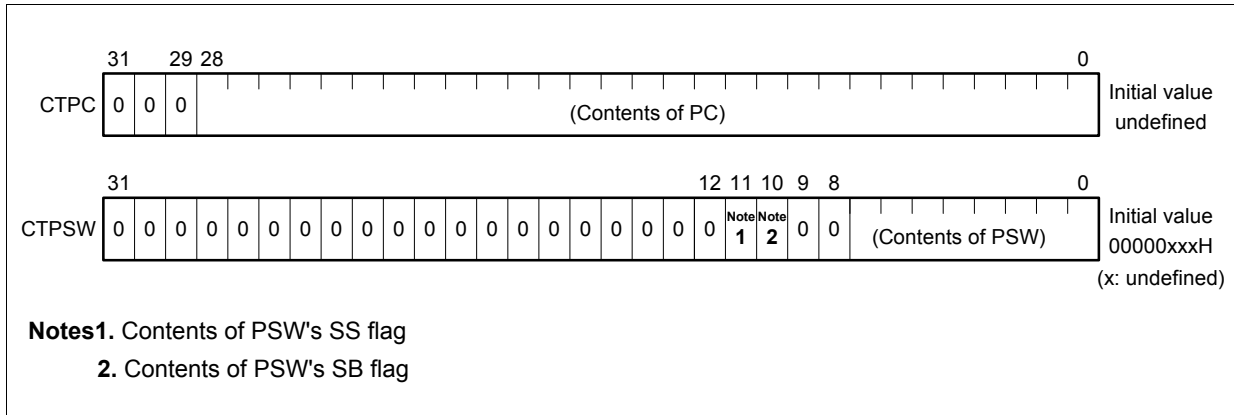
When a CALLT instruction is executed, the value of the program counter (PC) is saved in the CTPC register and the value in the program status word (PSW) is saved in the CTPSW register.

The value saved in the CTPC register is the addresses of the instruction following the CALLT instruction.

The current PSW value is saved in the CTPSW register.

CTPC register bits 31 to 29 and CTPSW register bits 31 to 12, 9 and 8 are reserved (fixed to 0) for future expanded functions.

**Figure 2-7. CALLT Status-saving Registers (CTPC and CTPSW)**



## 2. 2. 6 Exception/Debug Trap Status-saving Registers (DBPC and DBPSW)

Two exception/debug trap status saving registers are provided: DBPC and DBPSW.

When an exception trap, debug trap, or debug break occurs, or when a single-step operation is executed, the value in the program counter (PC) is saved in DBPC and the value in the program status word (PSW) is saved in the DBPSW register.

The value saved in the DBPC register is described below.

**Table 2-4. Value Saved in DBPC Register**

Factor		Value saved in DBPC
Exception trap occurs		Address of instruction following instruction where exception trap factor occurred
Debug trap occurs		Address of instruction following instruction where debug trap factor occurred
Debug break occurs	execution-related trap	Address of instruction where break factor occurred
	Misaligned access exception	
	Alignment error exception	
	access-related trap	Address of instruction following instruction where break factor occurred
Execution of single-step operation		Address of next instruction to be executed (instruction executed when recovering from debug monitor routine)

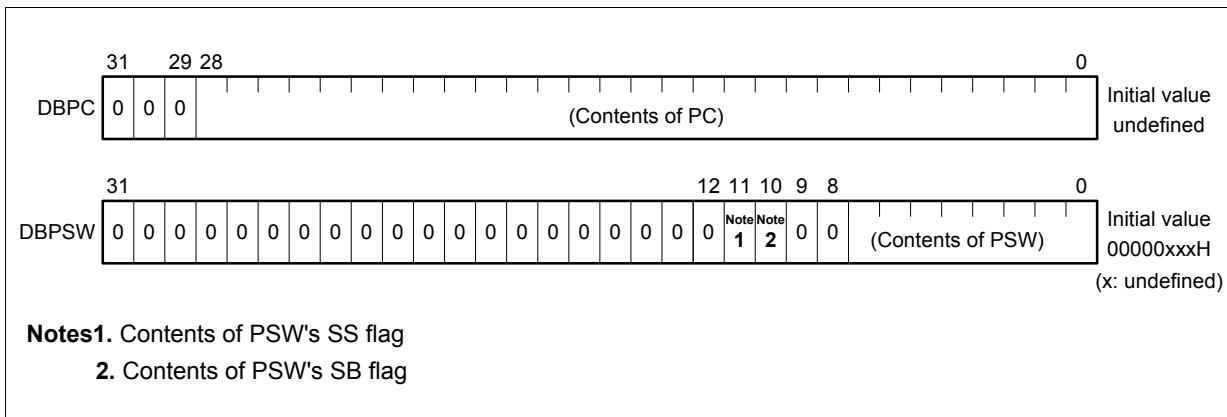
**Remark** For details of causes for saving, refer to **CHAPTER 9 SHIFTING TO DEBUG MODE**.

The current value of the PSW is saved to DBPSW.

Bits 31 to 29 of DBPC register and bits 31 to 12, 9 and 8 of DBPSW register are reserved (fixed to zero) for future expanded functions.

**Remark** The DBPC and DBPSW registers are read/write accessible in user mode.

**Figure 2-8. Exception/Debug Trap Status-saving Registers (DBPC and DBPSW)**

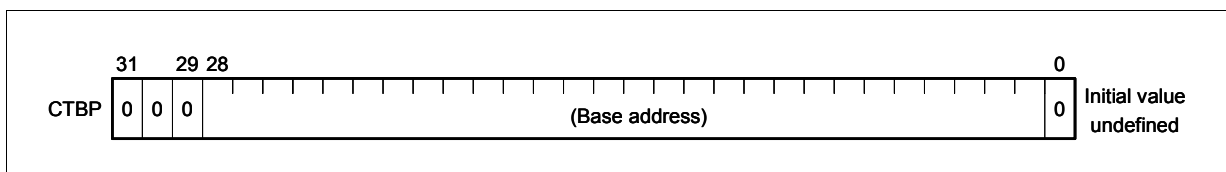


### 2. 2. 7 CALLT base pointer (CTBP)

The CALLT base pointer (CTBP) is used to specify table addresses and create target addresses (bit 0 is fixed to zero).

Bits 31 to 29 are reserved for future expanded functions (fixed to zero).

Figure 2-9. CALLT Base Pointer (CTBP)



### 2. 2. 8 Debug Interface register (DIR)

The Debug Interface register (DIR) is used to control and indicate the status of debug functions.

When the LDSR instruction is used to change the value in the bits in this register, the revised value becomes valid as soon as execution of the LDSR instruction ends.

In debug mode, each bits can always be written (except write-prohibited-bits (bits 31, 27 to 23, 19 to 17, and 15) and read-only-bits (bits 3 and 0)). In user mode, only CSL bit can be written. Moreover, in both debug mode and user mode, DIR register can always be read (the read value of bits 31, 27 to 23, 19 to 17, and 15 are undefined). However, in user mode, the read value of bits other than CSL bit is undefined (except read-only-bits (bits 3 and 0)).

Figure 2-10. DIR Register's Write-accessible Bits during user mode

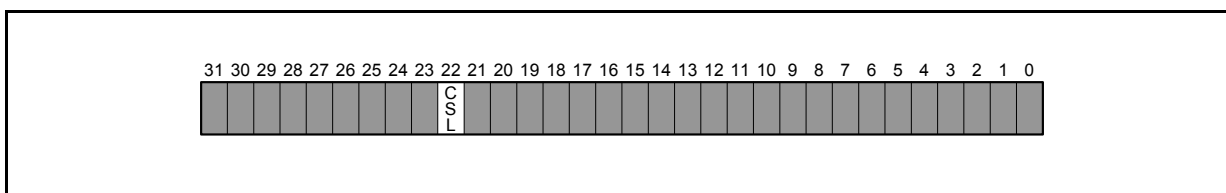


Figure 2-11. Debug Interface Register (DIR) (1 of 3)

- Cautions**
1. Either one of the DIR register's SQ1 and RE1 bits must be set (= 1) or both must be cleared (= 0). Operation is not guaranteed if both are set (= 1).
  2. Set a "1" to either the SQ0 or RE0 bit or zero-clear both of them. A break will not occur if both bits are set (= 1).
  3. Bit31, 27 to 23, 19 to 17, 15 of this register are writing values other than 0 to them is prohibited. Their values when read is undefined.

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
DIR	0	SQ 1	RE 1	CS 1	0	0	0	0	0	0	CS L	BT 3	BT 2	0	0	0	0	ST T	0	SQ 0	RE 0	CS 0	0	MA E	AE E	SE E	EX T	IN I	BT 1	BT 0	0	MT T	AT T	DM	Initial value 00000040H

Bit position	Bit name	Meaning						
30	SQ1 <sup>Note1</sup>	Sets sequential break mode for Channels 2 and 3 (when break occurs, it occurs in the sequence of Channel 2 → Channel 3). 0: Normal break mode (initial value) 1: Sequential break mode						
29	RE1 <sup>Note1</sup>	Sets range break mode for Channels 2 and 3 (when break occurs, it occurs at the same time for Channels 2 and 3). 0: Normal break mode (initial value) 1: Range break mode						
28	CS1 <sup>Note1</sup>	Enables settings in control registers (BPCn, BPAVn, BPAMn, BPDVn, and BPDMn) (n = 2 or 3) for Channels 2 and 3 (not an enable/disable setting for break conditions). 0: Enables settings in Channel 2 control register (BPC2, BPxx2) (initial value) 1: Enables settings in Channel 3 control register (BPC3, BPxx3)						
22	CSL	Enables settings in each channel's control register (see <b>Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid and Channels and Registers</b> ) <table><tr><th>CSL</th><th>Access target</th></tr><tr><td>0</td><td>Channels 0 and 1 (initial value)</td></tr><tr><td>1</td><td>Channels 2 and 3</td></tr></table>	CSL	Access target	0	Channels 0 and 1 (initial value)	1	Channels 2 and 3
CSL	Access target							
0	Channels 0 and 1 (initial value)							
1	Channels 2 and 3							
21	BT3 <sup>Note2</sup>	Set (= 1) when Channel 3 break occurs (cannot be freely set (= 1) by user program) (initial value = 0)						
20	BT2 <sup>Note2</sup>	Set (= 1) when Channel 2 break occurs (cannot be freely set (= 1) by user program) (initial value = 0)						

- Notes**
1. When the INI bit is set (= 1), write is disabled for the SQ1, RE1, and CS1 bits. Also, setting the INI bit automatically zero-clears each other bit.
  2. Bits BT2 and BT3 do not operate when the INI bit is set (= 1) (i.e., they are not set (= 1) when a break occurs). Once these bits are set (= 1), they are not cleared to zero until the LDSR instruction sets a "0" to them or until the INI bit is set (= 1).

Figure 2-11. Debug Interface Register (DIR) (2 of 3)

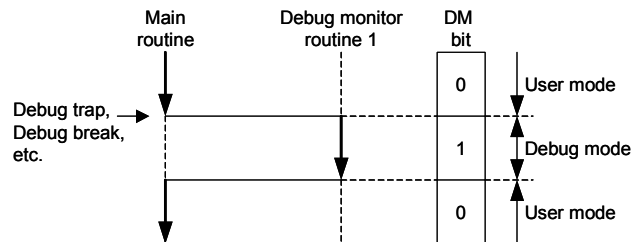
Bit position	Bit name	Meaning
16	STT	This bit is set (= 1) when a debug trap is executed (it cannot be freely set (= 1) by a user program) (initial value: 0). Once this bit is set (= 1) it cannot be automatically cleared to zero (it can only be cleared by the LDSR instruction).
14	SQ0 <sup>Note</sup>	Sets sequential break mode for Channels 0 and 1 (when break occurs, it occurs in the sequence of Channel 0 → Channel 1). 0: Normal break mode (initial value) 1: Sequential break mode
13	RE0 <sup>Note</sup>	Sets range break mode for Channels 0 and 1 (when break occurs, it occurs at the same time for Channels 0 and 1). 0: Normal break mode (initial value) 1: Range break mode
12	CS0 <sup>Note</sup>	Enables settings in control registers (BPCn, BPAVn, BPAMn, BPDVn, and BPDm) (n = 0 or 1) for Channels 0 and 1 (not an enable/disable setting for break conditions). 0: Enables settings in Channel 0 control register (BPC0, BPxx0) (initial value) 1: Enables settings in Channel 1 control register (BPC1, BPxx1)
10	MAE	Sets enable/disable status of misaligned access exception detection. 0: Disables misaligned access exception detection (initial value) 1: Enables misaligned access exception detection
9	AEE	Sets enable/disable status of alignment error exception detection. 0: Disables alignment error exception detection (initial value) 1: Enables alignment error exception detection
8	SSE	Sets enable/disable status of PSW's SS flag write operation. 0: Disables write to SS flag (SS flag is fixed to zero) (initial value) 1: Enables write to SS flag
7	EXT	Sets valid/invalid status of extended debug function (function assigned to bits 31 to 15 of this register). 0: Invalid (V850E1 CPU compatible) (initial value) 1: Valid

**Note** When the INI bit is set (= 1), write is disabled to the SQ0, RE0, and CS0 bits. Each other bit is automatically cleared to zero when the INI bit is set (= 1).

Figure 2-11. Debug Interface Register (DIR) (3 of 3)

Bit position	Bit name	Meaning
6	INI <sup>Note1</sup>	This bit is set (= 1) when the debug function is reset (initial value: 1). After a reset, this bit is set (= 1), so be sure to clear it to zero (when this bit is set (= 1), write is disabled to the SQn, REn, and CSn bits (n = 0 or 1). Also, bits BT3 to BT0 cannot be operated.)
5	BT1 <sup>Note2</sup>	This bit is set (= 1) when a break occurs in Channel 1 (cannot be freely set (= 1) by user program) (initial value = 0).
4	BT0 <sup>Note2</sup>	This bit is set (= 1) when a break occurs in Channel 0 (cannot be freely set (= 1) by user program) (initial value = 0).
2	MT <sup>Note1</sup>	This bit is set (= 1) when a misaligned access exception is detected (cannot be freely set (= 1) by user program) (initial value = 0).
1	AT <sup>Note1</sup>	This bit is set (= 1) when an alignment error exception is detected (cannot be freely set (= 1) by user program) (initial value = 0).
0	DM <sup>Note3</sup>	This bit is set (= 1) when processing goes to debug mode (initial value = 0). This bit is not write-accessible.

- Notes1.** When the INI, MT, or AT bit is set (= 1), this bit is not automatically cleared to zero (it can only be cleared to zero by an LDSR instruction).
- 2.** When the INI bit is set (= 1), the BT0 and BT1 bits do not operate (they are not set (= 1) when a break occurs). Also, once these bits are set (= 1), they are not cleared to zero unless cleared by the LDSR instruction or when the INI bit is set (= 1).
- 3.** The DM bit change as shown below.





**Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**

CSL	CS1	CS0	Channels and registers for which break conditions can be set	
			Channel	Registers
0	Don't care	0	Channel 0	BPC0, BPAV0, BPAM0, BPDV0, BPDM0
		1	Channel 1	BPC1, BPAV1, BPAM1, BPDV1, BPDM1
1	0	Don't care	Channel 2	BPC2, BPAV2, BPAM2, BPDV2, BPDM2
	1	Don't care	Channel 3	BPC3, BPAV3, BPAM3, BPDV3, BPDM3

## 2. 2. 9 Breakpoint Control registers 0 to 3 (BPC0 to BPC3)

Breakpoint Control registers 0 to 3 (BPC0 to BPC3) are used to control and indicate the debug functions for Channels 0 to 3.

The valid registers are selected corresponding to the channels that are selected by the settings of the DIR register's CSL, CS1, and CS0 bits (See **Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**).

When the LDSR instruction is used to change bit settings in this register, the modified bit settings become valid as soon as execution of the LDSR instruction ends (if the FE bit is set (= 1), the timing for becoming valid is a little slower, but the settings are reflected after the DBRET instruction has been executed).

**Cautions**1. Be sure to zero-clear bits 31 to 27, 14 to 12, 6, and 5 in the BPCn register (n = 0 to 3).

Operation is not guaranteed if any of these bits are set (= 1).

2. Only "0" can be written to bits FB2 to FB0 in the BPCn register (n = 0 to 3).

To update the values of these bits, clear the bits to zero. Operation is not guaranteed if any of these bits are set (= 1).

BPC0	<div> <div> <div>31</div> <div>27 26 25 24 23</div> <div>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> </div> <div> <div>0 0 0 0 0</div> <div>FB2 FB1 FB0</div> <div>BP ASID</div> <div>IE 0 0 0 TY</div> <div>V D V A M D</div> <div>0 0 T E B E F W R E</div> </div> </div> <div>Initial value 00xxxxx0H (x: undefined)</div>
BPC1	<div> <div> <div>31</div> <div>27 26 25 24 23</div> <div>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> </div> <div> <div>0 0 0 0 0</div> <div>FB2 FB1 FB0</div> <div>BP ASID</div> <div>IE 0 0 0 TY</div> <div>V D V A M D</div> <div>0 0 T E B E F W R E</div> </div> </div> <div>Initial value 00xxxxx0H (x: undefined)</div>
BPC2	<div> <div> <div>31</div> <div>27 26 25 24 23</div> <div>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> </div> <div> <div>0 0 0 0 0</div> <div>FB2 FB1 FB0</div> <div>BP ASID</div> <div>IE 0 0 0 TY</div> <div>V D V A M D</div> <div>0 0 T E B E F W R E</div> </div> </div> <div>Initial value 00xxxxx0H (x: undefined)</div>
BPC3	<div> <div> <div>31</div> <div>27 26 25 24 23</div> <div>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> </div> <div> <div>0 0 0 0 0</div> <div>FB2 FB1 FB0</div> <div>BP ASID</div> <div>IE 0 0 0 TY</div> <div>V D V A M D</div> <div>0 0 T E B E F W R E</div> </div> </div> <div>Initial value 00xxxxx0H (x: undefined)</div>

Bit position	Bit name	Meaning																								
26-24	FB2-FB0	<p>Indicates type of break that occurred when instruction fetch event occurred.</p> <table border="1"> <tr> <th>FB2</th><th>FB1</th><th>FB0</th><th>Break type</th></tr> <tr> <td>0</td><td>0</td><td>0</td><td>Break when execution of target instruction is interrupted (initial value)</td></tr> <tr> <td>0</td><td>1</td><td>0</td><td>Break when execution of target instruction and execution of previous instruction are interrupted</td></tr> <tr> <td>1</td><td>0</td><td>0</td><td>Break when execution of target instruction, execution of previous instruction are interrupted, and execution of instruction before last are interrupted</td></tr> <tr> <td>0</td><td>0</td><td>1</td><td>Break when execution of target instruction ends</td></tr> <tr> <td colspan="3">(Other than above)</td><td>(Reserved for future expansion of function)</td></tr> </table>	FB2	FB1	FB0	Break type	0	0	0	Break when execution of target instruction is interrupted (initial value)	0	1	0	Break when execution of target instruction and execution of previous instruction are interrupted	1	0	0	Break when execution of target instruction, execution of previous instruction are interrupted, and execution of instruction before last are interrupted	0	0	1	Break when execution of target instruction ends	(Other than above)			(Reserved for future expansion of function)
FB2	FB1	FB0	Break type																							
0	0	0	Break when execution of target instruction is interrupted (initial value)																							
0	1	0	Break when execution of target instruction and execution of previous instruction are interrupted																							
1	0	0	Break when execution of target instruction, execution of previous instruction are interrupted, and execution of instruction before last are interrupted																							
0	0	1	Break when execution of target instruction ends																							
(Other than above)			(Reserved for future expansion of function)																							
23-16	BP ASID	<p>Sets program ID where break occurred (initial value: undefined). This setting is valid only when the IE bit has been set (= 1).</p>																								
15	IE	<p>Sets comparison between BP ASID bit and program ID set to ASID register (initial value: undefined). 0: Do not compare 1: Compare</p>																								
11, 10	TY	<p>Sets access type for detecting break (initial value: undefined).</p> <table border="1"> <tr> <th>Bit 11</th><th>Bit 10</th><th>Access type for detecting break</th></tr> <tr> <td>0</td><td>0</td><td>Access of all data types</td></tr> <tr> <td>0</td><td>1</td><td>Byte access (including bit manipulation)</td></tr> <tr> <td>1</td><td>0</td><td>Half word access</td></tr> <tr> <td>1</td><td>1</td><td>Word access</td></tr> </table> <p>These registers' settings are invalid for execution-related traps.</p>	Bit 11	Bit 10	Access type for detecting break	0	0	Access of all data types	0	1	Byte access (including bit manipulation)	1	0	Half word access	1	1	Word access									
Bit 11	Bit 10	Access type for detecting break																								
0	0	Access of all data types																								
0	1	Byte access (including bit manipulation)																								
1	0	Half word access																								
1	1	Word access																								

**Figure 2-12. Breakpoint Control Registers 0 to 3 (BPC0 to BPC3) (2 of 2)**

Bit position	Bit name	Meaning
9	VD	Sets match condition for data comparator (initial value: undefined) 0: Break when match occurs 1: Break when mismatch occurs
8	VA	Sets match condition for address comparator (initial value: undefined) 0: Break when match occurs 1: Break when mismatch occurs
7	MD	Sets operation of data comparator 0: Break when data matches condition 1: Ignore when data matches condition (ignore data comparator), regardless of VD bit value or settings in BPDVx and BPDMx registers.
4	TE	Sets enable/disable status of trigger output when event occurs for Channels 0 to 3 0: Disable trigger output (initial value) 1: Enable trigger output (corresponding trigger is output)
3	BE	Sets whether or not to notify CPU when break is triggered by event occurring for Channels 0 to 3 0: Do not notify (initial value) 1: Notify (break occurs)
2	FE	Sets whether or not to mask event during an instruction fetch operation 0: Mask event (initial value) 1: Event occurs <sup>Note1</sup>
1	WE	Sets whether or not to mask event during a data write operation 0: Mask event (initial value) 1: Event occurs <sup>Note2</sup>
0	RE	Sets whether or not to mask event during a data read operation 0: Mask event (initial value) 1: Event occurs <sup>Note2</sup>

**Notes 1.** When the FE bit set (= 1), be sure to zero-clear the WE and RE bits.

**2.** When the WE bit or RE bit is set (= 1), be sure to zero-clear the FE bit.

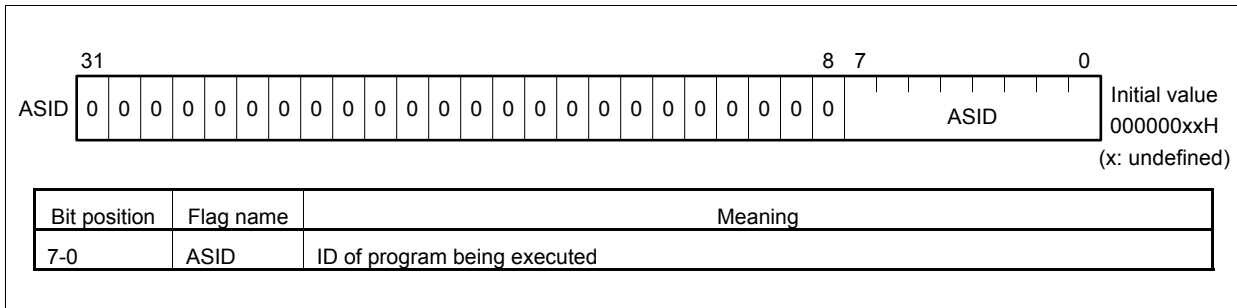
## 2. 2. 10 Program ID register (ASID)

This register is used to set the ID of the program (application software) currently being executed.

The Program ID is used when changing to debug mode is required for executing certain programs, such as when downloading different programs to the same address area in RAM. When the BPCn register's IE bit is set, if the Program IDs set to the BP ASID bit and the ASID register do not match, the mode is not switched to debug mode even if a break condition is met (n = 0 to 3).

In this register, bits 31 to 8 are reserved for future expanded functions (fixed to 0).

**Figure 2-13. Program ID Register (ASID)**



## 2. 2. 11 Breakpoint Address Setup registers 0 to 3 (BPAV0 to BPAV3)

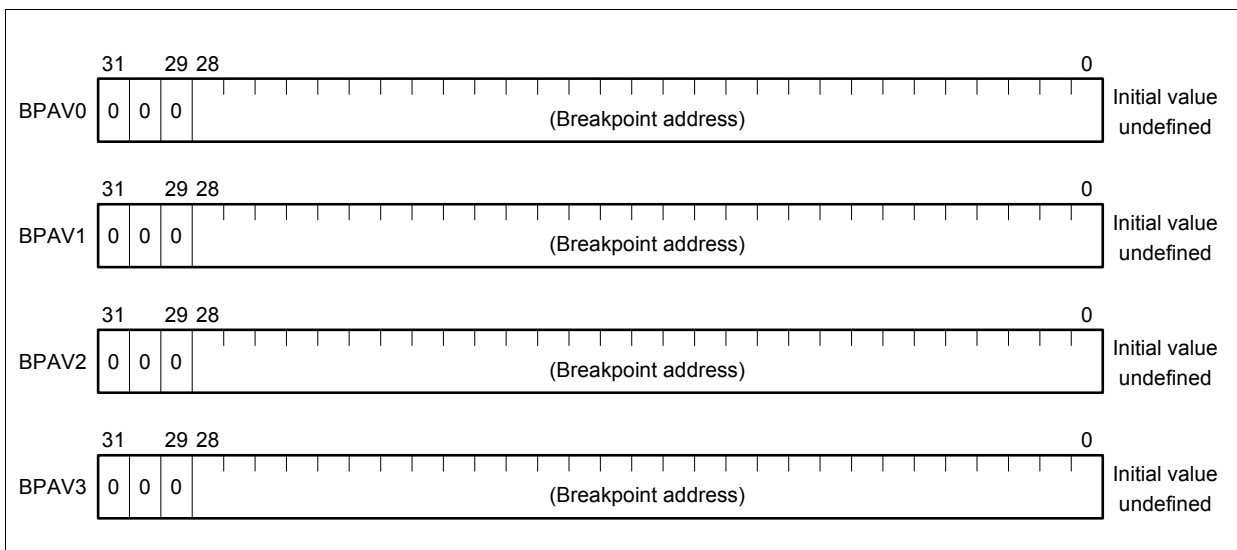
This register sets breakpoint addresses used by the address comparator.

Valid registers are selected via settings of the DIR register's CSL, CS1, and CS0 bits (See **Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**).

When not using these bits, make sure each bit is set (= 1).

In this register, bits 31 to 29 are reserved for future expanded functions (fixed to 0).

**Figure 2-14. Breakpoint Address Setup Registers 0 to 3 (BPAV0 to BPAV3)**



## 2. 2. 12 Breakpoint Address Mask registers 0 to 3 (BPAM0 to BPAM3)

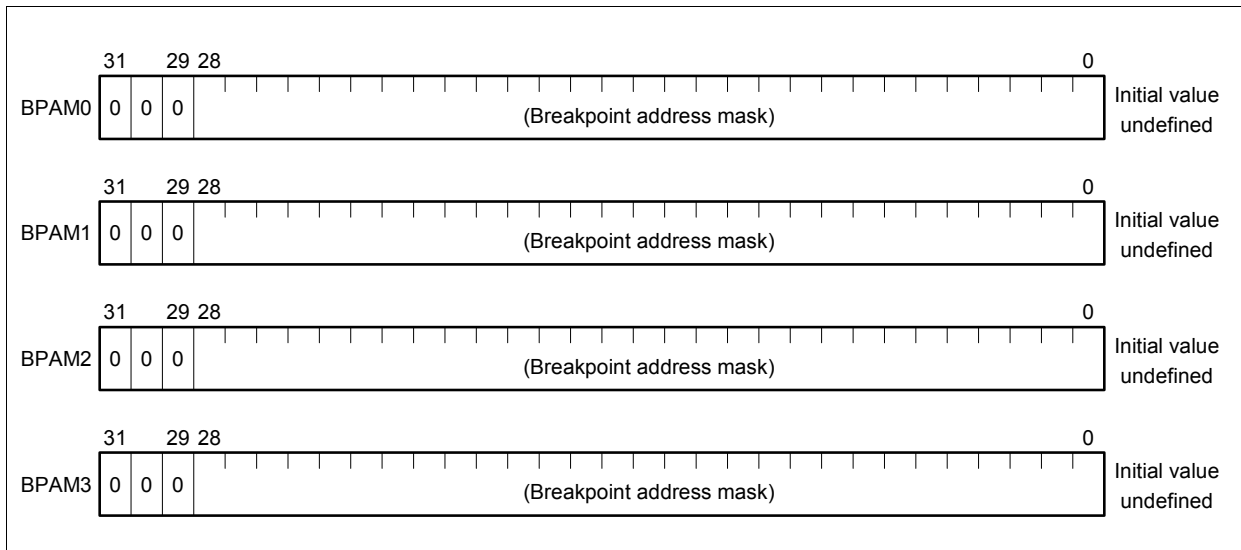
These registers are used to set masking of bits for address comparison (1 = masked).

Valid registers are selected via settings of the DIR register's CSL, CS1, and CS0 bits (See **Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**).

When not using these bits, make sure each bit is set (= 1).

In this register, bits 31 to 29 are reserved for future expanded functions (fixed to 0).

**Figure 2-15. Breakpoint Address Mask Registers 0 to 3 (BPAM0 to BPAM3)**



## 2. 2. 13 Breakpoint Data Setup registers 0 to 3 (BPDV0 to BPDV3)

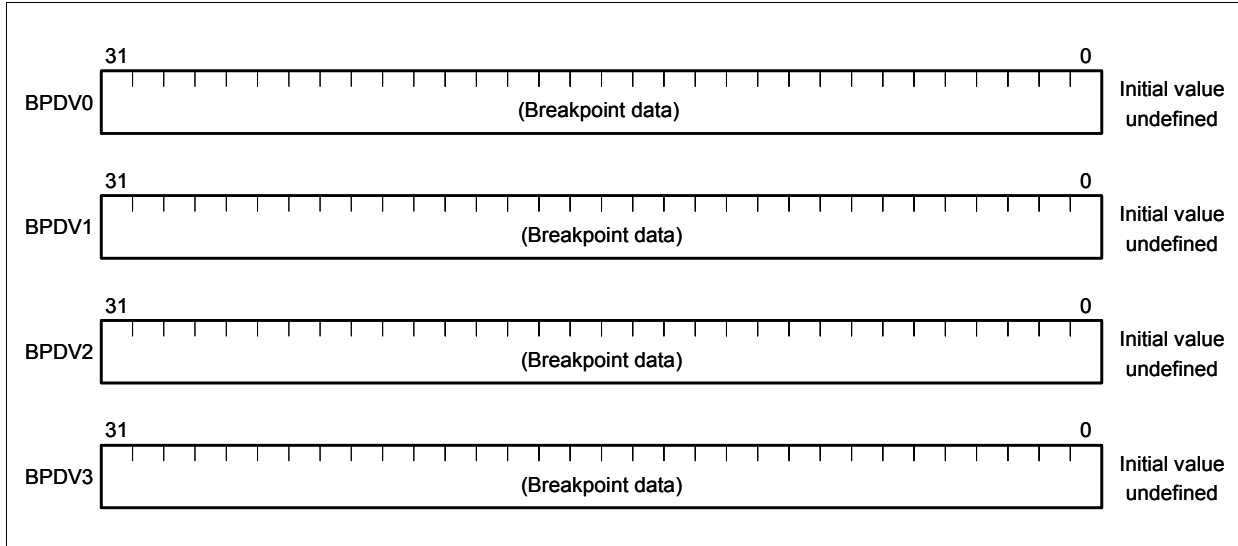
These registers are used to set breakpoint data used by the data comparator.

Valid registers are selected via settings of the DIR register's CSL, CS1, and CS0 bits (See **Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**).

When not using these bits, make sure each bit is set (= 1).

**Remark** When setting instruction codes for 16-bit instructions, align the setting with the LSB.  
When setting instruction codes for 32-bit instructions, enter the setting in little endian format.

**Figure 2-16. Breakpoint Data Setup Registers 0 to 3 (BPDV0 to BPDV3)**



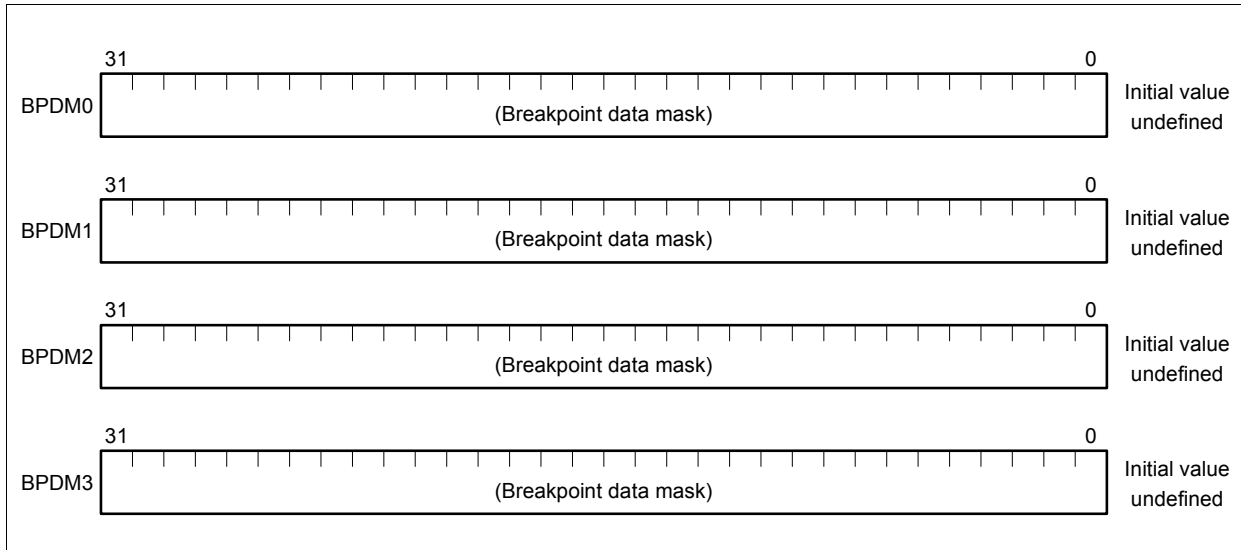
## 2. 2. 14 Breakpoint Data Mask Registers 0 to 3 (BPDM0 to BPDM3)

These registers are used to set masking of bits for data comparison (1 = masked).

Valid registers are selected via settings of the DIR register's CSL, CS1, and CS0 bits (See **Table 2-5. Relation between CSL, CS1, CS0 Bits and Valid Channels and Registers**).

When not using these bits, make sure each bit is set (= 1).

**Figure 2-17. Breakpoint Data Mask Registers 0 to 3 (BPDM0 to BPDM3)**



# CHAPTER 3 DATA TYPE

## 3. 1 Data Format

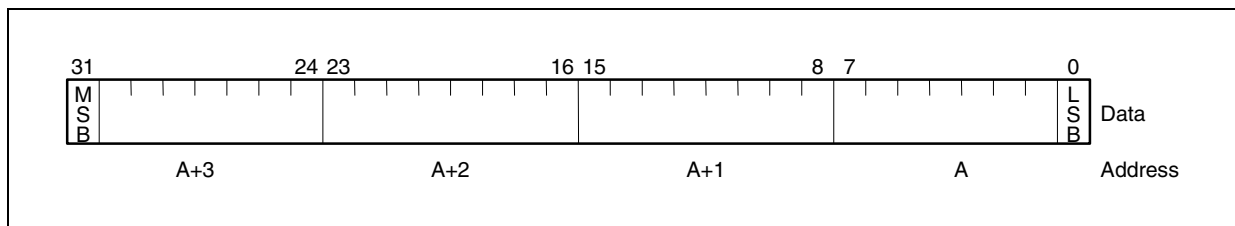
The following data types are supported. Refer to **3.2 Data Representation**.

- Integer (32, 16, 8 bits)
- Unsigned integer (32, 16, 8 bits)
- Bit

In the case of word (32 bits), half-word (16 bits), and byte (8 bits), byte 0 of any data is always the least significant byte, known as "little endian." It occupies at the right-most position in figures throughout this manual. The following describes the format of the fixed length data.

### (1) Word

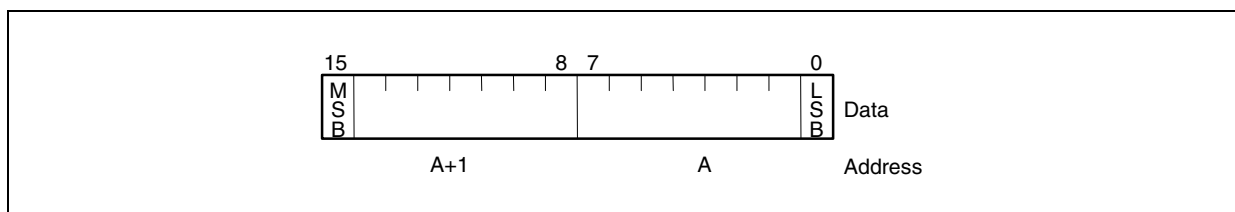
A word is 4-byte (32-bit) contiguous data that starts at any word boundary. Note It is expressed by "A," "A+1," "A+2," and "A+3." Each bit is assigned a number from 0 to 31; the LSB (Least Significant Bit) is bit 0 and the MSB (Most Significant Bit) is bit 31. A word is specified by its address "A." The lowest 2 bits are fixed to "0" with misalign access being disabled. Note



**Note** When misalign access is enabled, any byte boundary can be accessed whether access is gained in half-word or word. Refer to **3. 3 Data Alignment**.

### (2) Half-word

A halfword is 2-byte (16-bit) contiguous data that starts from any halfword boundary<sup>Note</sup>. Each bit is assigned a number from 0 to 15. The LSB is bit 0 and the MSB is bit 15. A halfword is specified by its address "A" (with the lowest bit fixed to 0<sup>Note</sup>), and occupies 2 bytes, A and A+1.

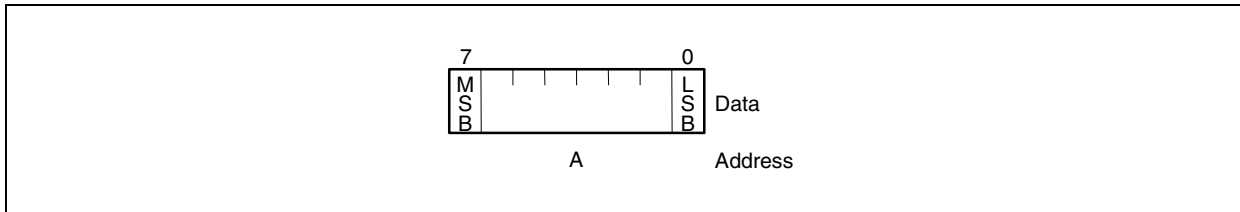


**Note** When misalign access is enabled, any byte boundary can be accessed whether access is gained in half-word or word. Refer to **3. 3 Data Alignment**.



(3) Byte

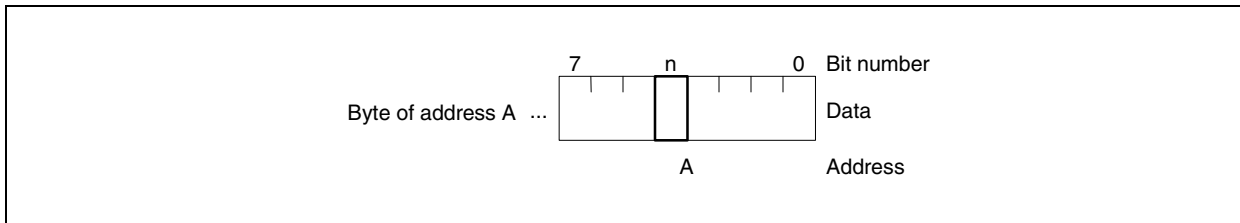
A byte is 8-bit contiguous data that starts from any byte boundary<sup>Note</sup>. Each bit is assigned a number from 0 to 7. The LSB is bit 0 and the MSB is bit 7. A byte is specified by its address "A".



**Note** When misalign access is enabled, any byte boundary can be accessed whether access is gained in half-word or word. Refer to **3. 3 Data Alignment**.

(4) Bit

A bit is 1-bit data at the "n" th bit position in 8-bit data that starts at any byte boundary<sup>Note</sup>. A bit is specified by its address "A" and bit number "n."



**Note** When misalign access is enabled, any byte boundary can be accessed whether access is gained in half-word or word. Refer to **3. 3 Data Alignment**.

## 3. 2 Data Representation

### 3. 2. 1 Integer

An integer is expressed as a binary number of 2's complement and is 32-, 16-, or 8-bit long. Regardless of its length, the bit 0 of an integer is the least significant bit. The higher the bit number, the more significant the bit. The most significant bit is designated as signed bit. The integer range of each data length is as follows:

- Word (32 bits):                      –2147483648 to +2147483647
- Half-word (16 bits):                –32768 to +32767
- Byte (8 bits):                        –128 to +127

### 3. 2. 2 Unsigned integer

An integer partakes of a value of either positive or negative, whereas an unsigned integer is either positive or "0." Signed or unsigned, an integer is expressed as 2's complement and is 32-, 16-, or 8- bit long. Regardless of its length, bit 0 of an unsigned integer is the least significant bit, and the higher the bit number, the more significant the bit. No signed bit is used. The unsigned integer range of each data length is as follows.

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

### 3. 2. 3 Bit

The 1-bit data that can partakes of a value of "0" (cleared) or "1" (set) is available. Bit manipulation can be executed only to the 1-byte data in the memory space. There are four types of bit manipulation:

- SET1
- CLR1
- NOT1
- TST1

### 3.3 Data Alignment

Data alignment (boundary alignment) may be required, depending on the setting (enabled/prohibited) for misaligned access.

When the data to be processed is in half word format, misaligned access occurs when an address beyond the half word boundary (when the address's MSB = 0) is accessed. When the data to be processed is in word format, misaligned access occurs when an address beyond the word boundary (when the address's lowest two bits = 0) is accessed.

**Remark** In the V850E2 core, the enabled/prohibited setting for misaligned access is set by the level of input to the IFIMAEN pin.

#### (1) When misaligned access is set as enabled

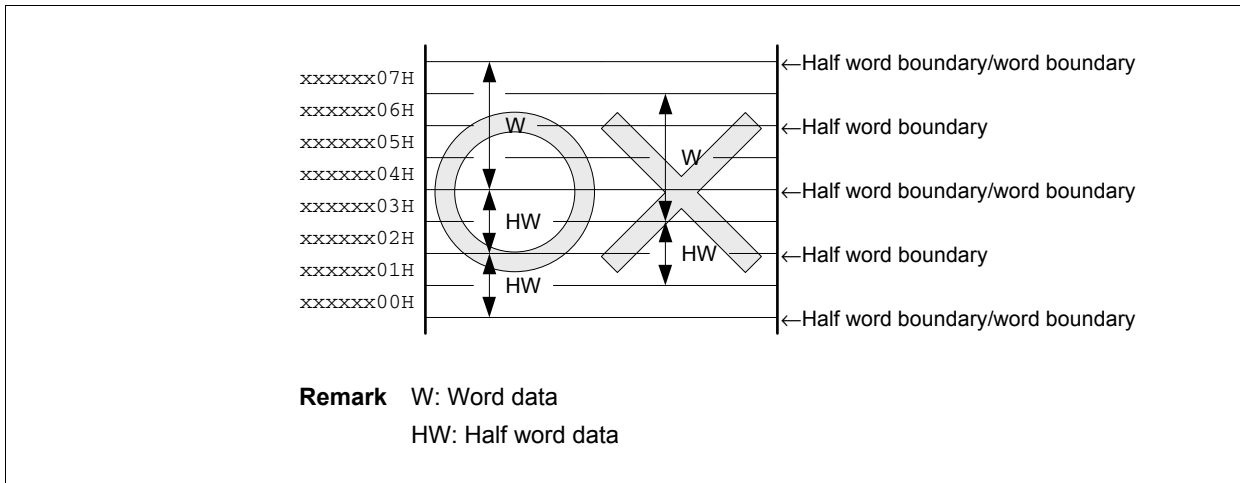
When misaligned access is set as enabled, data can be allocated to all addresses, regardless of the data format (byte, half word, or word).

However, when the data format is half word or word, if the data is not aligned, at least one bus cycle will occur, thereby lowering bus efficiency.

#### (2) When misaligned access is set as prohibited

The lower bits in the address (the MSB for half word data or the lower two bits for word data) are masked (to zero) when accessed, and data can be lost or discarded if not correctly aligned. Therefore, when setting data to be processed, be sure to set the data from correct boundary (the half word boundary for data in half word format and the word boundary for data in word format).

**Figure 3-1. Example of Data Allocation when Misaligned Access Is Prohibited**

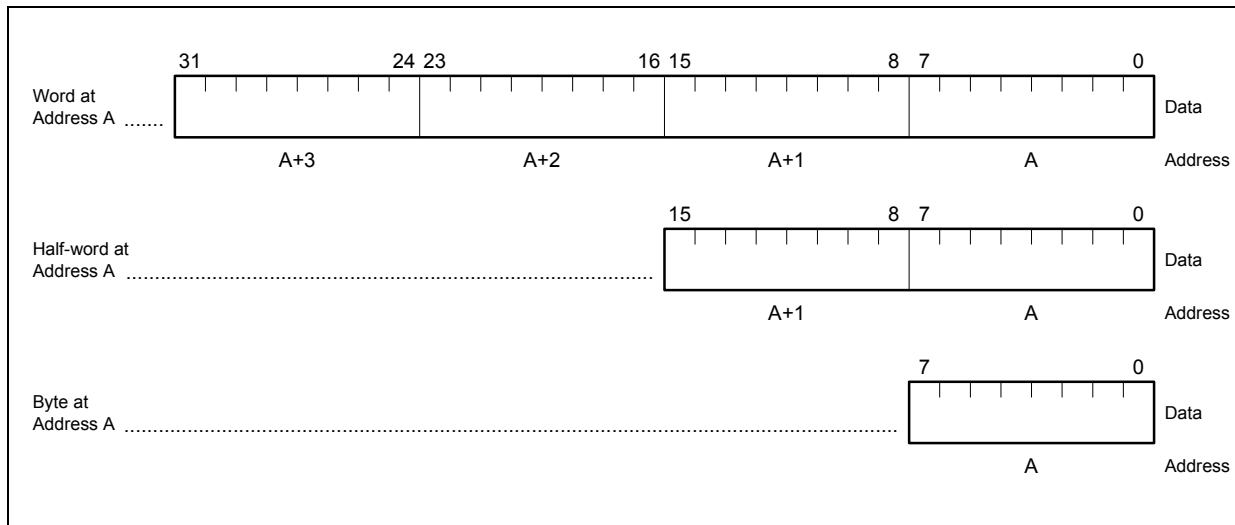


## CHAPTER 4 ADDRESS SPACE

The V850E2 CPU supports a 4-GB linear address space. Both memory and I/O are mapped to this address space, known as memory-mapped I/O. The V850E2 CPU outputs 32-bit addresses to the memory and I/O with the maximum address being  $2^{32}-1$ .

Byte data allocated at each address are defined with bit 0 as LSB and bit 7 as MSB. Multiple-byte data is formatted in little endian, i.e., the byte with the lowest address value has the LSB (Least Significant Bit) and the byte with the highest address value has the MSB (Most Significant Bit).

Throughout this specification the data comprising 2 or more bytes is illustrated as shown below, with the lower address shown on the right and the higher address on the left.

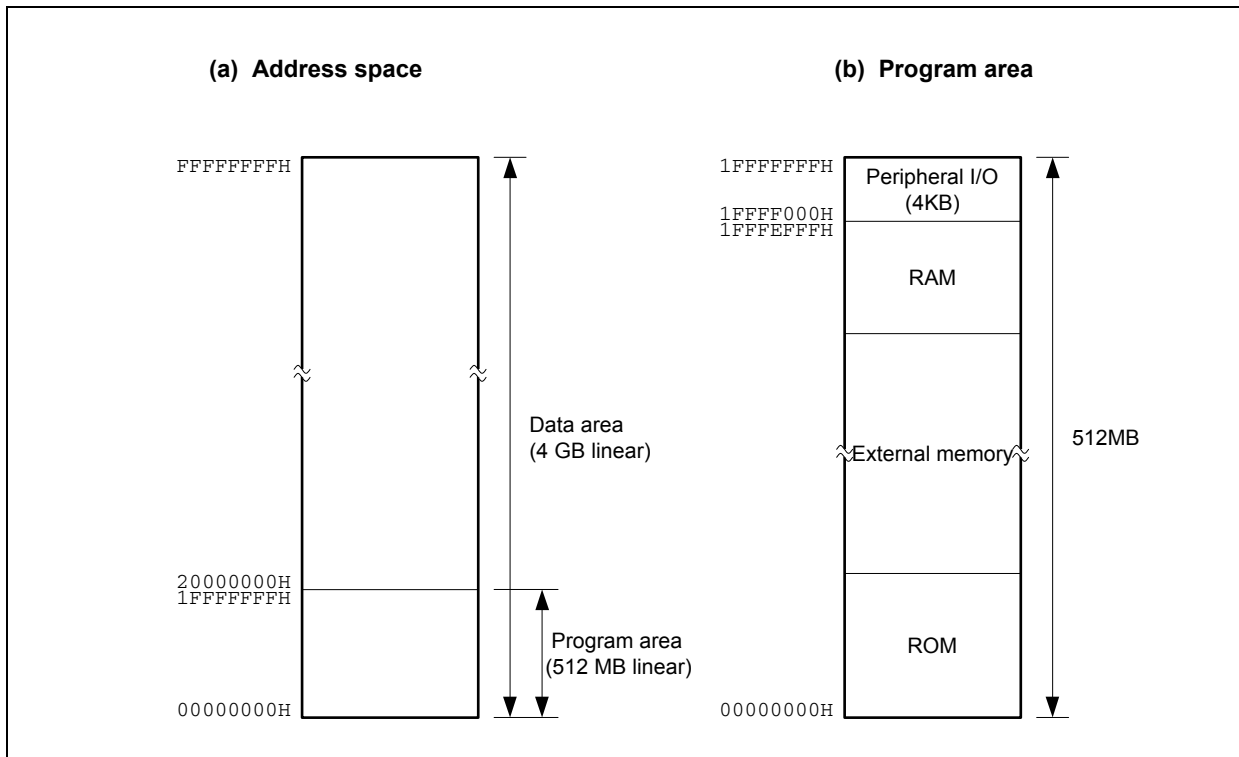


## 4. 1 Memory Map

The V850E2 CPU is a 32-bit architecture. It supports a linear address space (data area) of up to 4 GB for operand addressing (data access) and a linear address space (program area) of up to 512 MB for instruction addressing.

Figure 4-1 shows the memory map.

Figure 4-1. Memory Map



## 4.2 Addressing Modes

CPU generates 2 types of addresses: instruction addresses used for instruction fetch and branch operations; and operand addresses used for data access.

### 4.2.1 Instruction address

An instruction address is determined by the contents of the program counter (PC), and is automatically incremented according to the number of instruction bytes to be fetched with each execution of instructions. When a branch instruction is executed, the branch destination address is loaded into PC using one of the following three addressing modes:

- relative addressing
- register addressing
- Based addressing

#### (1) Relative addressing (PC relative)

The signed 9-, 22-, or 32-bit data of an instruction code (displacement: disp $\times$ ) is added to the value of the program counter (PC), where the displacement data is treated as 2's complement data with bits 8, 21, or 31 serving as sign bits (S). This addressing mode is used for JARL disp22, reg2 instruction, JR disp22 instruction, JARL disp32, reg1 instruction, JR disp32 instruction, and Bcond disp9 instruction only.

Figure 4-2. Relative Addressing (1 of 2)

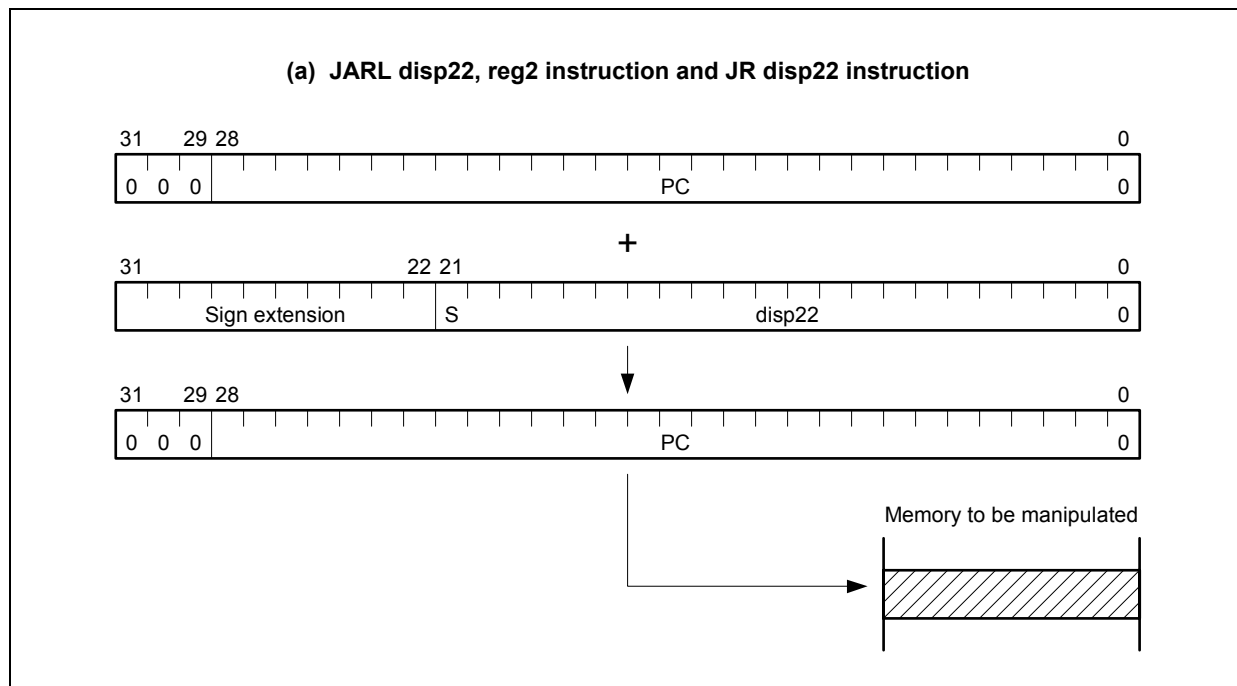
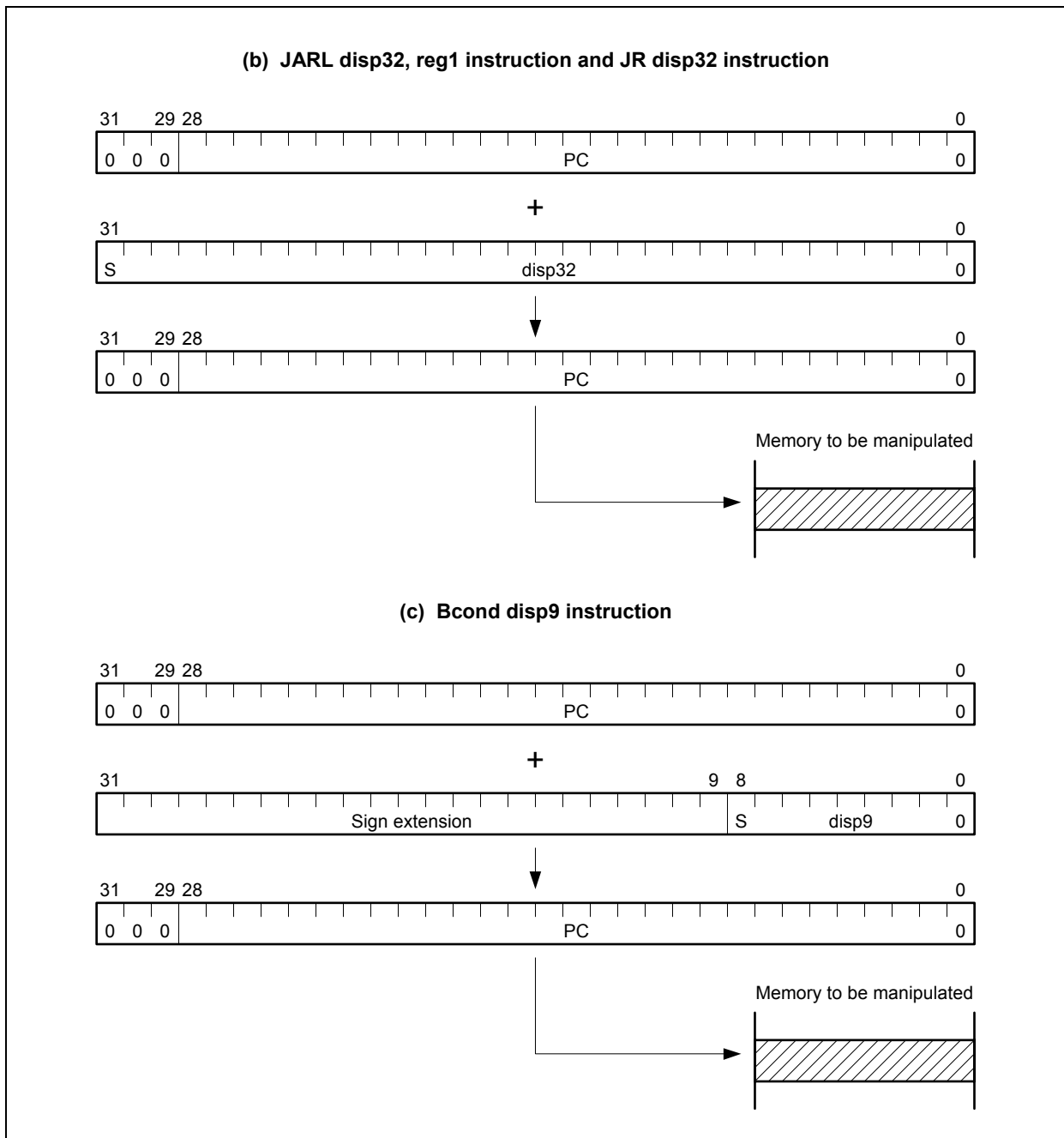


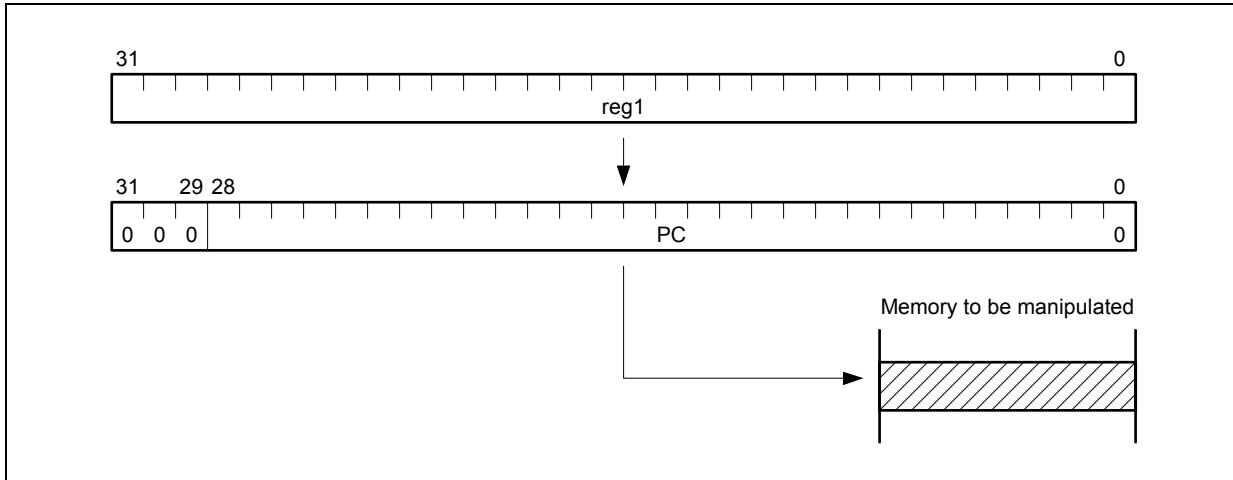
Figure 4-2. Relative Addressing (2 of 2)



**(2) Register addressing (register indirect)**

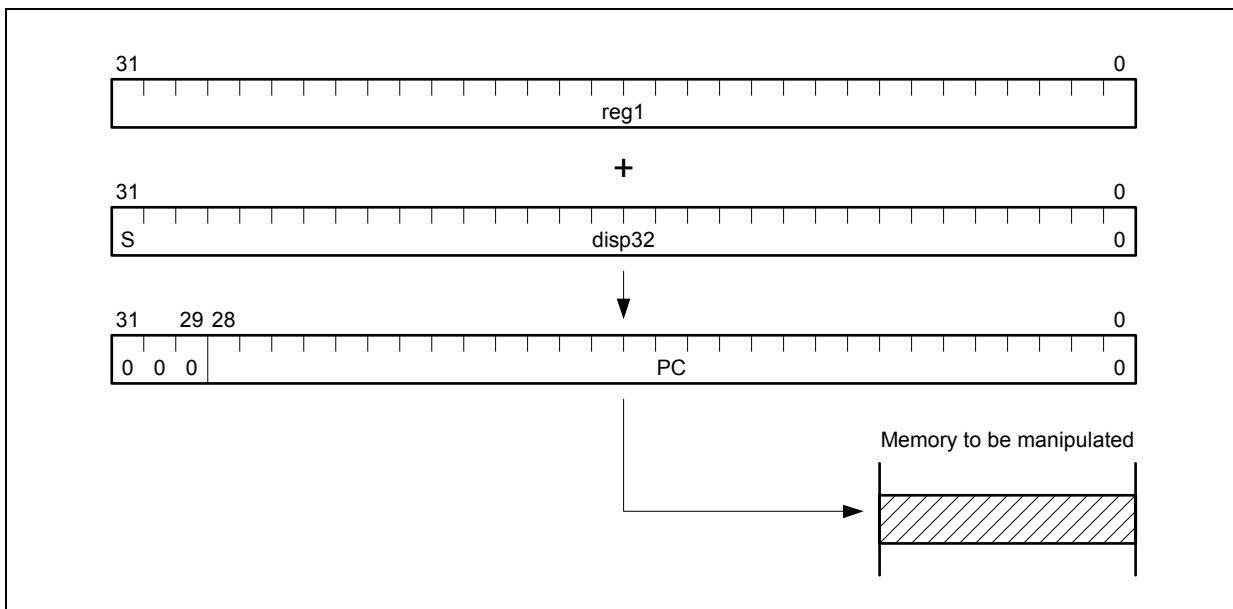
The contents of the general register (reg1) specified by an instruction are transferred to the program counter (PC). This addressing mode is applied to the JMP [reg1] instruction.

**Figure 4-3. Register Addressing (JMP [reg1] instruction)**

**(3) Based addressing**

The 32-bit displacement (disp32) data are added to the general register (reg1) contents and transferred to the program counter (PC). This addressing mode is applied to the JMP disp32 [reg1] instruction.

**Figure 4-4. Based Addressing (JMP disp32 [reg1] instruction)**





## 4. 2. 2 Operand address

The instruction execution requires one of the following four addressing modes to specify the register or memory area:

- Register addressing
- Immediate addressing
- Based addressing
- Bit addressing

### (1) Register addressing

The general register or system register specified in the general register specification field is accessed as operand. This addressing mode applies to instructions using the operand format reg1, reg2, reg3, or regID.

### (2) Immediate addressing

The 5-bit or 16-bit data for manipulation is contained in the instruction code. This addressing mode applies to instructions using the operand format imm5, imm16, vector, or cccc.

**Remark** vector: 5-bit immediate data operand specifies a trap vector (00H to 1FH) under TRAP instruction.  
 cccc: 4-bit data operand specifies condition code under CMOV, SASF, and SETF instructions.  
 Assigned as 5-bit immediate data by adding 1-bit 0 to the highest bit.

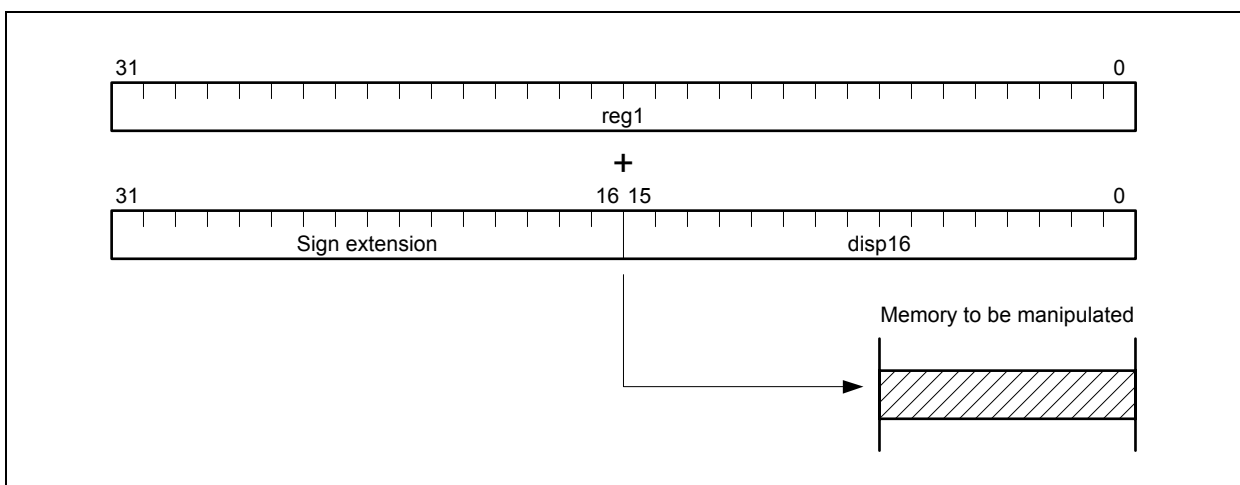
### (3) Based addressing

The following two types of based addressing are supported:

#### (a) Type 1

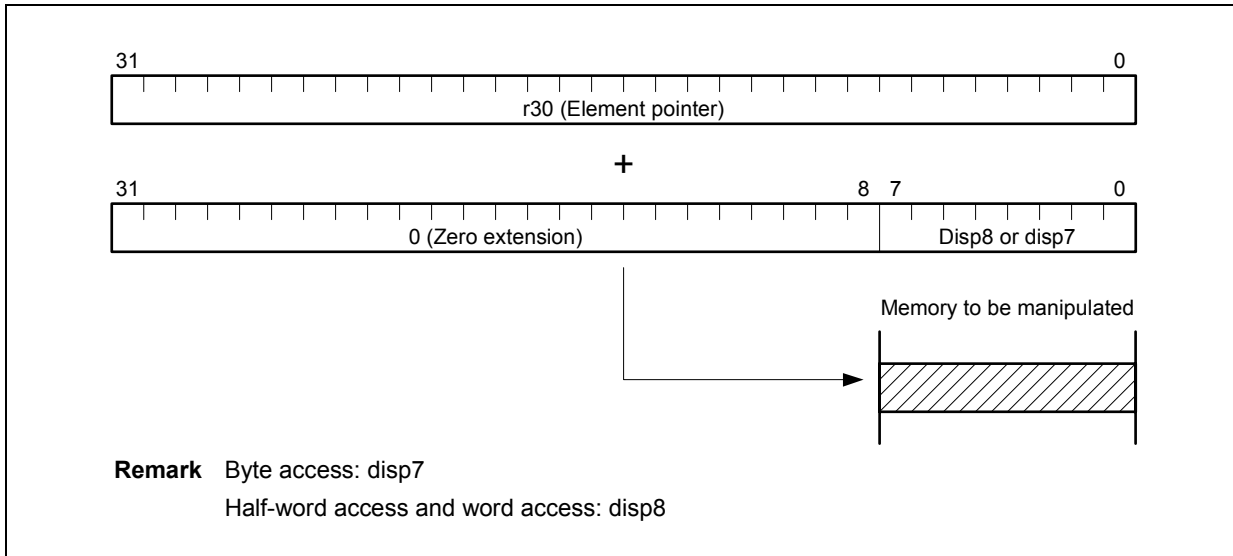
The address of the data memory location to be accessed is determined by adding the value in the specified general register (reg1) to the 16-bit displacement data (disp16) contained in the instruction code. This addressing mode applies to instructions using the operand format disp16 [reg1].

**Figure 4-5. Based Addressing (Type 1)**

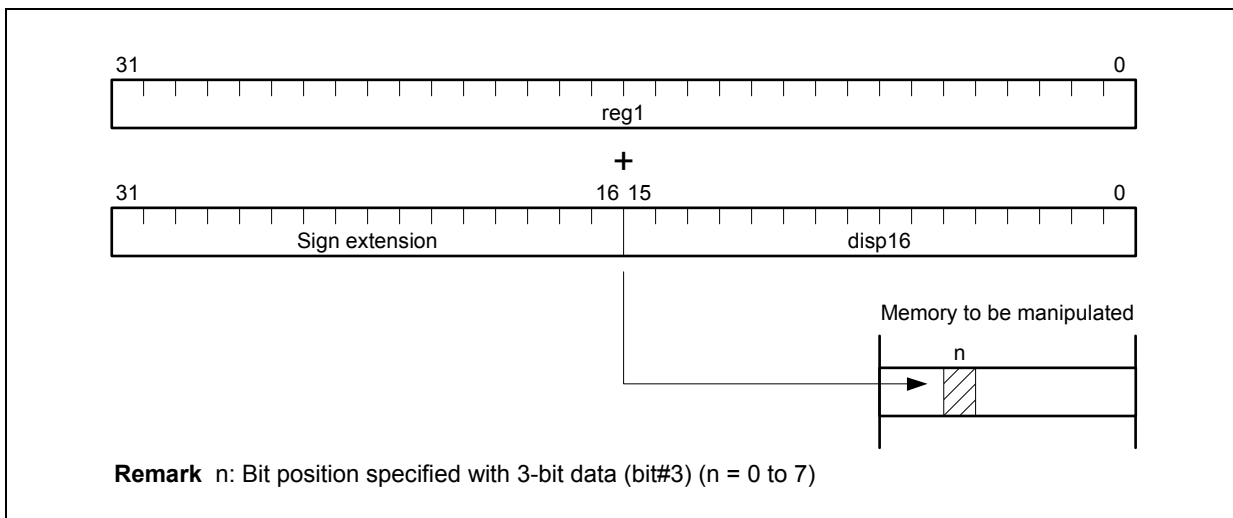


**(b) Type 2**

The address of the data memory location to be accessed is determined by adding the value in the element pointer (r30) to the 7- or 8-bit displacement data (disp7, disp8). This addressing mode applies to SLD and SST instructions.

**Figure 4-6. Based Addressing (Type 2)****(4) Bit addressing**

This addressing is for accessing 1 bit (specified with bit#3 of 3-bit data) among 1 byte of the memory space to be manipulated. It uses an operand address, expressed by the sum of the contents of a general register (reg1) and a 16-bit displacement (disp16) data, sign-extended to word length. This addressing mode applies only to the bit manipulation instructions.

**Figure 4-7. Bit Addressing**

# CHAPTER 5 INSTRUCTIONS

## 5. 1 Instruction Formats

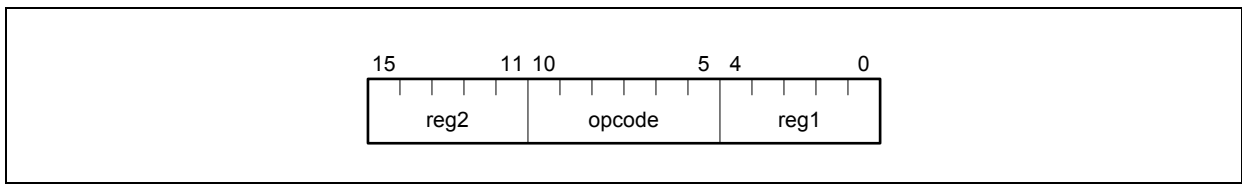
There are two types of instruction formats: 16-bit and 32-bit. The 16-bit format instructions include binary operations, controls, and conditional branch operations; and the 32-bit format instructions include loading/storing, jump operations, and 16-bit immediate data operations. An instruction is stored in memory as follows:

- Lower bytes of instruction (including bit 0) → lower address
- Higher bytes of instruction (including bit 16 or bit 31) → higher address

**Caution** Some instructions have an unused field (RFU). This field is reserved for future expansion only and must be fixed to "0."

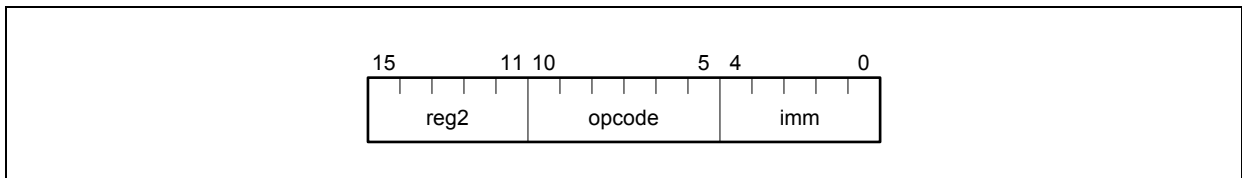
### (1) reg-reg instruction (Format I)

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



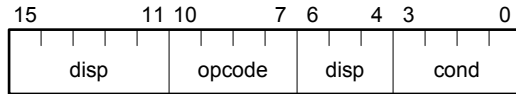
### (2) imm-reg instruction (Format II)

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



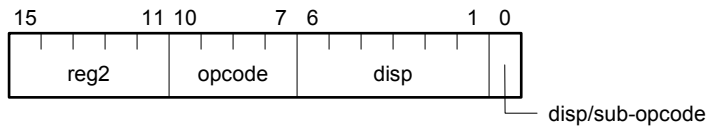
**(3) Conditional branch instruction (Format III)**

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

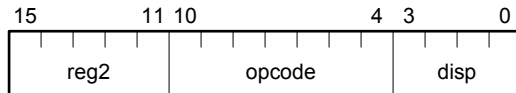


**(4) 16-bit load/store instruction (Format IV)**

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

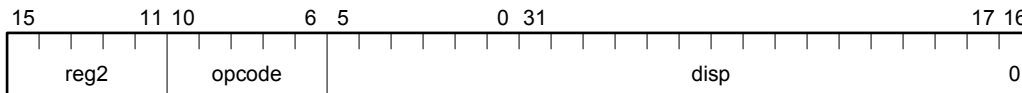


A 16-bit instruction format consists of a 7-bit opcode field, a general register specification field, and a 4-bit displacement field.



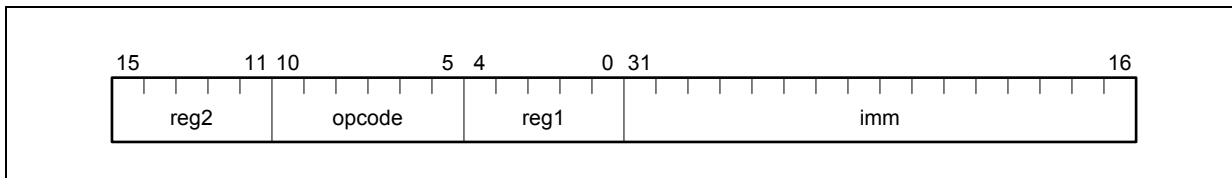
**(5) Jump instruction (Format V)**

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



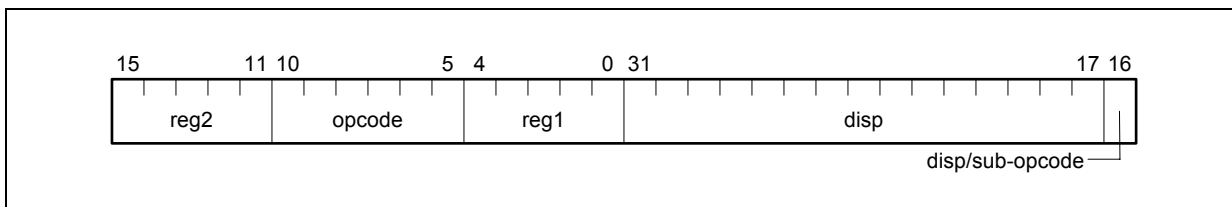
**(6) 3-operand instruction (Format VI)**

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



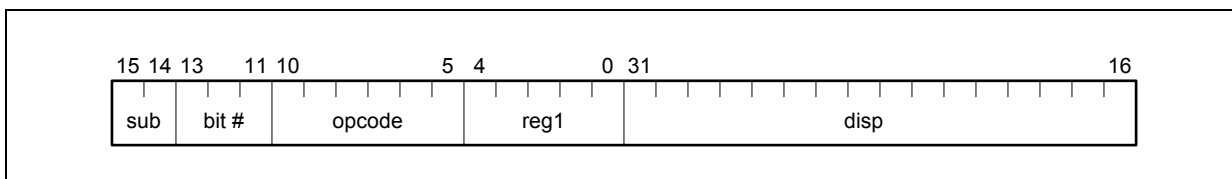
**(7) 32-bit load/store instruction (Format VII)**

A 32-bit instruction format consists of a 6-bit opcode field, two general register specification fields, and a 16-bit displacement field (or 15-bit displacement field + 1-bit sub-opcode field).



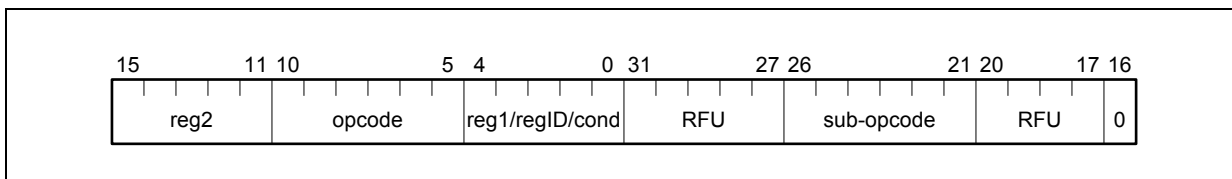
**(8) Bit manipulation instruction (Format VIII)**

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



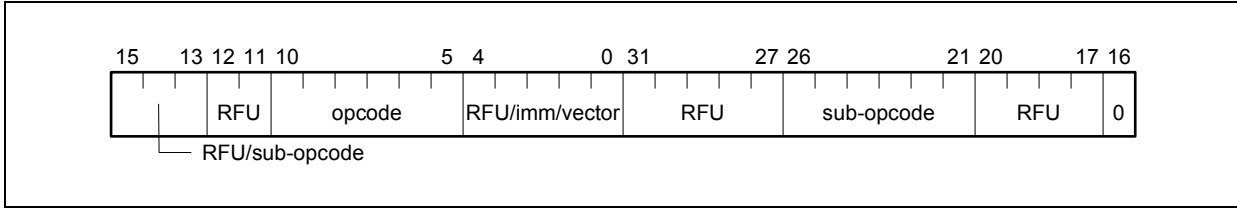
**(9) Extended instruction format 1 (Format IX)**

A 32-bit instruction format consists of a 6-bit opcode field, 6-bit sub-opcode field, and two general register specification fields. One of the fields may be register number field (regID) or condition code field (cond).



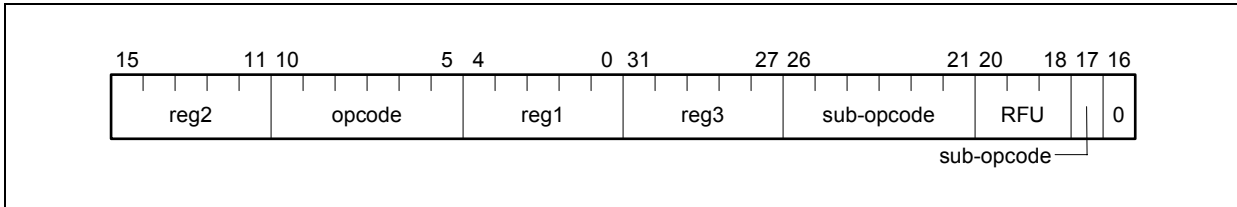
**(10) Extended instruction format 2 (Format X)**

A 32-bit instruction format consists of a 6-bit opcode field and 6-bit sub-opcode field.



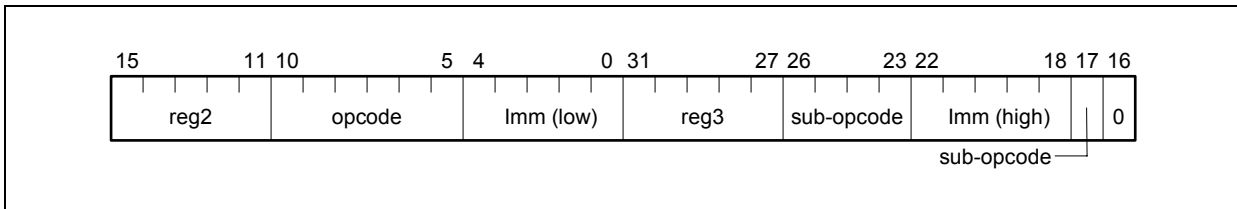
**(11) Extended instruction format 3 (Format XI)**

A 32-bit instruction format consists of a 6-bit opcode field, 6-bit and 1-bit sub-opcode field, and three general register specification fields.



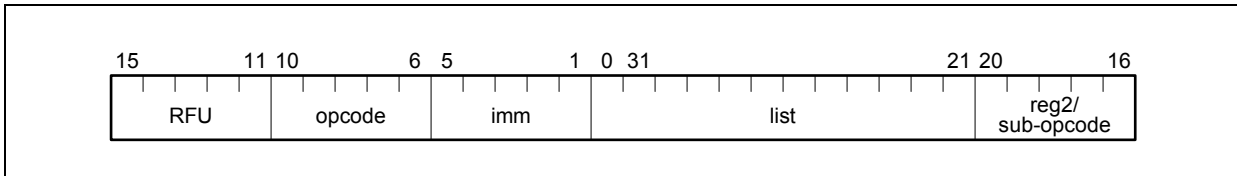
**(12) Extended instruction format 4 (Format XII)**

A 32-bit instruction format consists of a 6-bit opcode field, 4-bit and 1-bit sub-opcode field, 10-bit immediate field, and two general register specification fields.



**(13) Stack manipulation instruction format 1 (Format XIII)**

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



## 5.2 Outline 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 half-word
- LD.HU: Load half-word 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 half-word
- SLD.HU: Short format load half-word 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 half-word
- ST.W: Store word

#### (b) SST instructions

- SST.B: Short format store byte
- SST.H: Short format store half-word
- 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 half-word
- MULHI: Multiply half-word immediate
- MULU: Multiply word unsigned

**(4) Multiplication with addition 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 half-word
- 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) Saturate instructions:**

Execute saturate addition and subtraction. If the operation result exceeds the maximum positive value (7FFFFFFH), 7FFFFFFH returns. If the operation result exceeds the maximum negative value (80000000H), 80000000H returns. The following instructions (mnemonics) are provided.

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



**(8) Logical instructions:**

Include logical operation and shift 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 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 half-word
- BSW: Byte swap word
- CMOV: Conditional move
- HSH: Half-word swap half-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 half-word
- ZXB: Zero-extend byte
- ZXH: Zero-extend half-word

**(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 subtraction. The following instructions (mnemonics) are provided.

- DIV: Divide word
- DIVH: Divide half-word
- DIVHU: Divide half-word unsigned
- DIVU: Divide word unsigned

**(12) 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

**(13) 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

**(14) 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
- CTRET: Return from CALLT
- DI: Disable interrupt
- DISPOSE: Function dispose
- EI: Enable interrupt
- HALT: Halt
- LDSR: Load system register
- NOP: No operation
- PREPARE: Function prepare
- RETI: Return from trap or interrupt
- STSR: Store system register
- SWITCH: Jump with table look up
- TRAP: Trap

**(15) Debug function instructions:**

These instructions are instructions reserved for debug function. The following instructions (mnemonics) are provided.

- DBRET: Return from debug trap
- DBTRAP: Debug trap

## 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 Instruction Formats**).
- **Opcode:** Indicates the bit field of the instruction opcode (for symbols, refer to **Table 5-3**).
- **Flag:** Indicates the flag change 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 register (as source register)
reg2	General register (primarily as destination register with some as source registers)
reg3	General register (primarily used to store the remainder of a division result and/or the higher 32 bits of a multiply result)
bit#3	3-bit data to specify bit number
imm×	×-bit immediate data
disp×	×-bit displacement data
regID	System register number
vector	5-bit data to specify trap vector (00H to 1FH)
cccc	4-bit data to specify condition code
sp	Stack pointer (r3)
ep	Element pointer (r30)
list12	Lists of registers

Table 5-2. Conventions of Operation

Symbol	Meaning
←	Assignment
GR [ ]	General 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
load-memory-bit (a, b)	Reads bit b from address a
store-memory-bit (a, b, c)	Writes c to bit b of address a
saturate (n)	Performs saturate processing of "n." If $n \geq 7FFFFFFFH$ , $n = 7FFFFFFFH$ . If $n \leq 80000000H$ , $n = 80000000H$ .
result	Outputs results on flag
Byte	Byte (8 bits)
Half-word	Half-word (16 bits)
Word	Word (32 bits)
+	Add
–	Subtract
	Bit concatenation
×	Multiply
÷	Divide
%	Remainder of division results
AND	And
OR	Or
XOR	Exclusive Or
NOT	Logical negate
logically shift left by	Logical left-shift
logically shift right by	Logical right-shift
arithmetically shift right by	Arithmetic right-shift

**Table 5-3. Conventions of Opcode**

Symbol	Meaning
R	1-bit data of code specifying reg1 or regID
r	1-bit data of code specifying reg2
w	1-bit data of code specifying reg3
D	1-bit data of displacement (indicates higher bits of displacement)
d	1-bit data of displacement
I	1-bit data of immediate (indicates higher bits of immediate)
i	1-bit data of immediate
cccc	4-bit data for condition code specification (Refer to <b>Table 5-4. Conditions code</b> )
CCCC	4-bit data for condition code specification of Bcond instruction
bbb	3-bit data for bit number specification
L	1-bit data of code specifying general 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. Conditions code**

Conditions code (cccc)	Conditions formula
0000	OV = 1
1000	OV = 0
0001	CY = 1
1001	CY = 0
0010	Z = 1
1010	Z = 0
0011	(CY or Z) = 1
1011	(CY or Z) = 0
0100	S = 1
1100	S = 0
0101	always (Unconditional)
1101	SAT = 1
0110	(S xor OV) = 1
1110	(S xor OV) = 0
0111	((S xor OV) or Z) = 1
1111	((S xor OV) or Z) = 0

<Arithmetic instruction>

<b>ADD</b>	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)	rrrrr	001110RRRRR

	15	0
(2)	rrrrr	010010iiii

[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 the general register reg1 to the word data of the general register reg2 and stores the result in the general register reg2. The data of the general register reg1 is not affected.  
(2) Adds the 5-bit immediate data, sign-extended to word length, to the word data of the general register reg2 and stores the result in the general register reg2.

<Arithmetic instruction>

<b>ADDI</b>	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
rrrrr110000RRRRRiiiiiiiiiiiiiiiiiii			

[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 the general register reg1 and stores the result in the general register reg2. The reg1 data is not affected.



Add on condition flag

[Description] In cases where the result of adding the word data of general purpose register reg1 to the word data of general purpose register reg2 satisfies the condition designated in condition code [cccc], 1 is added, and in cases where it does not, 0 is added; the result is then stored in general purpose register reg3. General-purpose register reg2 is not affected. Please designate one of the condition codes shown in the following table as [cccc]. (However, cccc cannot equal 1101.)

Condition Code	Condition Formula	Condition Code	Condition Formula
0000	OV = 1	0100	S = 1
1000	OV = 0	1100	S = 0
0001	CY = 1	0101	Always (Unconditional)
1001	CY = 0	0110	(S xor OV) = 1
0010	Z = 1	1110	(S xor OV) = 0
1010	Z = 0	0111	( (S xor OV) or Z ) = 1
0011	(CY or Z) = 1	1111	( (S xor OV) or Z ) = 0
1011	(CY or Z) = 0	(1101)	Assignment Inhibited

AND	AND
	And

<Logical instruction>

<b>ANDI</b>	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
rrrrr	110110	RRRRR	iiiiiiiiiiiiiiiiiii

[Flags]

CY    --

OV    0

S     "1" if operation result's word data MSB is "1"; otherwise, "0".

Z     "1" if the operation result is "0"; otherwise, "0".

SAT   --

[Description]           ANDs the word data of the general register reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in the general register reg2. The reg1 data is not affected.

<Branch instruction>

## Bcond

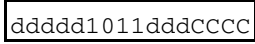
Branch on condition code with 9-bit displacement

Branch on Condition Code

[Instruction format] Bcond disp9

[Operation] if conditions are satisfied  
then  $PC \leftarrow PC + \text{sign-extend}(\text{disp9})$

[Format] Format III

[Opcode] 15 0  
  
 ddddddd is the higher 8 bits of disp9.

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

[Description] Checks each PSW flag specified by the instruction and branches if a condition is met; executes otherwise the next instruction. The PC branch destination is the sum of the current PC value and the 9-bit displacement (= 8-bit immediate data shifted by 1 and sign-extended to word length).

[Comment] Bit 0 of the 9-bit displacement is masked to "0". The current PC value used for calculation is the address of the first byte of this instruction. The displacement value being "0" signifies that the branch destination is the instruction itself.

Table 5-5. Bcond Instructions

Instruction		Condition Code (cccc)	Flag Status	Branch Condition
Signed integer	BGE	1110	(S xor OV) = 0	Greater than or equal to signed
	BGT	1111	( (S xor OV) or Z) = 0	Greater than signed
	BLE	0111	( (S xor OV) or Z) = 1	Less than or equal to signed
	BLT	0110	(S xor OV) = 1	Less than signed
Unsigned integer	BH	1011	(CY or Z) = 0	Higher (Greater than)
	BL	0001	CY = 1	Lower (Less than)
	BNH	0011	(CY or Z) = 1	Not higher (Less than or equal)
	BNL	1001	CY = 0	Not lower (Greater than or equal)
Common	BE	0010	Z = 1	Equal
	BNE	1010	Z = 0	Not equal
Others	BC	0001	CY = 1	Carry
	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
	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 saturate instruction sets "1" to the SAT flag. In normal operations, if an overflow occurs, the S flag is inverted (0 → 1 or 1 → 0). This is because the result is a negative value if it exceeds the maximum positive value and it is a positive value if it exceeds the maximum negative value. However, when a saturate 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.

&lt;Data operation instruction&gt;

**BSH**

Byte swap half-word

Byte swap of half-word

[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

15	0	31	16
rrrrr11111100000 wwwww01101000010			

[Flags]

CY "1" when at there is at least one byte value of zero in the lower half-word of the operation result; otherwise, "0".

OV 0

S "1" if operation result's word data MSB is "1"; otherwise, "0".

Z "1" when lower half-word of operation result is "0"; otherwise, "0".

SAT --

[Description] This instruction performs endian conversion.

<Data manipulation instruction>

<b>BSW</b>	Byte swap word
	Byte Swap Word

[Instruction format] BSW reg2, reg3

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

[Format] Format XII

[Opcode]

15	0 31	16
rrrrr11111100000	www01101000000	

[Flags]

CY "1" when at 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's word data MSB is "1"; otherwise, "0".

Z "1" if operation result's word data is "0"; otherwise, "0".

SAT --

[Description] Executes endian translation.

&lt;Special instruction&gt;

**CALLT**

Call with table look up

Subroutine Call with Table Look Up

[Instruction format] CALLT imm6

[Operation] CTPC  $\leftarrow$  PC + 2 (return PC)  
 CTPSW  $\leftarrow$  PSW  
 adr  $\leftarrow$  CTBP + zero-extend (imm6 logically shift left by 1)  
 PC  $\leftarrow$  CTBP + zero-extend (Load-memory (adr, Half-word) )

[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 half-word 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) Branches to the target address generated in step (4).

[Caution] When an interrupt occurs during the CALLT instruction execution, the execution is aborted after the end of the read/write cycle.



&lt;Bit manipulation instruction&gt;

<b>CLR1</b>	Clear bit
	Clear Bit

[Instruction format] (1) CLR1 bit#3, disp16 [reg1]  
 (2) CLR1 reg2, [reg1]

[Operation] (1)  $\text{adr} \leftarrow \text{GR}[\text{reg1}] + \text{sign-extend}(\text{disp16})$   
 $\text{Z flag} \leftarrow \text{Not}(\text{Load-memory-bit}(\text{adr}, \text{bit\#3}))$   
 $\text{Store-memory-bit}(\text{adr}, \text{bit\#3}, 0)$   
 (2)  $\text{adr} \leftarrow \text{GR}[\text{reg1}]$   
 $\text{Z flag} \leftarrow \text{Not}(\text{Load-memory-bit}(\text{adr}, \text{reg2}))$   
 $\text{Store-memory-bit}(\text{adr}, \text{reg2}, 0)$

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

[Opcode]

	15	0 31	16
(1)	10bbb111110RRRRR dddddddddddddddd		
	15	0 31	16
(2)	rrrrr111111RRRRR 0000000011100100		

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

[Description] (1) Adds the word data of general-purpose register reg1 to the 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Then reads the byte data referenced by the generated address, clears the bit specified by the 3-bit bit number, and writes back to the original address.  
 (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Then reads the byte data referenced by the generated address, clears the bit specified by the data of the lower 3 bits of reg2, and writes back to the original address.

[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 after this instruction has been executed.

&lt;Data operation instruction&gt;

**CMOV**

Conditional move

Conditional move

[Instruction format] (1) CMOV cccc, reg1, reg2, reg3  
 (2) CMOV cccc, imm5, reg2, reg3

[Operation] (1) if conditions are satisfied  
                   then GR [reg3]  $\leftarrow$  GR [reg1]  
                   else GR [reg3]  $\leftarrow$  GR [reg2]  
 (2) if conditions are satisfied  
                   then GR [reg3]  $\leftarrow$  sign-extended (imm5)  
                   else GR [reg3]  $\leftarrow$  GR [reg2]

[Format] (1) Format XI  
 (2) Format XII

[Opcode]

	15	0	31	16
(1)	rrrrr11111RRRR www011001cccc0			
	15	0	31	16
(2)	rrrrr11111iiii 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 conditions codes shown in the following table as "cccc".

Condition code	Condition formula	Condition code	Condition formula
0000	OV = 1	0100	S = 1
1000	OV = 0	1100	S = 0
0001	CY = 1	0101	Always (unconditional)
1001	CY = 0	1101	SAT = 1
0010	Z = 1	0110	(S xor OV) = 1
1010	Z = 0	1110	(S xor OV) = 0
0011	(CY or Z) = 1	0111	((S xor OV) or Z) = 1
1011	(CY or Z) = 0	1111	((S xor OV) or Z) = 0

- (2) When the condition specified by condition code "cccc" is met, 5-bit immediate data with word-length code extension 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 conditions codes shown in the following table as "cccc".

Condition code	Condition formula	Condition code	Condition formula
0000	$OV = 1$	0100	$S = 1$
1000	$OV = 0$	1100	$S = 0$
0001	$CY = 1$	0101	Always (unconditional)
1001	$CY = 0$	1101	$SAT = 1$
0010	$Z = 1$	0110	$(S \text{ xor } OV) = 1$
1010	$Z = 0$	1110	$(S \text{ xor } OV) = 0$
0011	$(CY \text{ or } Z) = 1$	0111	$((S \text{ xor } OV) \text{ or } Z) = 1$
1011	$(CY \text{ or } Z) = 0$	1111	$((S \text{ xor } OV) \text{ or } Z) = 0$

[Comment]

See the description of the SETF instruction.

&lt;Arithmetic instruction&gt;

**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] (1) 

15	0
rrrrr001111RRRRR	

(2) 

15	0
rrrrr010011iiii	

[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 the general register reg2 with the word data of the general register reg1 and outputs the result through the PSW flags. Comparison is enabled by subtracting the reg1 contents from the reg2 word data. The reg1 data and reg2 data are not affected.

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

<Special instruction>

<b>CTRET</b>	Return from CALLT
	Return from CALLT

[Instruction format] CTRET

[Operation] PC ← CTPC  
PSW ← CTPSW

[Format] Format X

[Opcode]

15	0	31	16
00000011111100000		0000000101000100	

[Flags]

CY	Value read from CTPSW is restored.
OV	Value read from CTPSW is restored.
S	Value read from CTPSW is restored.
Z	Value read from CTPSW is restored.
SAT	Value read from CTPSW is restored.

[Description] Fetches 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 read from the CTPC and CTPSW.
- (2) The values are restored to PC and PSW and the control is transferred to the return address.

&lt;Debug function instruction&gt;

**DBRET**

Return from debug trap

Return from debug trap

[Instruction format] DBRET

[Operation] PC ← DBPC  
PSW ← DBPSW

[Format] Format X

[Opcode] 15 0 31 16  

0000011111100000	0000000101000110
------------------	------------------

[Flags] CY Value read from DBPSW is restored.  
OV Value read from DBPSW is restored.  
S Value read from DBPSW is restored.  
Z Value read from DBPSW is restored.  
SAT Value read from DBPSW is restored.

[Description] Fetches the return PC and PSW from the appropriate system register and returns from the debug mode.

[Caution] Because the DBRET instruction is for debugging, it is essentially used by debug tools. When a debug tool is using this instruction, therefore, use of it in the application may cause a malfunction.

&lt;Debug function instruction&gt;

<b>DBTRAP</b>	Debug trap
	Debug trap

[Instruction format] DBTRAP

[Operation] DBPC  $\leftarrow$  PC + 2 (return PC)  
 DBPSW  $\leftarrow$  PSW  
 PSW.NP  $\leftarrow$  1  
 PSW.EP  $\leftarrow$  1  
 PSW.ID  $\leftarrow$  1  
 DIR.DM  $\leftarrow$  1  
 PC  $\leftarrow$  00000060H

[Format] Format I

[Opcode] 15 0  
 1111100001000000

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

[Description] The following steps are taken:

- (1) Saves the contents of both the return PC (the one subsequent to DBTRAP instruction) and the current PSW in DBPC and DBPSW, respectively.
- (2) Sets "1" to the flags of NP, EP, and ID as well as the DM bit on the DIR register.
- (3) Sets the exception trap handler address (00000060H) to PC to move to the debug mode.

PSW flags, other than NP, EP, and ID, are not affected. Note that the value saved in DBPC is the address of the instruction subsequent to the DBTRAP instruction.

[Caution] Because the DBTRAP instruction is for debugging, it is essentially used by debug tools. When a debug tool is using this instruction, therefore, use of it in the application may cause a malfunction.

&lt;Special instruction&gt;

**DI**

Disable interrupt

Disable Interrupt

[Instruction format] DI

[Operation] PSW.ID  $\leftarrow$  1 (Disables maskable interrupt)

[Format] Format X

15	0 31	16
00000111111000000000000101100000		

[Flags]	CY	--
	OV	--
	S	--
	Z	--
	SAT	--
	ID	1

[Description] Sets "1" to the ID flag of PSW to immediately disable the acknowledgement of maskable interrupts.

[Comment] Interrupts are not sampled during the DI instruction execution, thereby immediately disabling the interrupts. The PSW flag becomes valid only after the next instruction commences. Non-maskable interrupts (NMI) are not affected by this instruction.



&lt;Special instruction&gt;

**DISPOSE**

Function dispose

Stack frame deletion

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

[Operation] (1)  $sp \leftarrow sp + \text{zero-extend}(\text{imm5 logically shift left by } 2)$   
 $GR[\text{reg in list12}] \leftarrow \text{Load-memory}(sp, \text{Word})$   
 $sp \leftarrow sp + 4$   
 repeat 2 steps above until all regs in list12 is loaded  
 (2)  $sp \leftarrow sp + \text{zero-extend}(\text{imm5 logically shift left by } 2)$   
 $GR[\text{reg in list12}] \leftarrow \text{Load-memory}(sp, \text{Word})$   
 $sp \leftarrow sp + 4$   
 repeat 2 states above until all regs in list12 is loaded  
 $PC \leftarrow GR[\text{reg1}]$

[Format] Format XIII

[Opcode]

15	0 31	16
(1) 0000011001iiiiL	LLLLLLLLLLLL	00000

15	0 31	16
(2) 0000011001iiiiL	LLLLLLLLLLLLRRRRR	

The values of RRRRR must be other than "00000".

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.

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

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

- When all of the register's non-corresponding bits are "0": 08000001H
- When all of the register's non-corresponding bits are "1": 081FFFFFH

[Flags]	CY     -- OV     -- S       -- Z       -- SAT    --
[Description]	<p>(1) Adds the 5-bit immediate data, logically left-shifted by 2 and zero-extended to word length, to sp; returns to the general registers listed in list12 by loading the data from the address specified by sp and adds 4 to sp. Bit 0 of the address is masked to "0".</p> <p>(2) Adds the 5-bit immediate data, logically left-shifted by 2 and zero-extended to word length, to sp; returns to the general 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 the general register reg1. Bit 0 of the address is masked to "0".</p>
[Comment]	<p>General 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-bit of address specified by sp is always masked to "0" even if misaligned access is enabled.</p> <p>An interrupt, occurred before updating sp, aborts the execution to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. sp retains its original values.</p>
[Caution]	<p>If an interrupt is generated during instruction execution, due to manipulation of the stack, the execution of that instruction may stop after the read/write cycle and register value rewriting are complete. Execution is resumed after returning from the interrupt.</p>

&lt;Divide instruction&gt;

<b>DIV</b>	Divide word
	Divide Word

[Instruction format] DIV reg1, reg2, reg3

[Operation] GR [reg2]  $\leftarrow$  GR [reg2]  $\div$  GR [reg1]  
 GR [reg3]  $\leftarrow$  GR [reg2]  $\%$  GR [reg1]

[Format] Format XI

[Opcode] 15 0 31 16

rrrrr111111RRRRR	www01011000000
------------------	----------------

[Flags] CY --  
 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] Divides the word data of the general register reg2 by the word data of the general register reg1 and stores the quotient to the general register reg2 with the remainder to the general register reg3. The data divided by 0 results in overflow with the quotient being undefined. The reg1 data is not affected.

[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. When an interrupt occurs during the DIV instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. The general register reg1 and the general register reg2 retain their initial value. If reg2 and reg3 share the same address, the remainder is stored to reg2 (= reg3).

&lt;Divide instruction&gt;

**DIVH**

Divide half-word

Divide Half-word

[Instruction format] (1) DIVH reg1, reg2  
 (2) DIVH reg1, reg2, reg3

[Operation] (1)  $GR[reg2] \leftarrow GR[reg2] \div GR[reg1]$   
 (2)  $GR[reg2] \leftarrow GR[reg2] \div GR[reg1]$   
 $GR[reg3] \leftarrow GR[reg2] \% GR[reg1]$

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

[Opcode]

(1) 

15	0
rrrrr000010RRRRR	

(2) 

15	0	31	16
rrrrr11111RRRRR	www	ww	01010000000

[Flags]

CY --

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) Divides the word data of the general register reg2 by the lower half-word data of the general register reg1 and stores the quotient to the general register reg2. The data divided by 0 results in overflow with the quotient being undefined. The reg1 data is not affected.

(2) Divides the word data of the general register reg2 by the lower half-word data of the general register reg1 and stores the quotient to the general register reg2 with the remainder set to the general register reg3. The data divided by 0 results in overflow with the quotient being undefined. The reg1 data is not affected.

[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 interrupt occurs during the DIVH instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. The general register reg1 and the general register reg2 retain their initial value. Do not specify r0 as the destination register reg2. The higher 16 bits on the general register reg1 are ignored during the divide execution.
- (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.  
When an interrupt occurs during the DIVH instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. The general register reg1 and the general register reg2 retain their initial value. Do not specify r0 as the destination register reg2. The higher 16 bits on the general register reg1 are ignored during the divide execution. If reg2 and reg3 share the same address, the remainder is stored to reg2 (= reg3).

&lt;Divide instruction&gt;

**DIVHU**

Divide half-word unsigned

Divide Half-word Unsigned

[Instruction format] DIVHU reg1, reg2, reg3

[Operation] GR [reg2]  $\leftarrow$  GR [reg2]  $\div$  GR [reg1]  
 GR [reg3]  $\leftarrow$  GR [reg2]  $\%$  GR [reg1]

[Format] Format XI

[Opcode] 15 0 31 16

rrrrr11111RRRR	www0101000010
----------------	---------------

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

[Description] Divides the word data of the general register reg2 by the lower half-word data of the general register reg1 and stores the quotient to the general register reg2 with the remainder set to the general register reg3. The data divided by 0 results in overflow with the quotient being undefined. The reg1 data is not affected.

[Comment] Overflow occurs when data is divided by 0 (in which case the quotient is undefined).  
 When an interrupt occurs during the DIVHU instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. The general register reg1 and the general register reg2 retain their initial value. If reg2 and reg3 share the same address, the remainder is stored to reg2 (= reg3).

&lt;Divide instruction&gt;

**DIVU**

Divide word unsigned

Divide Word Unsigned

[Instruction format] DIVU reg1, reg2, reg3

[Operation] GR [reg2]  $\leftarrow$  GR [reg2]  $\div$  GR [reg1]  
 GR [reg3]  $\leftarrow$  GR [reg2]  $\%$  GR [reg1]

[Format] Format XI

[Opcode]                      15                      0 31                      16

rrrrr	111111RRRRR	www01011000010
-------	-------------	----------------

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

[Description]               Divides the word data of the general register reg2 by the lower half-word data of the general register reg1 and stores the quotient to the general register reg2 with the remainder set to the general register reg3. The data divided by 0 results in overflow with the quotient being undefined. The reg1 data is not affected.

[Comment]                   Overflow occurs when data is divided by 0 (in which case the quotient is undefined).  
 When an interrupt occurs during the DIVU instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt. The general register reg1 and the general register reg2 retain their initial value. If reg2 and reg3 share the same address, the remainder is stored to reg2 (= reg3).

<Special instruction>

<b>EI</b>	Enable interrupt
	Enable Interrupt

[Instruction format] EI

[Operation] PSW.ID ← 0 (enables maskable interrupt)

[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 interrupts starting the next instruction.

[Comment] Interrupts are not sampled during the EI instruction execution.



[Instruction format]    HALT

[Operation]	Halt
-------------	------

[Format]	Format X
----------	----------

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

[Description]	Stops the CPU operating clock and places the system in the HALT mode.
---------------	---

[Comment]	<p>The HALT mode can be released by any of the following three events.</p> <ul style="list-style-type: none"> <li>• Reset input</li> <li>• Non-maskable interrupt request</li> <li>• Unmasked maskable interrupt request</li> </ul>
-----------	---

If an interrupt is acknowledged during the HALT mode, the next instruction address is stored to EIPC or FEPC.

<Data Manipulation Instructions>

<b>HSH</b>	Half-word swap half-word
	Half-word Swap Half-word

[Instruction format] HSH reg2, reg3

[Operation] GR [reg3] ← GR [reg2]

[Format] Format XII

[Opcode]

15	0 31	16
rrrrr	1111100000	www01101000110

[Flags]

CY	"1" if the lower half-word of the operation result is "0"; otherwise, "0".
OV	0
S	"1" if operation result's word data MSB is "1"; otherwise, "0".
Z	"1" if the lower half-word of the operation result is "0"; otherwise, "0".
SAT	--

[Description] The content of general-purpose register reg2 is stored in general-purpose register reg3, and the flag judgment result is stored in PSW.

<Data manipulation instruction>

<b>HSW</b>	Half-word swap word  Half-word Swap Word
------------	--

[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	www01101000100
------------------	----------------

[Flags]

CY	"1" when at there is at least one half-word of zero in the word data of the operation result; otherwise, "0".
OV	0
S	"1" if operation result's word data MSB is "1"; otherwise, "0".
Z	"1" if operation result's word data is "0"; otherwise, "0".
SAT	--

[Description]           Executes endian translation.

&lt;Branch instruction&gt;

**JARL**

Jump and register link

Jump and Register Link

[Instruction format] (1) JARL disp22, reg2

(2) JARL disp32, reg1

[Operation] (1) GR [reg2]  $\leftarrow$  PC + 4  
                   PC  $\leftarrow$  PC + sign-extend (disp22)  
 (2) GR [reg1]  $\leftarrow$  PC + 6  
                   PC  $\leftarrow$  PC + disp32

[Format] (1) Format V  
 (2) Format VI

[Opcode]

(1) 

15	0 31	16
rrrrrr11110	ddddd	ddddd0

  
       ddddd is the higher 21 bits of disp22.

(2) 

15	0 31	16 47	32
00000010111	RRRRR	ddddd0	DDDDDDDDDDDDDDDDDD

  
       DDDDDDDDDDDDDDDDDD is the higher 31 bits of disp32.

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

[Description] (1) Saves the current PC value+4 in the general 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 the general 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 functions as a call-subroutine instruction, and saves the return PC address in either reg1 or reg2. JMP instruction functions as a subroutine-return instruction, and can be used to specify the general register containing the return address as reg1 to the return PC.

<Branch instruction>

<b>JMP</b>	Jump register
	Jump Register

[Instruction format] (1) JMP [reg1]  
(2) JMP disp32 [reg1]

[Operation] (1) PC ← GR [reg1]  
(2) PC ← GR [reg1] + disp32

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

[Opcode]

(1) 

15	0
00000000011RRRRR	

(2) 

15	0	31	16	47	32
00000110111RRRRR		ddddddddddddddd0	DDDDDDDDDDDDDDDDDD		

  
DDDDDDDDDDDDDDDDDD is the higher 31 bits of disp32.

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

[Description] (1) Transfers the control to the address specified by the general register reg1. Bit 0 of the address is masked to "0".  
(2) Adds the 32-bit displacement to the general register reg1, and transfers the control to the resulting address specified by the general register reg1. Bit 0 of the address is masked to "0".

[Comment] Using this instruction as the subroutine-return instruction requires the return PC to be specified by the general register reg1.

<Branch instruction>

<b>JR</b>	Jump relative
	Jump 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	ddddd	d	0

ddddddddddddddddddd is the higher 21 bits of disp22.

(2) 

15	0	31	16	47	32
000001011100000	d	d	d	d	d

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>

<b>LD.B</b>	Load byte  Load byte
-------------	----------------------------

[Instruction format] LD.B disp16 [reg1] , reg2

[Operation] adr ← GR [reg1] + sign-extend (disp16)  
GR [reg2] ← sign-extend (Load-memory (adr, Byte) )

[Format] Format VII

[Opcode]

15	0	31	16
rrrrr	111000	RRRRR	dddddddddddddddd

[Flags]

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

[Description] Adds the word data of the general 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 to the general register reg2.

[Comment]

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

<Load instruction>

<b>LD.BU</b>	Load byte unsigned
	Load byte unsigned

[Instruction format] LD.BU disp16 [reg1] , reg2

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

[Format] Format VII

[Opcode]

15	0 31	16
rrrrrr11110bRRRRR	dddddddddddddddl	

dddddddddddddd is the higher 15 bits of disp16. b is the bit 0 of disp16.

[Flags]

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

[Description] Adds the word data of the general 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 to the general register reg2.  
 Don't specify 0 to be reg2.

[Comment] The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.  
 If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.



<Load instruction>

<b>LD.H</b>	Load half-word
	Load half-word

[Instruction format] LD.H disp16 [reg1] , reg2

[Operation]  $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$   
 $GR [reg2] \leftarrow \text{sign-extend} (\text{Load-memory} (adr, \text{Half-word}) )$

[Format] Format VII

[Opcode]

15	0	31	16
rrrrr		111001RRRRR	dddddddddddddd0

dddddddddddddd is the higher 15 bits of disp16.

[Flags]

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

[Description] Adds the word data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, sign-extended to word length, and stored to the general register reg2.

[Caution] Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 is masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 is not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

<Load instruction>

**LD.HU**

Load half-word unsigned

Load half-word unsigned

[Instruction format] LD.HU disp16 [reg1] , reg2

[Operation]  $\text{adr} \leftarrow \text{GR}[\text{reg1}] + \text{sign-extend}(\text{disp16})$   
 $\text{GR}[\text{reg2}] \leftarrow \text{zero-extend}(\text{Load-memory}(\text{adr}, \text{Half-word}))$

[Format] Format VII

[Opcode]

15	0 31	16
rrrrr	11111RRRRR	dddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16.

[Flags]

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

[Description] Adds the word data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, sign-extended to word length, and stored to the general register reg2.  
 Don't specify 0 to be reg2.

[Caution] Adding the word data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 is masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 is not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.  
 If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

<Load instruction>

<b>LD.W</b>	Load word
	Load word

[Instruction format] LD.W disp16 [reg1] , reg2

[Operation]  $adr \leftarrow GR [reg1] + \text{sign-extend} (disp16)$   
 $GR [reg2] \leftarrow \text{Load-memory} (adr, \text{Word})$

[Format] Format VII

[Opcode]

15	0	31	16
rrrrr111001RRRRR		dddddddddddddd1	

dddddddddddddd is the higher 15 bits of disp16.

[Flags]

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

[Description] Adds the word data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Word data is read from the generated address, and stored to the general register reg2.

[Caution] Adding the word data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

&lt;Special instruction&gt;

**LDSR**

Load to system register

Load to System Register

[Instruction format] LDSR reg2, regID

[Operation] SR [regID] ← GR [reg2]

[Format] Format IX

15	0 31	16
rrrrr11111RRRR0000000000100000		

**Caution** The fields to define reg1 and reg2 are swapped in this instruction. “RRR” is normally used for reg1 and is the source operand with “rrr” representing reg2 and the destination operand. In this instruction, “RRR” is still the source operand, but is represented by reg2 with “rrr” being as the register destination, as labeled below:

**rrrrr: regID specification****RRRRR: reg2 specification**

[Flags]

CY --(Refer to **Comment** below.)

OV --(Refer to **Comment** below.)

S --(Refer to **Comment** below.)

Z --(Refer to **Comment** below.)

SAT --(Refer to **Comment** below.)

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

[Comment] If the system register number (regID) is 5 (PSW), the values of the corresponding bits of PSW are set according to the reg2 contents. This only affects the flag bits, and the reserved bits remain 0. Interrupts are not sampled when the PSW is updated, thereby immediately disabling any interrupt. The ID flag becomes valid only after the next instruction commences.

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

&lt;Multiplication with addition instruction&gt;

**MAC**

Multiply word and add

Multiplication with addition of (signed) word data

[Instruction format] MAC reg1, reg2, reg3, reg4

[Operation]  $GR[reg4+1] \parallel GR[reg4] \leftarrow GR[reg2] \times GR[reg1] + GR[reg3+1] \parallel GR[reg3]$ 

[Format] Format XI

15	0 31	16
rrrrr111111RRRRR	www0011110	mmmm0

[Flags]

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

[Description] The word data in general-purpose register reg2 is multiplied by the word data in general-purpose register reg1, then the result (64-bit data) is added to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose registers 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.

This instruction treats contents of general-purpose register reg1 and reg2 as 32-bit signed integer.

This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.

[Caution] The 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.

&lt;Multiplication with addition instruction&gt;

**MACU**

Multiply word unsigned and add

Multiplication with addition of (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

15	0 31	16
rrrrr11111RRRRR	www0011111mmmm0	

[Flags]

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

[Description]        The word data in general-purpose register reg2 is multiplied by the word data in general-purpose register reg1, then the result (64-bit data) is added to 64-bit data consisting of the lower 32 bits of general-purpose register reg3 and the data in general-purpose registers 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.

This instruction treats contents of general-purpose register reg1 and reg2 as 32-bit unsigned integer.

This has no effect on general-purpose register reg1, reg2, reg3, or reg3+1.

[Caution]            The 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>

<b>MOV</b>	Move register/immediate (5-bit) /immediate (32-bit)
	Move

[Instruction format] (1) MOV reg1, reg2  
(2) MOV imm5, reg2  
(3) MOV imm32, reg1

[Operation] (1) GR [reg2] ← GR [reg1]  
(2) GR [reg2] ← sign-extend (imm5)  
(3) GR [reg1] ← imm32

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

[Opcode]

(1) 

15	0
rrrrr000000RRRRR	

(2) 

15	0
rrrrr010000iiii	

(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) Transfers the word data of the general register reg1 to the general register reg2. The reg1 data is not affected.  
(2) Transfers the 5-bit immediate data, sign-extended to word length, to the general register reg2. Do not specify r0 as the destination register reg2.  
(3) Transfers the 32-bit immediate data to the general register reg1.

<Arithmetic instruction>

<b>MOVEA</b>	Move effective address
	Move Effective Address

[Instruction format]    MOVEA imm16, reg1, reg2

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

[Format]                Format VI

[Opcode]

15	0	31	16
rrrrr110001RRRRR		iiiiiiiiiiiiiiiiiii	

[Flags]

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

[Description]           Adds the 16-bit immediate data, sign-extended to word length, to the word data of the general register reg1 and stores the result in the general register reg2. Neither the reg1 data nor the flags is affected. Do not specify r0 as the destination register reg2.

[Comment]              This instruction is to execute a 32-bit address calculation with PSW flag value intact.



<Arithmetic instruction>

<b>MOVHI</b>	Move high half-word
	Move High Half-word

[Instruction format] MOVHI imm16, reg1, reg2

[Operation]  $GR[reg2] \leftarrow GR[reg1] + (imm16 \parallel 0^{16})$

[Format] Format VI

[Opcode]

15	0	31	16
rrrrr110010RRRRRiiiiiiiiiiiiiiiiii			

[Flags]

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

[Description] Adds the word data with its higher 16 bits specified by the 16-bit immediate data and the lower 16 bits being "0" to the word data of the general register reg1 and stores the result in the general register reg2. Neither the reg1 data nor the flags is affected. Do not specify r0 as the destination register reg2.

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

<Multiplication instruction>

# MUL

Multiply word by register/immediate (9-bit)

Multiplication of (signed) word data

[Instruction format] (1) MUL reg1, reg2, reg3  
(2) MUL imm9, reg2, reg3

[Operation] (1) GR [reg3] || GR [reg2] ← GR [reg2] × GR [reg1]  
(2) GR [reg3] || GR [reg2] ← GR [reg2] × sign-extend (imm9)

[Format] (1) Format XI  
(2) Format XII

[Opcode]

	15	0	31	16
(1)	rrrrr11111RRRRR wwwww01000100000			
	15	0	31	16
(2)	rrrrr11111iiii wwwww01001IIII00			

iiii are the lower 5 bits of 9-bit immediate data.  
IIII are the upper 4 bits of 9-bit immediate data.

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

[Description] (1) The word data in general-purpose register reg2 is multiplied by the word data in general-purpose register reg1, then upper 32 bits of the result (64-bit data) are stored in general-purpose register reg3 and the lower 32 bits are stored in general-purpose register reg2. This instruction treats contents of general-purpose register reg1 and reg2 as 32-bit signed integer. This has no effect on general-purpose register reg1.  
(2) The word data in general-purpose register reg2 is multiplied by 9-bit immediate data with word-length code extension, then upper 32 bits of the result (64-bit data) are stored in general-purpose register reg3 and the lower 32 bits are stored in general-purpose register reg2. This instruction treats contents of general-purpose register reg2 as 32-bit signed integer.

[Caution] In the “MUL reg1, reg2, reg3” instruction, do not use registers in combinations that satisfy all the following conditions. If the instruction is executed with all the following conditions satisfied, the operation is not guaranteed.

- reg1 = reg3
- reg1 ≠ reg2
- reg1 ≠ r0
- reg3 ≠ r0

[Comment]            When general-purpose register reg2 and general-purpose register reg3 are the same register, only the upper 32 bits of the multiplication result are stored in the register.

&lt;Multiply instruction&gt;

**MULH**

Multiply half-word by register/immediate (5-bit)

Multiply Half-word

[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]	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">(1)</div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center;"> <div style="margin-right: 5px;">15</div> <div style="flex-grow: 1; border-bottom: 1px solid black; text-align: center;">rrrrr000111RRRRR</div> <div style="margin-left: 5px;">0</div> </div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="margin-right: 10px;">(2)</div> <div style="border: 1px solid black; padding: 2px; display: flex; align-items: center;"> <div style="margin-right: 5px;">15</div> <div style="flex-grow: 1; border-bottom: 1px solid black; text-align: center;">rrrrr010111iiii</div> <div style="margin-left: 5px;">0</div> </div> </div>
[Flags]	CY    -- OV    -- S     -- Z     -- SAT   --
[Description]	(1) Multiplies the lower half-word data of the general register reg2 by the lower half-word data of the general register reg1 and stores the result in the general register reg2 as word data. The data of the general register reg1 is not affected. Do not specify r0 as the destination register reg2. (2) Multiplies the lower half-word data of the general register reg2 by the 5-bit immediate data, sign-extended to half-word length, and stores the result in the general register reg2. Do not specify r0 as the destination register reg2.
[Comment]	In the case of a multiplier or a multiplicand, the higher 16 bits of the general registers, reg1 and reg2, are ignored.

<Multiply instruction>

**MULHI**

Multiply half-word by immediate (16-bit)

Multiply Half-word Immediate

[Instruction format] MULHI imm16, reg1, reg2

[Operation] GR [reg2]  $\leftarrow$  GR [reg1]  $\times$  imm16

[Format] Format VI

15	0 31	16
<div style="display: flex; justify-content: space-between;"> <span>rrrrr110111RRRRR</span> <span>iiiiiiiiiiiiiiiiiii</span> </div>		

[Flags]

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

[Description] Multiplies the lower half-word data of the general register reg1 by the 16-bit immediate data and stores the result in the general register reg2. The data of the general register reg1 is not affected. Do not specify r0 as the destination register reg2.

[Comment] In the case of a multiplicand, The higher 16 bits of the general register reg1 are ignored.

## &lt;Multiplication instruction&gt;

**MULU**

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

Multiplication of (unsigned) word data

- [Instruction format] (1) MULU reg1, reg2, reg3  
 (2) MULU imm9, reg2, reg3

- [Operation] (1) GR [reg3] || GR [reg2]  $\leftarrow$  GR [reg2]  $\times$  GR [reg1]  
 (2) GR [reg3] || GR [reg2]  $\leftarrow$  GR [reg2]  $\times$  zero-extend (imm9)

- [Format] (1) Format XI  
 (2) Format XII

- [Opcode]
- |     |                                  |  |      |  |    |
|-----|----------------------------------|--|------|--|----|
|     | 15                               |  | 0 31 |  | 16 |
| (1) | rrrrr11111RRRRR wwwww01000100010 |  |      |  |    |
- 
- |     |                                 |  |      |  |    |
|-----|---------------------------------|--|------|--|----|
|     | 15                              |  | 0 31 |  | 16 |
| (2) | rrrrr11111iiii wwwww01001IIII10 |  |      |  |    |
- iiii are the lower 5 bits of 9-bit immediate data.  
 IIII are the upper 4 bits of 9-bit immediate data.

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

- [Description] (1) The word data in general-purpose register reg2 is multiplied by the word data in general-purpose register reg1, then upper 32 bits of the result (64-bit data) are stored in general-purpose register reg3 and the lower 32 bits are stored in general-purpose register reg2. This instruction treats contents of general-purpose register reg1 and reg2 as 32-bit unsigned integer. This has no effect on general-purpose register reg1.  
 (2) The word data in general-purpose register reg2 is multiplied by 9-bit immediate data with word-length code extension, then upper 32 bits of the result (64-bit data) are stored in general-purpose register reg3 and the lower 32 bits are stored in general-purpose register reg2. This instruction treats contents of general-purpose register reg2 as 32-bit unsigned integer.

- [Caution] In the “MULU reg1, reg2, reg3” instruction, do not use registers in combinations that satisfy all the following conditions. If the instruction is executed with all the following conditions satisfied, the operation is not guaranteed.

- reg1 = reg3
- reg1  $\neq$  reg2
- reg1  $\neq$  r0
- reg3  $\neq$  r0

[Comment]

When general-purpose register reg2 and general-purpose register reg3 are the same register, only the upper 32 bits of the multiplication result are stored in the register.

<Special instruction>

<b>NOP</b>	No operation
	No Operation

[Instruction format] NOP

[Operation] Executes nothing and consumes at least one clock.

[Format] Format I

[Opcode] 15 0  
0000000000000000

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

[Description] Executes nothing and consumes a minimum of one clock cycle.

[Comment] The PC contents are incremented by +2. The opcode is the same as that of MOV r0, r0.



<Logical instruction>

<b>NOT</b>	NOT
	Not

[Instruction format] NOT reg1, reg2

[Operation] GR [reg2] ← NOT (GR [reg1] )

[Format] Format I

[Opcode]

15	0
rrrrr000001RRRRR	

[Flags]

CY --

OV 0

S "1" if operation result's 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 the general register reg1 using the 1's complement and stores the result in the general register reg2. The reg1 data is not affected.

&lt;Bit manipulation instruction&gt;

<b>NOT1</b>	NOT bit
	Not Bit

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

(2) NOT1 reg2, [reg1]

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

(2)  $\text{adr} \leftarrow \text{GR}[\text{reg1}]$   
 $\text{Z flag} \leftarrow \text{Not}(\text{Load-memory-bit}(\text{adr}, \text{reg2}))$   
 $\text{Store-memory-bit}(\text{adr}, \text{reg2}, \text{Z flag})$

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

[Opcode]

	15	0	31	16
(1)	01bbb111110RRRRR ddddddddddddddd			
	15	0	31	16
(2)	rrrrr11111RRRRR 0000000011100010			

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

[Description] (1) Adds the word data of general-purpose register reg1 to a 16-bit displacement, sign-extended to word length to generate a 32-bit address. Then reads the byte data referenced by the generated address, inverts the bit specified by the 3-bit bit number ( $0 \rightarrow 1$  or  $1 \rightarrow 0$ ), and writes back to the original address.

(2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Then reads the byte data referenced by the generated address, inverts the bit specified by the data of lower 3 bits of reg2 ( $0 \rightarrow 1$  or  $1 \rightarrow 0$ ), and writes back to the original address.

[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.

<Logical instruction>

<b>OR</b>	OR
	Or

[Instruction format] OR reg1, reg2

[Operation] GR [reg2] ← GR [reg2] OR GR [reg1]

[Format] Format I

[Opcode]

15	0
rrrrr001000RRRRR	

[Flags]

CY --

OV 0

S "1" if operation result's word data MSB is "1"; otherwise, "0".

Z "1" if the operation result is "0"; otherwise, "0".

SAT --

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

<Logical instruction>

**ORI**

OR immediate (16-bit)

Or Immediate

[Instruction format] ORI imm16, reg1, reg2

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

[Format] Format VI

[Opcode]

15		0	31		16
rrrrr110100RRRRRiiiiiiiiiiiiiiiiiii					

[Flags]

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

[Description] ORs the word data of the general register reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in the general register reg2. The reg1 data is not affected.

<Special instruction>

<b>PREPARE</b>	Function prepare
	Create stack frame

- [Instruction format] (1) PREPARE list12, imm5  
 (2) PREPARE list12, imm5, sp/imm<sup>Note</sup>

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

- [Operation] (1) Store-memory (sp – 4, GR [reg in list12], Word) sp ← sp – 4  
 repeat 1 step above until all regs in list12 is stored  
 sp ← sp – zero-extend (imm5)  
 (2) Store-memory (sp – 4, GR [reg in list12], Word) sp ← sp – 4  
 repeat 1 step above until all regs in list12 is stored  
 sp ← sp – zero-extend (imm5)  
 ep ← sp/imm

[Format] Format XIII

- [Opcode]
- |     |   |      |    |                         |
|-----|---|------|----|-------------------------|
|     | 15  | 0 31 | 16 |                         |
| (1) | 0000011110iiiiL LLLLLLLLLLLL00001               |      |    |                         |
|     | 15  | 0 31 | 16 | Option (47-32 or 63-32) |
| (2) | 0000011110iiiiL LLLLLLLLLLLLff011 imm16 / imm32 |      |    |                         |
- In the case of 32-bit immediate data (imm32), bits 47 to 32 are the lower 16 bits of imm32 and bits 63 to 48 are the upper 16 bits of imm32.

- ff = 00: sp is loaded to ep  
 ff = 01: Code-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 bit21 in list12).

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

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

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

- When all of the register's non-corresponding bits are "0": 08000001H
- When all of the register's non-corresponding bits are "1": 081FFFFFFH

[Flags]

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

[Description]

- (1) The general-purpose registers specified in list12 are saved (4 is subtracted from the sp value and the data is stored in that address). Next, 2-bit logical left shifts are used to create 5-bit immediate data with zeros extended to word length, which is subtracted from the sp value.
  - (2) The general-purpose registers specified in list12 are saved (4 is subtracted from the sp value and the data is stored in that address). Next, 2-bit logical left shifts are used to create 5-bit immediate data with zeros extended to word length, which is subtracted from the sp value.
- Next, the data specified by the third operand (sp/imm) is loaded to ep.

[Comment]

list12's 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.  
Even when misaligned access is enabled, the lower two bit address specified by sp is always masked (= 0).  
If an interrupt occurs before the sp is updated, execution is stopped and the interrupt is processed using the return address as this instruction's start address, then once the interrupt processing is completed execution of this instruction is resumed. In such cases, the sp and ep values prior to execution of this instruction are retained.

[Caution]

When an interrupt occurs during execution of an instruction, the stack operation may cause execution of the instruction to be stopped after the read/write cycles and register overwrite operations have been completed.

&lt;Special instruction&gt;

**RETI**

Return from trap or interrupt

Return from Trap or Interrupt

[Instruction format] RETI

[Operation]

```

if PSW.EP = 1
then PC ← EIPC
    PSW ← EIPSW
else if PSW.NP = 1
then PC ← FEPC
    PSW ← FEPSW
else PC ← EIPC
    PSW ← EIPSW

```

[Format] Format X

[Opcode]

15	0 31	16
0000011111100000	0000000101000000	

[Flags]

CY Value read from FEPSW or EIPSW is restored.

OV Value read from FEPSW or EIPSW is restored.

S Value read from FEPSW or EIPSW is restored.

Z Value read from FEPSW or EIPSW is restored.

SAT Value read from FEPSW or EIPSW is restored.

[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 flag of PSW is "1", the return PC and PSW are read from EIPC and EIPSW, regardless of the status of the NP flag of PSW.

If the EP flag of PSW is "0" and the NP flag of PSW is "1", the return PC and PSW are read from FEPC and FEPSW.

If the EP flag of PSW is "0" and the NP flag of PSW is "0", the return PC and PSW are read from EIPC and EIPSW.

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

**[Caution]**

In order to correctly restore the PC and PSW values when using a RETI instruction to recover from non-maskable interrupt processing or software exception processing, the NP and EP flags must be set as follows before executing the RETI instruction.

- When using RETI instruction to recover from non-maskable interrupt processing: NP = 1 and EP = 0
- When using RETI instruction to recover from software exception processing: EP = 1

The LDSR instruction is used for program-based settings.

Due to the operation of the interrupt controller, interrupts cannot be received in the ID stage during the second half of this instruction.



## &lt;Data manipulation instruction&gt;

**SAR**

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

Shift Arithmetic Right

[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]	<div> <div>1503116</div> <div>(1) rrrrr11111RRRRR0000000010100000</div> </div> <div> <div>150</div> <div>(2) rrrrr010101iiii</div> </div> <div> <div>1503116</div> <div>(3) rrrrr11111RRRRRwww00010100010</div> </div>
[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 the general register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits on the general register reg1, by copying the pre-shift MSB value to the post-shift MSB. The result is written to the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions. The reg1 data is not affected. (2) Arithmetically right-shifts the word data of the general register reg2 by 'n' (0 to +31), the position specified by the 5-bit immediate data, zero-extended to word length (after the shift, the MSB prior to shift execution is copied and set as the new MSB value). The result is written to the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions. (3) For the portion of the word data of general purpose register reg2 for which the shift number is indicated by the lower 5 bits of general purpose register reg1, 0 through +31 are arithmetic right-shifted (MSB value prior to shift is copied to the post-shift MSB) and written into general purpose register reg3. In cases where the shift number is 0, general-purpose register reg3 retains the value prior to execution of instructions. General purpose registers reg1 and reg2 are not affected.

&lt;Data manipulation instruction&gt;

**SASF**

Shift and set flag condition

Shift and Set Flag Condition

[Instruction format] SASF cccc, reg2

[Operation] if conditions are satisfied  
 then GR [reg2]  $\leftarrow$  (GR [reg2] Logically shift left by 1) OR 00000001H  
 else GR [reg2]  $\leftarrow$  (GR [reg2] Logically shift left by 1) OR 00000000H

[Format] Format IX

[Opcode] 15 0 31 16  
 rrrrrr1111110cccc|0000001000000000

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

[Description] When the condition specified by condition code "cccc" is met, register reg2 is logically left-shifted by "1"; if a condition is met, the least significant bit (LSB) of the register 2 is set to "1" and if a condition is not met, the least significant bit (LSB) of the register 2 is set to "0". Please designate one of the condition codes shown in the following table as [cccc].

Condition code	Condition formula	Condition code	Condition formula
0000	OV = 1	0100	S = 1
1000	OV = 0	1100	S = 0
0001	CY = 1	0101	always (unconditional)
1001	CY = 0	1101	SAT = 1
0010	Z = 1	0110	(S xor OV) = 1
1010	Z = 0	1110	(S xor OV) = 0
0011	(CY or Z) = 1	0111	((S xor OV) or Z) = 1
1011	(CY or Z) = 0	1111	((S xor OV) or Z) = 0

[Comment] Refer to the SETF instruction.

## &lt;Saturated Operation Instructions&gt;

**SATADD**

Saturated add register/immediate (5-bit)

Saturated add register/immediate (5-bit)

[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] + sigh-extend (imm5))  
 (3) GR [reg3] ← saturated (GR [reg2] + GR [reg1] )

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

[Opcode]

(1) 

15	0
rrrrr000110RRRRR	

(2) 

15	0
rrrrr010001iiii	

(3) 

15	0	31	16
rrrrr111111RRRRR	wwwwww01110111010		

[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) The word data of general-purpose register reg1 are added to the word data of general purpose register reg2, and the result is stored in general-purpose register reg2. However, in cases where the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg2, and in cases where it exceeds the maximum negative value 80000000H, 80000000H is stored in reg2; then the SAT flag is set at (1). General-purpose register reg1 is not affected.  
 Please do not designate r0 in reg2.

(2) The 5-bit immediate code-extended to the word length is added to the word data of general purpose register reg2, and the result is stored in general purpose register reg2. However, in cases where the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg2, and in cases where it exceeds the maximum negative value 80000000H, 80000000H is stored in reg2; then the SAT flag is set at (1).  
 Please do not designate r0 in reg2.

- (3) The word data of general-purpose register reg1 are added to the word data of general purpose register reg2, and the result is stored in general-purpose register reg3. However, in cases where the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg3, and in cases where it exceeds the maximum negative value 80000000H, 80000000H is stored in reg3; then the SAT flag is set at (1). General-purpose register 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 saturate instruction is executed normally, even with the SAT flag set to "1".

[Caution]           To clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

&lt;Saturate instruction&gt;

<b>SATSUB</b>	Saturated subtract
	Saturate Subtract

[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	

(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 the general register reg1 from the word data of the general register reg2 and stores the result in the general register reg2. If the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored to reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored to reg2. The SAT flag is set to "1". The reg1 data is not affected. Do not specify r0 as the destination register reg2.

(2) The word data of general-purpose register reg1 are subtracted to the word data of general-purpose register reg2, and the result is stored in general purpose register reg3. However, in cases where the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg3, and in cases where it exceeds the maximum negative value 80000000H, 80000000H is stored in reg3; then the SAT flag is set at (1). General-purpose register 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 saturate instruction is executed normally, even with the SAT flag set to "1".

[Caution] To clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

&lt;Saturate instruction&gt;

**SATSUBI**

Saturated subtract immediate

Saturate Subtract Immediate

[Instruction format] SATSUBI imm16, reg1, reg2

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

[Format] Format VI

15	0 31	16
rrrrr110011RRRRiiiiiiiiiiiiiiiiii		

[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 the general register reg1 and stores the result in the general register reg2. If the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored to reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored to reg2. The SAT flag is set to "1". The reg1 data is not affected. Do not specify r0 as the destination register reg2.

[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 saturate instruction is executed normally, even with the SAT flag set to "1".

[Caution] To clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

&lt;Saturate instruction&gt;

**SATSUBR**

Saturated subtract reverse

Saturate Subtract Reverse

[Instruction format] SATSUBR reg1, reg2

[Operation] GR [reg2] ← saturated (GR [reg1] – GR [reg2] )

[Format] Format I

[Opcode]                    15                    0

rrrrr000100RRRRR

[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 the general register reg2 from the word data of the general register reg1 and stores the result in the general register reg2. If the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored to reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored to reg2. The SAT flag is set to "1". The reg1 data is not affected. Do not specify r0 as the destination register reg2.

[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 saturate instruction is executed normally, even with the SAT flag set to "1".

[Caution]                   To clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

## &lt;Conditional Operation Instructions&gt;

**SBF**

Subtract on condition flag

Subtract on condition flag

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

[Operation] if conditions are satisfied  
 then GR [reg3]  $\leftarrow$  GR [reg2] – GR [reg1] – 1  
 else GR [reg3]  $\leftarrow$  GR [reg2] – GR [reg1] – 0

[Format] Format XI

[Opcode] 15 0 31 16  
 rrrrrr111111RRRRR|www011100cccc0

[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] In cases where the result of subtracting the content of general purpose register reg1 from the word data of general purpose register reg2 satisfies the condition designated in condition code [cccc], 1 is subtracted, and in cases where it does not, 0 is subtracted; the result is then stored in general purpose register reg3. General purpose register reg2 is not affected. Please designate one of the condition codes shown in the following table as [cccc]. (However, cccc cannot equal 1101.)

Condition Code	Condition formula	Condition Code	Condition formula
0000	OV = 1	0100	S = 1
1000	OV = 0	1100	S = 0
0001	CY = 1	0101	always (Unconditional)
1001	CY = 0	0110	(S xor OV) = 1
0010	Z = 1	1110	(S xor OV) = 0
1010	Z = 0	0111	( (S xor OV) or Z ) = 1
0011	(CY or Z) = 1	1111	( (S xor OV) or Z ) = 0
1011	(CY or Z) = 0	(1101)	Assignment Inhibited



## &lt;Bit Search Instructions&gt;

**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

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] Word data of general purpose register reg2 are searched from the left side (MSB side), and the position at which bit (0) is first found is written into general purpose register reg3 in base 16 (e.g., in cases where bit 31 of reg2 is 0, 01H is written into reg3). In cases where bit (0) is not found, 0 is written into reg3, and the Z-flag is simultaneously set at (1). In cases where bit (0) is eventually found, the CY flag is set at (1).

## &lt;Bit Search Instructions&gt;

**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

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] Word data of general purpose register reg2 are searched from the right side (LSB side), and the position at which bit (0) is first found is written into general purpose register reg3 in base 16 (e.g., in cases where bit 0 of reg2 is 0, 01H is written into reg3). In cases where bit (0) is not found, 0 is written into reg3, and the Z-flag is simultaneously set at (1). In cases where bit (0) is eventually found, the CY flag is set at (1).

<Bit Search Instructions>

<b>SCH1L</b>	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
rrrrr	11111100000	wwwwww01101100110

[Flags]

CY	"1" if bit (1) is found eventually; otherwise, "0".
OV	0
S	0
Z	"1" if bit (1) is not found; otherwise, "0".
SAT	--

[Description] Word data of general purpose register reg2 are searched from the left side (MSB side), and the position at which bit (1) is first found is written into general purpose register reg3 in base 16 (e.g., in cases where bit 31 of reg2 is 1, 01H is written into reg3). In cases where bit (1) is not found, 0 is written into reg3, and the Z-flag is simultaneously set at (1). In cases where bit (1) is eventually found, the CY flag is set at (1).

## &lt;Bit Search Instructions&gt;

**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

15	0 31	16
rrrrr11111100000 www01101100010		

[Flags]

CY    "1" if bit (1) is found eventually; otherwise, "0".

OV    0

S     0

Z     "1" if bit (1) is not found; otherwise, "0".

SAT   --

[Description] Word data of general purpose register reg2 are searched from the right side (LSB side), and the position at which bit (1) is first found is written into general purpose register reg3 in base 16 (e.g., in cases where bit 0 of reg2 is 1, 01H is written into reg3). In cases where bit (1) is not found, 0 is written into reg3, and the Z-flag is simultaneously set at (1). In cases where bit (1) is eventually found, the CY flag is set at (1).

&lt;Bit manipulation instruction&gt;

<b>SET1</b>	Set bit
	Set Bit

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

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

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

[Opcode]

	15	0	31	16
(1)	00bbb111110RRRRR dddddddddddddddd			
	15	0	31	16
(2)	rrrrr11111RRRRR 0000000011100000			

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

[Description] (1) Adds the 16-bit displacement, sign-extended to word length, to the word data of general-purpose register reg1 to generate a 32-bit address. Then reads the byte data referenced by the generated address, sets the bit specified by the 3-bit bit number to 1, and writes back to the original address.  
 (2) Reads the word data of general-purpose register reg1 to generate a 32-bit address. Then reads the byte data referenced by the generated address, sets the bit specified by the data of lower 3 bits of reg2 to 1, and writes back to the original address.

[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.

<Data manipulation instruction>

# SETF

Set flag condition

Set Flag Condition

[Instruction format] SETF cccc, reg2

[Operation] if conditions are satisfied  
then GR [reg2]  $\leftarrow$  00000001H  
else GR [reg2]  $\leftarrow$  00000000H

[Format] Format IX

[Opcode] 15 0 31 16  
rrrrrr1111110cccc0000000000000000

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

[Description] When the condition specified by condition code "cccc" is met, Sets "1" to the general register reg2 if a condition is met and sets "0" if a condition is not met.  
Please designate one of the condition codes shown in the following table as [cccc].

Condition code	Condition formula	Condition code	Condition formula
0000	OV = 1	0100	S = 1
1000	OV = 0	1100	S = 0
0001	CY = 1	0101	Always (unconditional)
1001	CY = 0	1101	SAT = 1
0010	Z = 1	0110	(S xor OV) = 1
1010	Z = 0	1110	(S xor OV) = 0
0011	(CY or Z) = 1	0111	( (S xor OV) or Z) = 1
1011	(CY or Z) = 0	1111	( (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 more 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 to register  $R_a$ . By performing a logical operation to  $R_{a_n}$  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 to the general register  $reg2$ . Therefore, a carry from the lower bits can be expressed as a numeric value.

&lt;Data manipulation instruction&gt;

**SHL**

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

Shift Logical Left

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

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

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

[Opcode]

15	0	31	16
(1)	rrrrr11111RRRRR0000000011000000		
15	0		
(2)	rrrrr010110iiii		
15	0	31	16
(3)	rrrrr11111RRRRRwww00011000010		

[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 the general register reg2 by 'n'(0 to +31), the position specified by the lower 5 bits on the general register reg1, by shifting "0" to LSB. The result is written to the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions. The data of the general register reg1 is not affected.  
 (2) Logically left-shifts the word data of the general 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 the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions.  
 (3) For the portion of the word data of general purpose register reg2 for which the shift number is indicated by the lower 5 bits of general-purpose register reg1, 0 through +31 are logic left-shifted (0 is fed into the LSB side) and written into general purpose register reg3. In cases where the shift number is 0, general-purpose register reg3 retains the value prior to execution of instructions. General purpose registers reg1 and reg2 are not affected.



## &lt;Data manipulation instruction&gt;

**SHR**

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

Shift Logical Right

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

[Operation] (1) GR [reg2]  $\leftarrow$  GR [reg2] logically shift right by GR [reg1]  
 (2) GR [reg2]  $\leftarrow$  GR [reg2] logically shift right by zero-extend(imm5)  
 (3) GR [reg3]  $\leftarrow$  GR [reg2] logically shift right by GR [reg1]

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

[Opcode]

	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 the general register reg2 by 'n' (0 to +31), the position specified by the lower 5 bits on the general register reg1, by shifting "0" to MSB. The result is written to the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions. The data of the general register reg1 is not affected.  
 (2) Logically right-shifts the word data of the general 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 the general register reg2. In cases where the shift number is 0, general-purpose register reg2 retains the value prior to execution of instructions.  
 (3) For the portion of the word data of general purpose register reg2 for which the shift number is indicated by the lower 5 bits of general-purpose register reg1, 0 through +31 are logic right-shifted (0 is fed into the MSB side) and written into general purpose register reg3. In cases where the shift number is 0, general-purpose register reg3 retains the value prior to execution of instructions. General purpose registers reg1 and reg2 are not affected.

&lt;Load instruction&gt;

**SLD.B**

Short format load byte

Short format load byte

[Instruction format]	SLD.B disp7 [ep] , reg2
[Operation]	$\text{adr} \leftarrow \text{ep} + \text{zero-extend}(\text{disp7})$ $\text{GR}[\text{reg2}] \leftarrow \text{sign-extend}(\text{Load-memory}(\text{adr}, \text{Byte}))$
[Format]	Format IV
[Opcode]	<div style="display: flex; justify-content: space-between; align-items: center;"> <span>15</span> <span>0</span> </div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">rrrrrr0110dddddd</div>
[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 to reg2.
[Comment]	<p>When an interrupt occurs during the SLD.B instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt.</p> <p>The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.</p>

&lt;Load instruction&gt;

**SLD.BU**

Short format load byte unsigned

Short format load byte unsigned

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

[Operation]  $\text{adr} \leftarrow \text{ep} + \text{zero-extend}(\text{disp4})$   
 $\text{GR}[\text{reg2}] \leftarrow \text{zero-extend}(\text{Load-memory}(\text{adr}, \text{Byte}))$

[Format] Format IV

[Opcode] 15                      0  
rrrrrr0000110dddd  
 rrrrrr must be other than 00000.

[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, sign-extended to word length, and stored to reg2.

[Comment] When an interrupt occurs during the SLD.BU instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

&lt;Load instruction&gt;

**SLD.H**

Short format load half-word

Short format load half-word

[Instruction format]	SLD.H disp8 [ep] , reg2
[Operation]	$adr \leftarrow ep + \text{zero-extend}(\text{disp8})$ $GR[\text{reg2}] \leftarrow \text{sign-extend}(\text{Load-memory}(adr, \text{Half-word}))$
[Format]	Format IV
[Opcode]	<div> <div>15</div> <div>0</div> <div>rrrrrr1000dddddd</div> </div> <p>ddddddd is the higher 7 bits of disp8.</p>
[Flags]	CY    -- OV    -- S     -- Z     -- SAT   --
[Description]	Adds the element pointer to the 8-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, sign-extended to word length, and stored to the general register reg2.
[Comment]	<p>When an interrupt occurs during the SLD.H instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt.</p> <p>The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.</p>
[Caution]	<p>Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>• Bit0 is masked to "0" and address is generated (when misaligned access is disabled).</li> <li>• Bit0 is not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see <b>3.3 Data Alignment</b>.</p>

&lt;Load instruction&gt;

**SLD.HU**

Short format load half-word unsigned

Short format load half-word unsigned

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

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

[Format] Format IV

[Opcode] 15                      0  
rrrrr0000111dddd  
 dddd is the higher 4 bits of disp5. rrrrr must be other than 00000.

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

[Description] Adds the element pointer to the 5-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, sign-extended to word length, and stored to the general register reg2.

[Comment] When an interrupt occurs during the SLD.HU instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

[Caution] Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 is masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 is not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

SLD.W	Short format load word
	Short format load word

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

[Opcode]                      15                      0

rrrrrr1010dddddd0

dddddd is the higher 6 bits of disp8.

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

[Comment] When an interrupt occurs during the SLD.W instruction execution, the execution is aborted to process the interrupt. The execution resumes at the original instruction address upon returning from the interrupt.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

&lt;Store instruction&gt;

**SST.B**

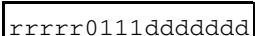
Short format store byte

Short format store byte

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

[Operation]  $\text{adr} \leftarrow \text{ep} + \text{zero-extend}(\text{disp7})$   
 Store-memory (adr, GR [reg2] , Byte)

[Format] Format IV

[Opcode] 15 0  


[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 and stores the data on the lowest byte of reg2 to the generated address.

[Comment] If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

&lt;Store instruction&gt;

**SST.H**

Short format store half-word

Short format store half-word

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

[Operation]  $\text{adr} \leftarrow \text{ep} + \text{zero-extend}(\text{disp8})$   
 Store-memory (adr, GR [reg2], Half-word)

[Format] Format IV

[Opcode] 150  
rrrrrr1001dddddd  
 ddddddd is the higher 7 bits of disp8.

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

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

[Caution] Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 is masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 is not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.



&lt;Store instruction&gt;

**SST.W**

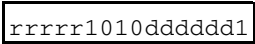
Short format store word

Short format store word

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

[Operation]  $\text{adr} \leftarrow \text{ep} + \text{zero-extend}(\text{disp8})$   
 Store-memory (adr, GR [reg2] , Word)

[Format] Format IV

[Opcode] 15 0  
  
 ddddd is the higher 6 bits of disp8.

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

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

[Caution] Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

&lt;Store instruction&gt;

<b>ST.B</b>	Store byte
	Store byte

[Instruction format] ST.B reg2, disp16 [reg1]

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

[Format] Format VII

[Opcode]           15                           0 31                           16  
rrrrrr111010RRRRRdddddddddddddddd

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

[Description]    Adds the data of the general 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 the general register reg2 to the generated address.

[Comment]        If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

&lt;Store instruction&gt;

<b>ST.H</b>	Store half-word
	Store half-word

[Instruction format] ST.H reg2, disp16 [reg1]

[Operation]  $\text{adr} \leftarrow \text{GR}[\text{reg1}] + \text{sign-extend}(\text{disp16})$   
 Store-memory (adr, GR [reg2], Half-word)

[Format] Format VII

[Opcode]

15	0 31	16
rrrrr111011RRRRR	d	ddddd0

ddddd is the higher 15 bits of disp16.

[Flags]

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

[Description] Adds the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address and stores the lower half-word data of the general register reg2 to the generated 32-bit address.

[Caution] Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 is masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 is not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

&lt;Store instruction&gt;

<b>ST.W</b>	Store word
	Store word

[Instruction format] ST.W reg2, disp16 [reg1]

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

[Format] Format VII

[Opcode] 

15	0 31	16
rrrrr111011RRRRR	ddddd	1

  
 ddddd is the higher 15 bits of disp16.

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

[Description] Adds the data of the general 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 the general register reg2 to the generated 32-bit address

[Caution] Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:

- Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).
- Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).

For details on misaligned access, see **3.3 Data Alignment**.

[Comment] If an interrupt occurs during instruction execution, execution is aborted, and the interrupt is serviced. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction.

The bus cycle sequence may be changed if accessing the resources connected to a bus, such as VFB, VDB, VSB, NPB, instruction cache bus, or data cache bus. The bus sequence change will not occur for an access to the same bus.

<Special instruction>

# STSR

Store contents of system register

Store Contents of System Register

[Instruction format] STSR regID, reg2

[Operation] GR [reg2] ← SR [regID]

[Format] Format IX

[Opcode]

15	0	31	16
rrrrr11111RRRRR0000000001000000			

[Flags]

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

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

[Caution] The system register number regID is to identify a system register. Accessing reserved system registers is prohibited.

&lt;Arithmetic instruction&gt;

<b>SUB</b>	Subtract  Subtract
------------	--------------------------

[Instruction format] SUB reg1, reg2

[Operation] GR [reg2] ← GR [reg2] – GR [reg1]

[Format] Format I

[Opcode] 15 0

rrrrr001101RRRR
-----------------

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

[Description] Subtracts the word data of the general register reg1 from the word data of the general register reg2 and stores the result in the general register reg2. The reg1 data is not affected.

<Arithmetic instruction>

<b>SUBR</b>	Subtract reverse
	Subtract Reverse

[Instruction format] SUBR reg1, reg2

[Operation] GR [reg2] ← GR [reg1] – GR [reg2]

[Format] Format I

[Opcode]

15	0
rrrrr	001100RRRRR

[Flags]

CY "1" if a borrow occurs from MSB; otherwise, "0".

OV "1" if overflow occurs; otherwise, "0".

S "1" if the operation result is negative; otherwise, "0".

Z "1" if the operation result is "0"; otherwise, "0".

SAT --

[Description] Subtracts the word data of the general register reg2 from the word data of general register reg1 and stores the result in the general register reg2. The reg1 data is not affected.

&lt;Special instruction&gt;

**SWITCH**

Jump with table look up

Jump with Table Look Up

[Instruction format] SWITCH reg1

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

[Format] Format I

[Opcode] 15 0  
 00000000010RRRRR

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

[Description] The following steps are taken.

- (1) Adds the entry address (the one subsequent to the SWITCH instruction) to the data of the general register reg1, logically left-shifted by 1, to generate a 32-bit table entry address.
- (2) Loads the half-word entry data indicated by the address generated in step (1) and sign-extends to word length.
- (3) Adds the table entry address after 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).



<b>SXB</b>	Sign extend byte
	Sign-Extend Byte

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

<b>SXH</b>	Sign extend half-word
	Sign-Extend Half-word

&lt;Special instruction&gt;

<b>TRAP</b>	Trap
	Trap

[Instruction format] TRAP vector

[Operation]

$EIPC \leftarrow PC + 4$  (return PC)  
 $EIPSW \leftarrow PSW$   
 $ECR.EICC \leftarrow$  exception code (40H-4FH, 50H-5FH)  
 $PSW.EP \leftarrow 1$   
 $PSW.ID \leftarrow 1$   
 $PC \leftarrow$  00000040H (vector 00H-0FH (exception code: 40H-4FH))  
           00000050H (vector 10H-1FH (exception code: 50H-5FH))

[Format] Format X

[Opcode]

15	0	31	16
<div style="border: 1px solid black; padding: 2px; display: inline-block;"> 000001111111iiii </div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 10px;"> 0000000100000000 </div>			

[Flags]

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

[Description]

The following steps are taken:

- (1) Saves the return PC and PSW to EIPC and EIPSW, respectively.
- (2) Sets the exception code (EICC of ECR).
- (3) Sets the PSW flags by assigning "1" to EP flag and ID flag.
- (4) Jumps to the handler address corresponding to the trap vector (00H to 1FH) specified by the vector to start exception processing.

PSW flags, other than EP and ID, are not affected. The return PC is the address of the instruction subsequent to the TRAP instruction.

&lt;Logical instruction&gt;

**TST**

Test

Test

[Instruction format] TST reg1, reg2

[Operation] result ← GR [reg2] AND GR [reg1]

[Format] Format I

[Opcode] 15 0

rrrrr001011RRRR
-----------------

[Flags]

CY --

OV 0

S "1" if operation result's word data MSB is "1"; otherwise, "0".

Z "1" if the operation result is "0"; otherwise, 0.

SAT --

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

&lt;Bit manipulation instruction&gt;

<b>TST1</b>	Test bit
	Test Bit

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

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

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

[Opcode]

	15	0 31	16
(1)	11bbb111110RRRRR dddddddddddddddd		
	15	0 31	16
(2)	rrrrr111111RRRRR 0000000011100110		

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

[Description] (1) Adds the data of the general 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 data of the general 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.

<b>XOR</b>	Exclusive OR
	Exclusive Or

[Instruction format] XOR reg1, reg2

[Operation]             $GR[reg2] \leftarrow GR[reg2] \text{ XOR } GR[reg1]$

[Format]	Format I
----------	----------

[Opcode]                      15                      0  
rrrrrr001001RRRRR

[Flags]	CY	--
	OV	0
	S	"1" if operation result's 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 the general register reg2 with the word data of the general register reg1 and stores the result in the general register reg2. The reg1 data is not affected.
---------------	---

<Logical instruction>

**XORI**

Exclusive OR immediate (16-bit)

Exclusive Or Immediate

[Instruction format] XORI imm16, reg1, reg2

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

[Format] Format VI

[Opcode]

15	0	31	16
rrrrr110101RRRRRiiiiiiiiiiiiiiiiii			

[Flags]

CY --

OV 0

S "1" if operation result's 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 the general register reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in the general register reg2. The reg1 data is not affected.

<Data manipulation instruction>

<b>ZXB</b>	Zero extend byte
	Zero-Extend Byte

[Instruction format] ZXB reg1

[Operation] GR [reg1] ← zero-extend (GR [reg1] (7:0) )

[Format] Format I

[Opcode]

15	0
00000000100RRRR	

[Flags]

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

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



ZXH	Zero extend half-word
	Zero-Extend Half-word

[Description]	Zero-extends the lower half-word on the general register reg1 to word length.
---------------	---

## 5. 4 Number of Instruction Execution Clock Cycles

Table 5-6 Shows clock requirements for each instruction execution. Combination of instructions may change the number of clock cycles required. For details, refer to **CHAPTER 8 PIPELINE OPERATIONS**.

**Table 5-6. Clock Requirements (1 of 4)**

Instruction type	Mnemonics	Operand	Byte	Number of clocks required			parallel issuance <sup>Note1</sup>
				issue	repeat	latency	
Load instruction	LD.B	disp16 [reg1] , reg2	4	1	1	Note 2	Yes(L)
	LD.H	disp16 [reg1] , reg2	4	1	1	Note 2	Yes(L)
	LD.W	disp16 [reg1] , reg2	4	1	1	Note 2	Yes(L)
	LD.BU	disp16 [reg1] , reg2	4	1	1	Note 2	Yes(L)
	LD.HU	disp16 [reg1] , reg2	4	1	1	Note 2	Yes(L)
	SLD.B	disp7 [ep] , reg2	2	1	1	Note 2	Yes(L)
	SLD.BU	disp4 [ep] , reg2	2	1	1	Note 2	Yes(L)
	SLD.H	disp8 [ep] , reg2	2	1	1	Note 2	Yes(L)
	SLD.HU	disp5 [ep] , reg2	2	1	1	Note 2	Yes(L)
	SLD.W	disp8 [ep] , reg2	2	1	1	Note 2	Yes(L)
Store instruction	ST.B	reg2, disp16 [reg1]	4	1	1	1	Yes(L)
	ST.H	reg2, disp16 [reg1]	4	1	1	1	Yes(L)
	ST.W	reg2, disp16 [reg1]	4	1	1	1	Yes(L)
	SST.B	reg2, disp7 [ep]	2	1	1	1	Yes(L)
	SST.H	reg2, disp8 [ep]	2	1	1	1	Yes(L)
	SST.W	reg2, disp8 [ep]	2	1	1	1	Yes(L)
Multiply instruction	MUL	reg1, reg2, reg3	4	1	1	3	Yes(L)
	MUL	imm9, reg2, reg3	4	1	1	3	Yes(L)
	MULH	reg1, reg2	2	1	1	3	Yes(L)
	MULH	imm5, reg2	2	1	1	3	Yes(L)
	MULHI	imm16, reg1, reg2	4	1	1	3	Yes(L)
	MULU	reg1, reg2, reg3	4	1	1	3	Yes(L)
	MULU	imm9, reg2, reg3	4	1	1	3	Yes(L)
Multiplication with addition instruction	MAC	reg1, reg2, reg3, reg4	4	1	1	3	No
	MACU	reg1, reg2, reg3, reg4	4	1	1	3	No
Arithmetic instruction	ADD	reg1, reg2	2	1	1	1	Yes(R/L)
	ADD	imm5, reg2	2	1	1	1	Yes(R/L)
	ADDI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	CMP	reg1, reg2	2	1	1	1	Yes(R/L)
	CMP	imm5, reg2	2	1	1	1	Yes(R/L)

Table 5-6. Clock Requirements (2 of 4)

Instruction type	Mnemonics	Operand	Byte	Number of clocks required			parallel issuance <sup>Note1</sup>
				issue	repeat	latency	
Arithmetic instruction	MOV	reg1, reg2	2	1	1	1	Yes(R/L)
	MOV	imm5, reg2	2	1	1	1	Yes(R/L)
	MOV	imm32, reg1	6	1	1	1	No
	MOVEA	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	MOVHI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	SUB	reg1, reg2	2	1	1	1	Yes(R/L)
	SUBR	reg1, reg2	2	1	1	1	Yes(R/L)
Conditional Operation Instructions	ADF	cccc, reg1, reg2, reg3	4	1	1	1	No
	SBF	cccc, reg1, reg2, reg3	4	1	1	1	No
Saturate instruction	SATADD	reg1, reg2	2	1	1	1	Yes(R/L)
	SATADD	imm5, reg2	2	1	1	1	Yes(R/L)
	SATADD	reg1, reg2, reg3	4	1	1	1	No
	SATSUB	reg1, reg2	2	1	1	1	Yes(R/L)
	SATSUB	reg1, reg2, reg3	4	1	1	1	No
	SATSUBI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	SATSUBR	reg1, reg2	2	1	1	1	Yes(R/L)
Logical instruction	AND	reg1, reg2	2	1	1	1	Yes(R/L)
	ANDI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	NOT	reg1, reg2	2	1	1	1	Yes(R/L)
	OR	reg1, reg2	2	1	1	1	Yes(R/L)
	ORI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
	TST	reg1, reg2	2	1	1	1	Yes(R/L)
	XOR	reg1, reg2	2	1	1	1	Yes(R/L)
	XORI	imm16, reg1, reg2	4	1	1	1	Yes(R/L)
Data manipulation instruction	BSH	reg2, reg3	4	1	1	1	Yes(R)
	BSW	reg2, reg3	4	1	1	1	Yes(R)
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1	Yes(R)
	CMOV	cccc, imm5, reg2, reg3	4	1	1	1	Yes(R)
	HSH	reg2, reg3	4	1	1	1	Yes(R)
	HSW	reg2, reg3	4	1	1	1	Yes(R)
	SAR	reg1, reg2	4	1	1	1	Yes(R)
	SAR	imm5, reg2	2	1	1	1	Yes(R)
	SAR	reg1, reg2, reg3	4	1	1	1	Yes(R)
	SASF	cccc, reg2	4	1	1	1	Yes(R)

Table 5-6. Clock Requirements (3 of 4)

Instruction type	Mnemonics	Operand	Byte	Number of clocks required			parallel issuance <sup>Note1</sup>
				issue	repeat	latency	
Data manipulation instruction	SETF	cccc, reg2	4	1	1	1	Yes(R)
	SHL	reg1, reg2	4	1	1	1	Yes(R)
	SHL	imm5, reg2	2	1	1	1	Yes(R)
	SHL	reg1, reg2, reg3	4	1	1	1	Yes(R)
	SHR	reg1, reg2	4	1	1	1	Yes(R)
	SHR	imm5, reg2	2	1	1	1	Yes(R)
	SHR	reg1, reg2, reg3	4	1	1	1	Yes(R)
	SXB	reg1	2	1	1	1	No
	SXH	reg1	2	1	1	1	No
	ZXB	reg1	2	1	1	1	No
	ZXH	reg1	2	1	1	1	No
Bit Search Instructions	SCH0L	reg2, reg3	4	1	1	1	Yes(R)
	SCH0R	reg2, reg3	4	1	1	1	Yes(R)
	SCH1L	reg2, reg3	4	1	1	1	Yes(R)
	SCH1R	reg2, reg3	4	1	1	1	Yes(R)
Divide instruction	DIV	reg1, reg2, reg3	4	35	35	35	No
	DIVH	reg1, reg2	2	35	35	35	No
	DIVH	reg1, reg2, reg3	4	35	35	35	No
	DIVHU	reg1, reg2, reg3	4	34	34	34	No
	DIVU	reg1, reg2, reg3	4	34	34	34	No
Branch instruction	Bcond	disp9 (When condition is satisfied)	2	4 <sup>Note3</sup>	4 <sup>Note3</sup>	4 <sup>Note3</sup>	Yes(B+R/L)
		disp9 (When condition is not satisfied)	2	1	1	1	Yes(B+R/L)
	JARL	disp22, reg2	4	4	4	4	Yes(B+R/L)
	JARL	disp32, reg1	6	4	4	4	No
	JMP	[reg1]	2	5	5	5	Yes(B+R/L)
	JMP	disp32 [reg1]	6	5	5	5	No
	JR	disp22	4	4	4	4	Yes(B+R/L)
	JR	disp32	6	4	4	4	No
Bit manipulation instruction	CLR1	bit#3, disp16 [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	CLR1	reg2, [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	NOT1	bit#3, disp16 [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	NOT1	reg2, [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	SET1	bit#3, disp16 [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No

Table 5-6. Clock Requirements (4 of 4)

Instruction type	Mnemonics	Operand	Byte	Number of clocks required			parallel issuance <sup>Note1</sup>
				issue	repeat	latency	
Bit manipulation instruction	SET1	reg2, [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	TST1	bit#3, disp16 [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
	TST1	reg2, [reg1]	4	4 <sup>Note4</sup>	4 <sup>Note4</sup>	4 <sup>Note4</sup>	No
Special instruction	CALLT	imm6	2	8	8	8	No
	CTRET	--	4	9	9	9	No
	DI	--	4	2	2	2	No
	DISPOSE	imm5, list12	4	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	DISPOSE	imm5, list12, [reg1]	4	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	EI	--	4	2	2	2	No
	HALT	--	4	1	1	1	No
	LDSR	reg2, regID	4	2	2	2	No
	NOP	--	2	1	1	1	Yes(R/L)
	PREPARE	list12, imm5	4	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	PREPARE	list12, imm5, sp	4	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	PREPARE	list12, imm5, imm16	6	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	PREPARE	list12, imm5, imm32	8	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	n+1 <sup>Note5</sup>	No
	RETI	--	4	undefined	undefined	undefined	No
	STSR	regID, reg2	4	1	1	1	No
	SWITCH	reg1	2	8	8	8	No
	TRAP	vector	4	9	9	9	No
Debug function instruction	DBRET	--	4	undefined	undefined	undefined	No
	DBTRAP	--	2	undefined	undefined	undefined	No
Undefined instruction code			4	4	4	4	--

**Notes1.** "Yes" indicates when an instruction can be issued at the same time as another instruction, and "No" indicates when such parallel issuance of instructions is not possible (instructions must be issued one at a time). The pipeline that is used for parallel issuance is shown in parentheses (L: Lpipe, R: Rpipe, B: Bpipe, and R/L: Rpipe or Lpipe). For details, see **CHAPTER 8 PIPELINE OPERATIONS**.

2. According to the number of wait states (if there are no wait states, then value is 3).
3. Value is 6 if previous instruction overwrites PSW.
4. When there are no wait states (4 + number of read access wait states).
5. n is the total number of registers specified by list12 (This depends on the number of wait states (if there are no wait states, then n is the total number of registers specified by list12. This is the same whether n = 0 or n = 1)).

**Remarks1.** Operand convention

Symbol	Meaning
reg1	General register (as source register)
reg2	General register (primarily as destination register with some as source registers)
reg3	General register (primarily used to store the remainder of a division result and/or the higher 32 bits of a multiply result)
bit#3	3-bit data to specify bit number
imm ×	×-bit immediate data
disp ×	×-bit displacement data
regID	System register number
vector	5-bit data to specify trap vector (00H to1FH)
cccc	4-bit data to specify condition code
sp	Stack pointer (r3)
ep	Element pointer (r30)
list12	Lists of registers

**2.** Execution clock convention

Symbol	Meaning
issue	When other instruction is executed immediately after executing an instruction
repeat	When the same instruction is repeatedly executed immediately after the instruction has been executed
latency	When a subsequent instruction uses the result of execution of the preceding instruction immediately after its execution

## CHAPTER 6 INTERRUPTS AND EXCEPTIONS

Interrupts are events that occur independently of the program execution and are divided into two types: maskable interrupts and non-maskable interrupts (NMI). Exceptions are events that are program-execution dependent and are divided into four types: software exception, exception trap, debug trap, and debug break. When either an interrupt or an exception occurs, controls are transferred to a handler whose address is determined by the interrupt/exception source. The interrupt/exception source is specified by the exception code stored in the exception cause register (ECR). The handler analyzes the ECR register for appropriate interrupt servicing or exception processing. The return PC and restore PSW are subsequently written to the status-saving registers. Two instructions, RETI and DBRET, are used. RETI instruction enables a return from interrupt or software exception processing. DBRET instruction enables a return from exception trap, debug trap or debug break. The return PC and PSW can be read from the status-saving registers, and controls are subsequently transferred to the return PC.

**Table 6-1. Interrupt/Exception Codes**

Interrupt/Exception Source			Classification	Exception Code	Handler Address	Return PC
Name		Triggered by				
Non-maskable interrupt (NMI)		NMI0 input	Interrupt	0010H	00000010H	next PC <sup>Note 1</sup>
		NMI1 input	Interrupt	0020H	00000020H	next PC <sup>Note 1,2</sup>
		NMI2 input <sup>Note 3</sup>	Interrupt	0030H	00000030H	next PC <sup>Note 1,2</sup>
Maskable interrupt		<b>Note 4</b>	Interrupt	<b>Note 4</b>	<b>Note 5</b>	next PC <sup>Note 2</sup>
Software exception	TRAP0n(n = 0-FH)	TRAP instruction	Exception	004nH	00000040H	next PC
	TRAP1n(n = 0-FH)	TRAP instruction	Exception	005nH	00000050H	next PC
Exception trap (ILGOP)		Illegal instruction code	Exception	0060H	00000060H	next PC <sup>Note6</sup>
Debug trap		DBTRAP instruction	Exception	0060H	00000060H	next PC
Debug break		Debug break	Exception	0060H	00000060H	next PC

**Notes 1.** Except when an interrupt is acknowledged during execution of the one of the instructions listed below (if an interrupt is acknowledged during instruction execution, execution is stopped, and then resumed after the completion of interrupt servicing).

- Divide instructions (DIV, DIVH, DIVU, DIVHU)
- PREPARE and DISPOSE instructions (if an interrupt is generated before the stack pointer updates).

**2.** RETI instruction cannot restore the PC content. A system reset is required after interrupt servicing.

**3.** Non-maskable interrupt can be acknowledged even if the NP flag of PSW is set to "1."

**4.** Varies based on the type of interrupts.

**5.** The upper 16bits are 0000H. For the lower 16 bits, see the Exception Code column.

**6.** The execution address of the illegal instruction is obtained by "Return PC – 4".

**Remark** Return PC indicates PC value saved to EIPC or FEPC at the outset of interrupt/exception processing.  
Next PC indicates PC value to initiate process after interrupt/exception processing.

## 6. 1 Interrupt Servicing

### 6. 1. 1 Maskable interrupt

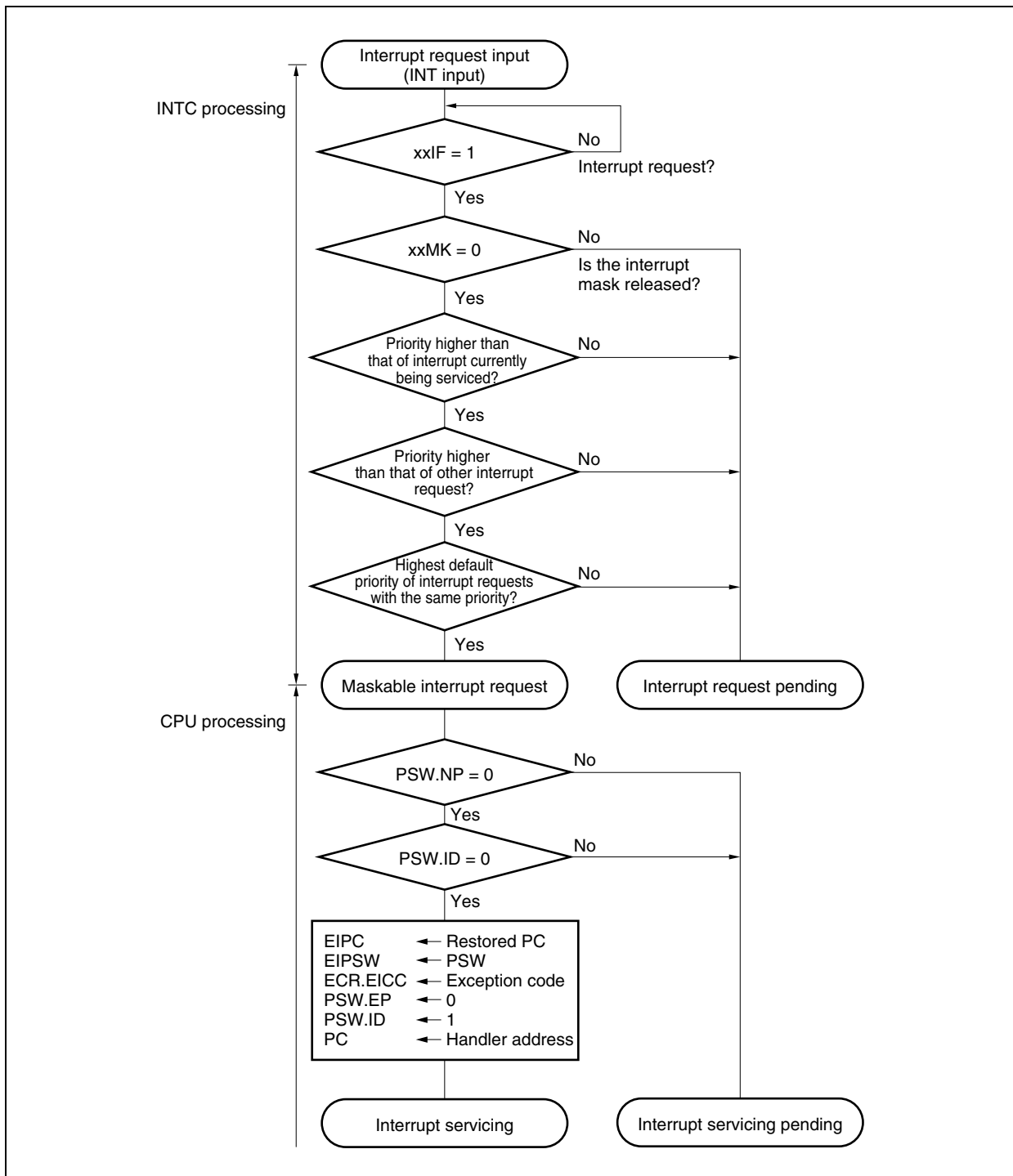
The INTC (interrupt controller) register can be masked by interrupt controller registers (PICS). The INTC register issues an interrupt request to CPU in the order of highest priority interrupt. The interrupt request input (INT input) enables CPU to perform the following steps, followed by a control transfer to the handler routine:

- (1) Saves the return PC to EIPC.
- (2) Saves the current PSW to EIPSW.
- (3) Writes the exception code to the lower half-word of ECR (EICC).
- (4) Sets "1" to ID flag of PSW and clears EP flag to "0."
- (5) The value of SB flag of PSW is transmitted to SS flag.
- (6) Sets a handler address for each interrupt to PC and executes a control transfer.

EIPC and EIPSW are used as the status-saving registers. INT inputs are held pending at INTC when one of the two conditions occur: INT input masked by its interrupt controller or interrupt service routine currently being executed (when the NP flag of PSW is "1" or when the ID flag of PSW is "1"). Interrupts are enabled either by clearing the mask condition or using LDSR instruction that sets "0" to the NP flag and the ID flag, prompting the pending INT input to perform the next maskable interrupt servicing. Because only a single set of EIPC and EIPSW is provided, in order to enable multiple interrupts, the contents of both the EIPC register and the EIPSW register need be saved. Figure 6-1 shows a flow for maskable interrupt servicing.



Figure 6-1. Maskable Interrupt Servicing



### 6.1.2 Non-maskable interrupt

The non-maskable interrupt (NMI) are acknowledged at any time and cannot be disabled by an instruction. The non-maskable interrupt is generated through the NMI input, which enables CPU to perform the following steps, followed by a control transfer to the handler routine:

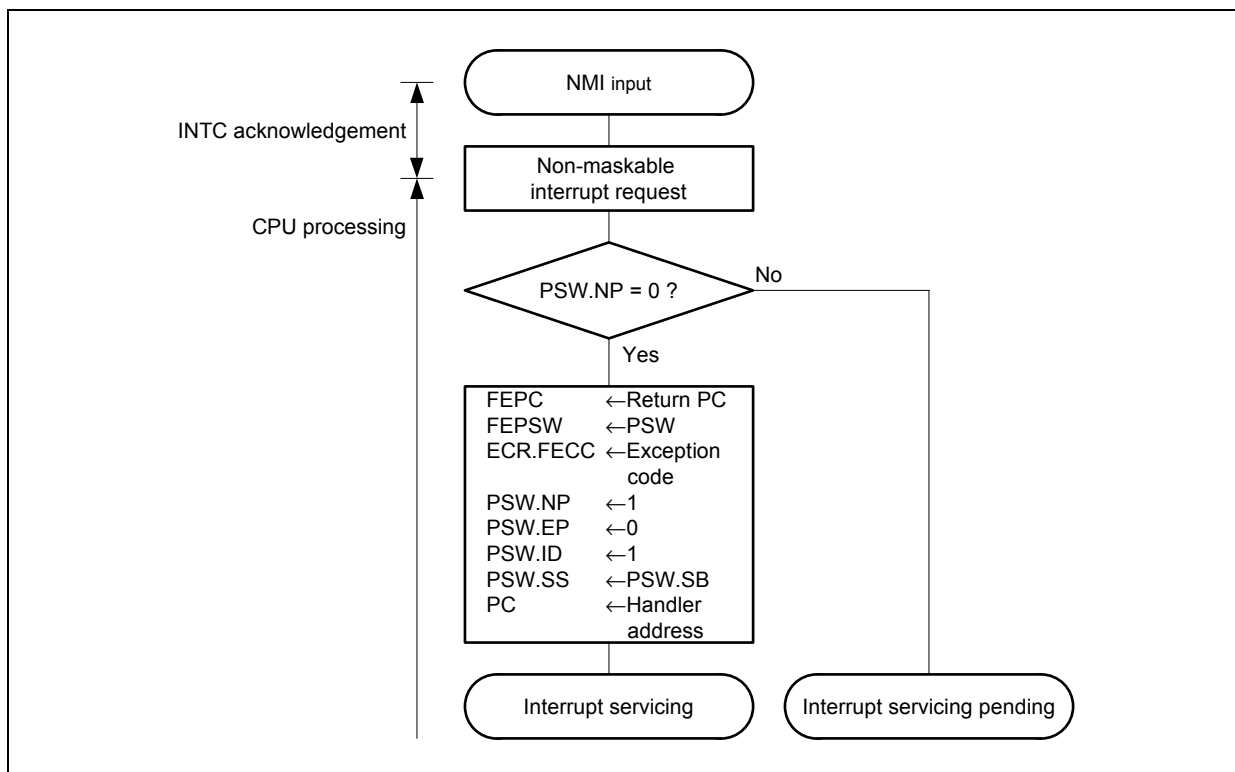
- (1) Saves the restore PC to FEPC.
- (2) Saves the current PSW to FEPSW.
- (3) Writes the exception code (0010H) to the higher half-word of ECR (FECC).
- (4) Sets "1" to NP and ID flags of PSW and clears EP flag to "0."
- (5) The value of SB flag of PSW is transmitted to SS flag.
- (6) Sets a handler address for the non-maskable interrupt to PC and executes a control transfer.

FEPC and FEPSW are used as the status-saving registers. NMI inputs are held pending at the INTC while a non-maskable interrupt is being executed (when the NP flag of PSW is "1"). Non-maskable interrupts are enabled by RETI and LDSR instructions that set "0" to the NP flag of PSW, prompting the pending NMI input to perform the next non-maskable interrupt servicing.

However, when NMI2 or the run time error set as "DIR register 's UTT bit = 1 1" occurs, it is not based on the value of NP flag, but NMI processing is performed.

Figure 6-2 shows a flow for non-maskable interrupt servicing.

**Figure 6-2. Non-Maskable Interrupt Servicing**



## 6.2 Exception Processing

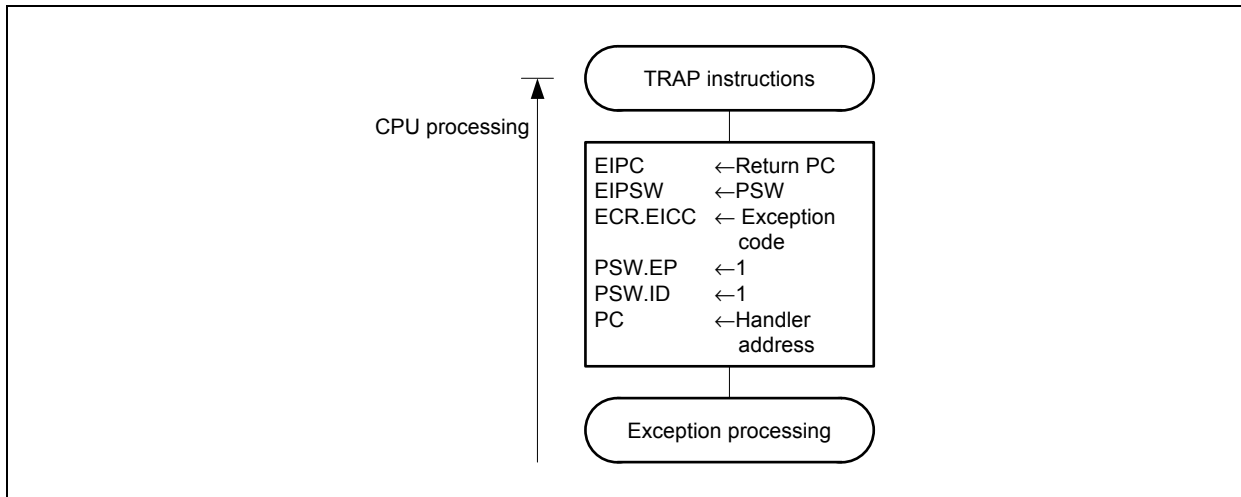
### 6.2.1 Software exception

A software exception is generated by TRAP instruction and can be acknowledged at any time. If a software exception occurs, CPU performs the following steps, followed by a control transfer to the handler routine.

- (1) Saves the return PC to EIPC.
- (2) Saves the current PSW to EIPSW.
- (3) Writes the exception code to the lower 16 bits (EICC) of ECR (interrupt source).
- (4) Sets "1" to EP and ID flags of PSW.
- (5) Sets a handler address (00000040H, 00000050H, or 00000070H) for software exception to PC and executes a control transfer.

Figure 6-3 shows a flow for software exception processing

**Figure 6-3. Software Exception Processing**



An exception trap is generated when an illegal instruction is executed. The illegal opcode trap (ILGOP) is the exception trap.

### 6. 2. 3 Debug traps and debug breaks

Debug traps and debug breaks are exceptions that can always be received.

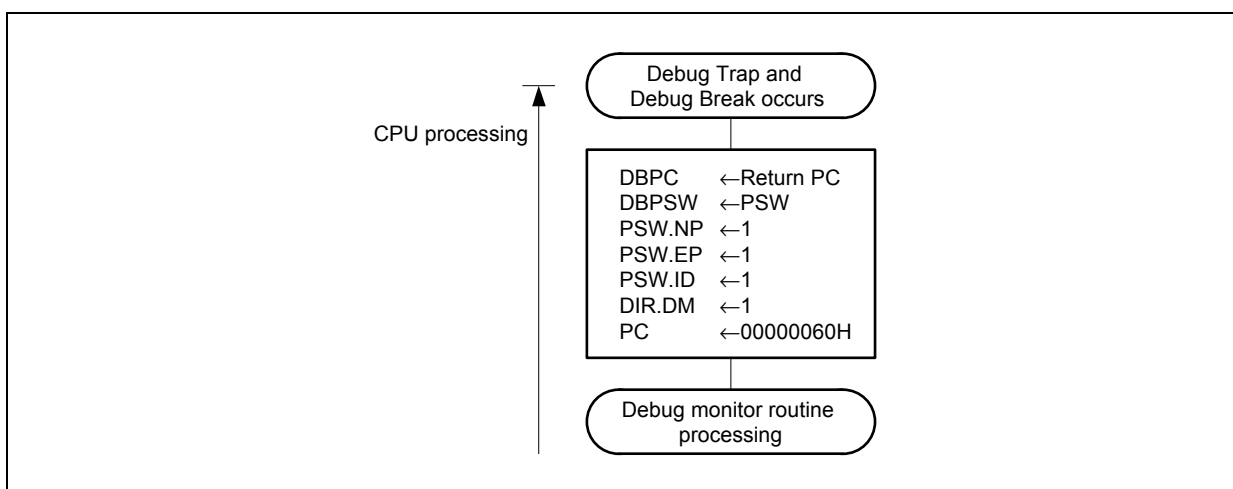
Debug traps are issued by executing a DBTRAP instruction.

When a debug trap or debug break has occurred, the CPU performs the following processing, then control is passed to a debug monitor routine and the mode is switched to debug mode.

- <1> Return PC is saved to DBPC.
- <2> Current PSW value is saved to DBPSW.
- <3> PSW's NP, EP, and ID flags are set (= 1).
- <4> DIR register's DM bit is set (= 1).
- <5> The debug trap or debug break's corresponding handler address (00000060H) is set to the PC, and control is passed to a debug monitor routine.

The processing of debug traps and debug breaks is illustrated below.

**Figure 6-6. Processing of Debug Trap and Debug Break**



## 6.3 Interrupt/Exception Processing Return

### 6.3.1 Interrupt/software exception return

RETI instruction executes all return and restore operations involving interrupts and software exceptions. Under RETI instruction, CPU performs the following steps to execute a transfer control to the return PC address:

- (1) If the EP flag of PSW is "0" and the NP flag of PSW is "1," the return PC and PSW are read from FEPC and FEPSW. The return PC and PSW are otherwise read from EIPC and EIPSW.
- (2) Transfer control is executed on the address of the return PC and PSW.

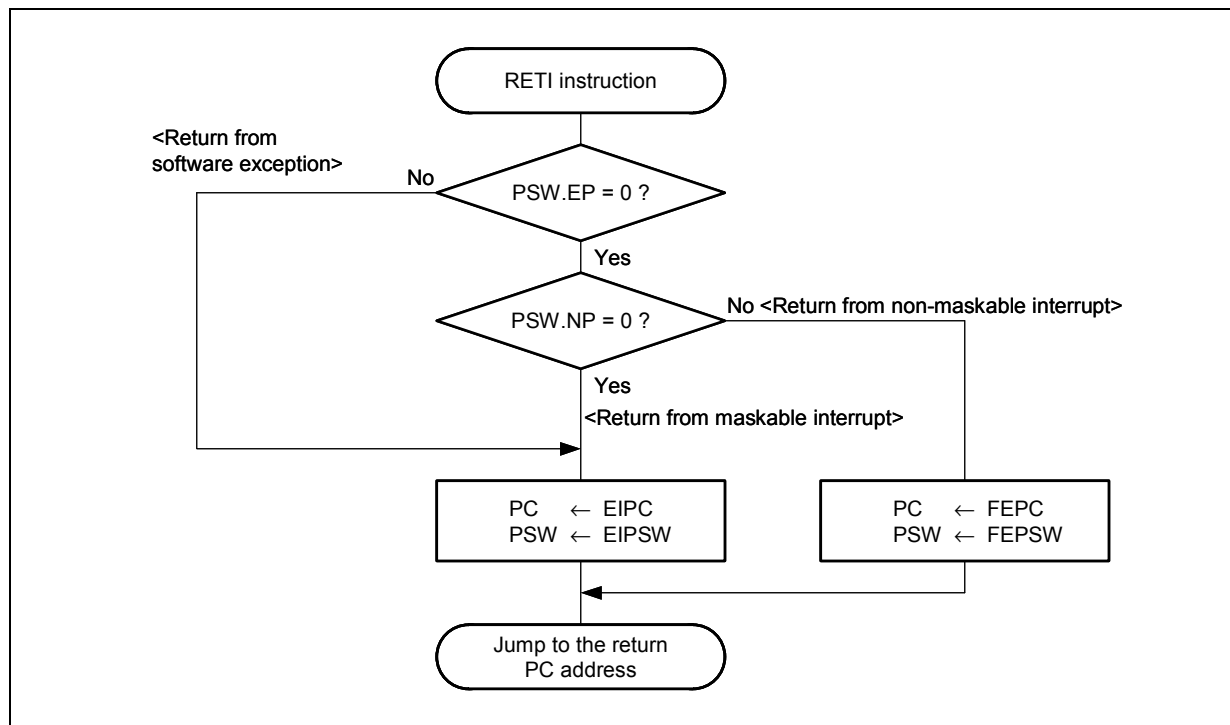
Returns from each interrupt servicing require the NP and EP flags of PSW to be set to the appropriate values immediately before the execution of RETI instruction. This is to ensure a normal operation, and the task requires LDSR instruction. Values are as follows:

- To return from non-maskable interrupt servicing<sup>Note</sup>: NP flag of PSW = "1" and EP flag = "0."
- To return from maskable interrupt servicing: NP flag of PSW = "0" and EP flag = "0."
- To return from exception processing: EP flag of PSW = "1"

**Note** By RETI instruction, the return from NMI1, NMI2 or the run time error set as "DIR register 's UTT bit = 1 1" is not possible. Perform a system reset after interrupt servicing. NMI2 or the run time error set as "DIR register 's UTT bit = 1 1" is received even if NP flag of PSW is set(1).

Figure 6-7 shows a flow for the return operation from interrupt/exception processing.

**Figure 6-7. Return from Interrupt and Software Exceptions**



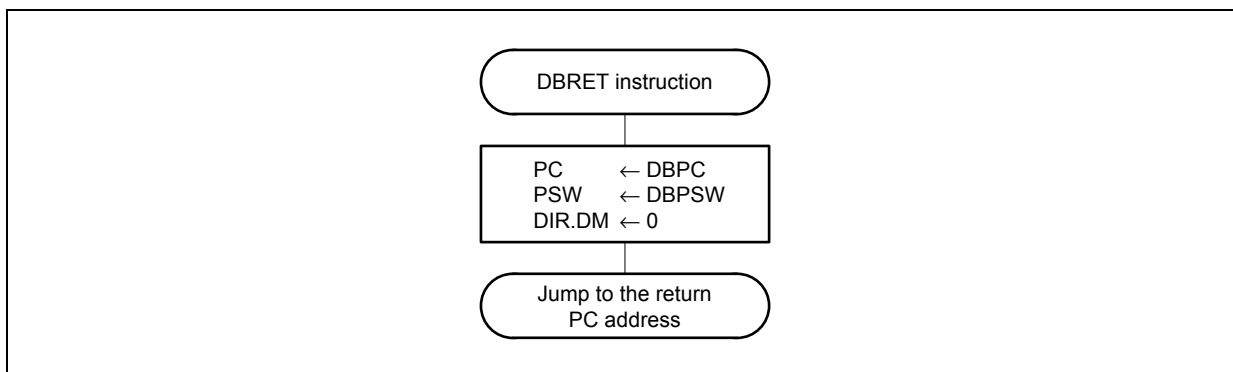
### 6. 3. 2 Exception trap, debug trap, and debug break return

DBRET instruction executes a return from exception trap, debug trap, and debug break. Under DBRET instruction, CPU performs the following steps to execute a transfer control on the return PC address.

- (1) The return PC and PSW are read from DBPC and DBPSW.
- (2) Transfer control is executed on the address of the return PC and PSW.
- (3) DIR register's DM bit is clear (= 0).

Figure 6-8 shows a flow for returns from exception trap/debug trap/debug break processing.

**Figure 6-8. Return from Exception Trap/Debug Trap/Debug Break**



# CHAPTER 7 RESET

## 7.1 Post-Reset Register Status

A low-level signal applied to a reset pin executes the system reset, resulting in an address change to both program registers and system registers. Table 7-1 shows the post-reset status of both program registers and system registers. When the reset signal turns to "high," the reset status is cleared, followed by the program execution. Initialize the contents of each register, as needed.

**Table 7-1. Post-Reset Register Status (1 of 2)**

Register		Post-reset status (initial value)
Program registers	General register (r0)	00000000H (fixed)
	General register (r1 to r31)	Undefined
	Program counter (PC)	00000000H
System registers	Interrupt status-saving register (EIPC)	000x xxxx xxxx xxxx xxxx xxxx xxxx xxx0B
	Interrupt status-saving register (EIPSW)	0000 0000 0000 0000 0000 x00x xxxx xxxxB
	NMI status-saving register (FEPC)	000x xxxx xxxx xxxx xxxx xxxx xxxx xxx0B
	NMI status-saving register (FEPSW)	0000 0000 0000 0000 0000 x00x xxxx xxxxB
	Exception cause register (ECR)	00000000H
	Program status word (PSW)	00000020H
	CALLT status-saving register (CTPC)	000x xxxx xxxx xxxx xxxx xxxx xxxx xxx0B
	CALLT status-saving register (CTPSW)	0000 0000 0000 0000 0000 x00x xxxx xxxxB
	Exception/debug trap status-saving register (DBPC)	000x xxxx xxxx xxxx xxxx xxxx xxxx xxxxB
	Exception/debug trap status-saving register (DBPSW)	0000 0000 0000 0000 0000 x00x xxxx xxxxB
	CALLT base pointer (CTBP)	000x xxxx xxxx xxxx xxxx xxxx xxxx xxx0B
	Debug interface register (DIR)	00000040H
	Breakpoint control register 0 (BPC0)	0000 0000 xxxx xxxx x000 xxxx x000 0000B
	Breakpoint control register 1 (BPC1)	
	Breakpoint control register 2 (BPC2)	
	Breakpoint control register 3 (BPC3)	



Table 7-1. Post-Reset Register Status (2 of 2)

Register		Post-reset status (initial value)
System registers	Program ID register (ASID)	000000xxH
	Breakpoint address setting register 0 (BPAV0)	000x xxxx xxxx xxxx
	Breakpoint address setting register 1 (BPAV1)	xxxx xxxx xxxx xxxxB
	Breakpoint address setting register 2 (BPAV2)	
	Breakpoint address setting register 3 (BPAV3)	
	Breakpoint address mask register 0 (BPAM0)	000x xxxx xxxx xxxx
	Breakpoint address mask register 1 (BPAM1)	xxxx xxxx xxxx xxxxB
	Breakpoint address mask register 2 (BPAM2)	
	Breakpoint address mask register 3 (BPAM3)	
	Breakpoint data setting register 0 (BPDV0)	Undefined
	Breakpoint data setting register 1 (BPDV1)	
	Breakpoint data setting register 2 (BPDV2)	
	Breakpoint data setting register 3 (BPDV3)	
	Breakpoint data mask register 0 (BPDM0)	Undefined
	Breakpoint data mask register 1 (BPDM1)	
	Breakpoint data mask register 2 (BPDM2)	
	Breakpoint data mask register 3 (BPDM3)	

**Remark** "x" indicates an undefined value.

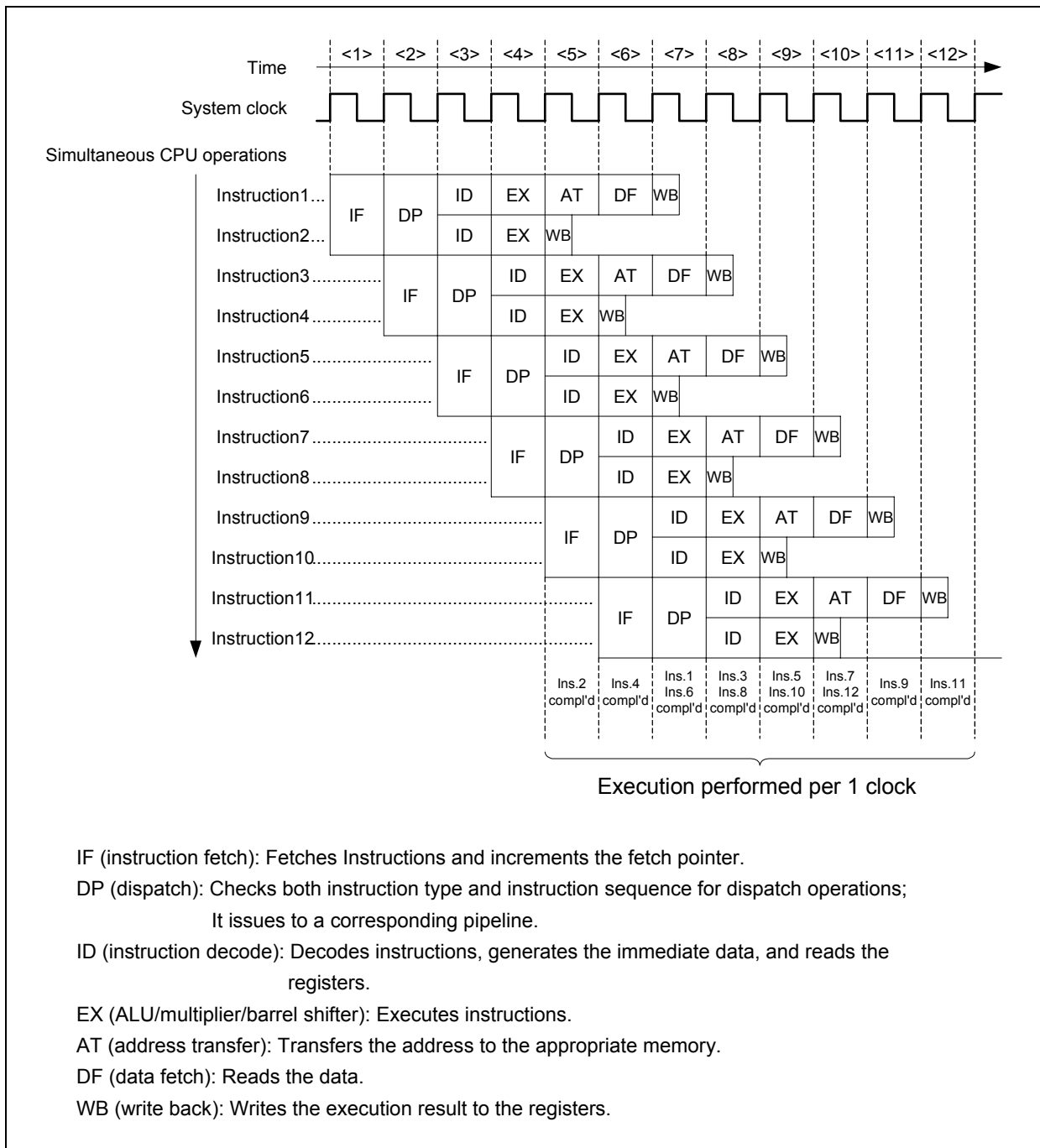
## 7.2 Post-Reset Initialization

CPU begins a program execution from the address 00000000H after it has been reset. No immediate interrupt requests are acknowledged after reset. Program interrupts are enabled with the ID flag of PSW="0."

## CHAPTER 8 PIPELINE OPERATIONS

The V850E2 CPU is based on the RISC architecture, which executes nearly all the instructions in one clock cycle through the 7-stage pipeline operation. The instruction execution sequence consists of seven stages ranging from instruction fetch (IF) to write back (WB), with the execution time depending on the instruction type and the memory access type. Figure 8-1 shows an example of pipeline operation where twelve standard instructions are simultaneously executed.

Figure 8-1. Example of Executing Twelve Standard Instructions



**Remark** <1> to <12> indicate CPU state. Standard instruction enables the parallel execution of 2 instructions per one clock.

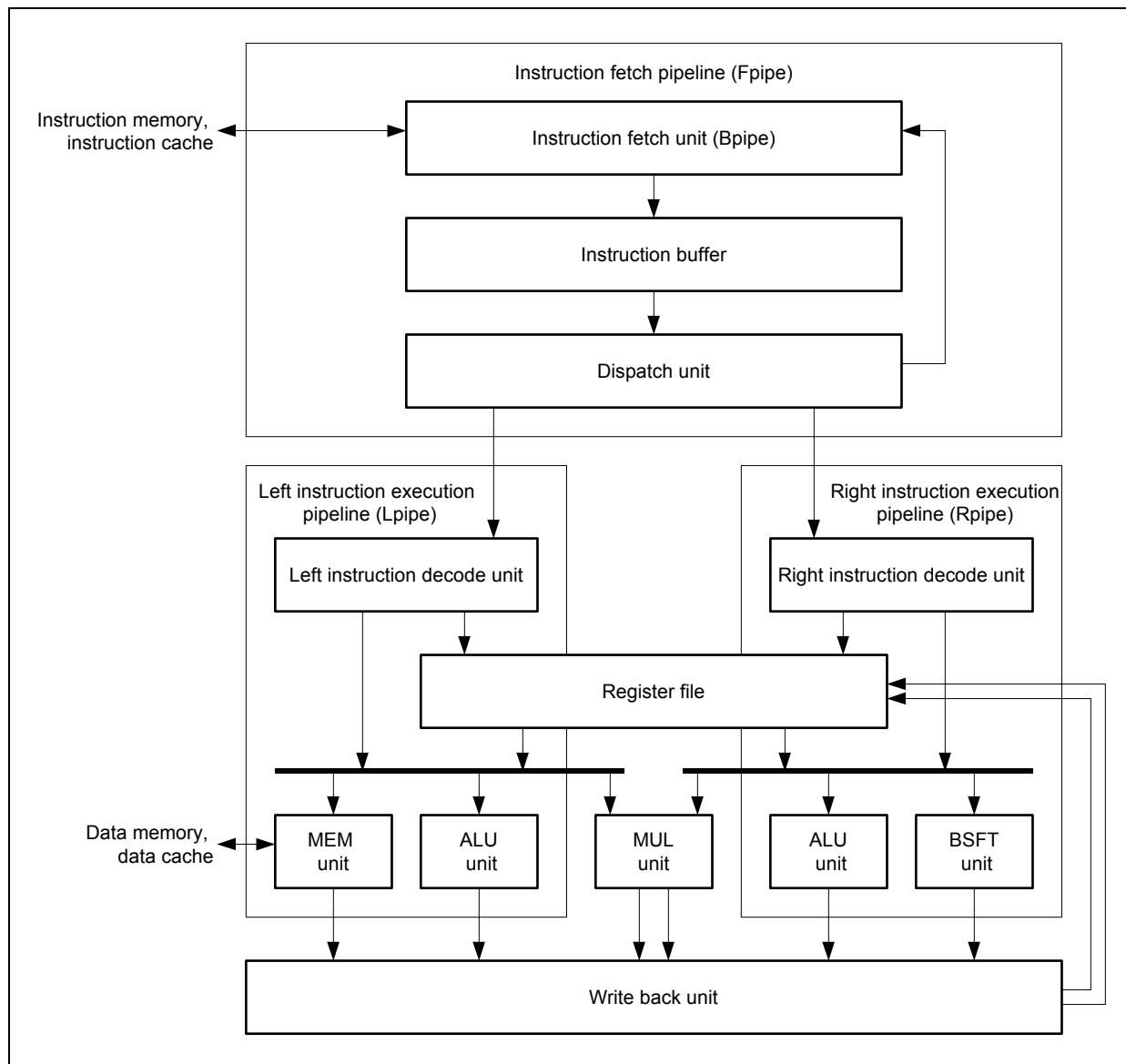
## 8.1 Features

The V850E2 CPU includes the following three independent pipelines.

- Instruction fetch pipeline (Fpipe)
- Left instruction execution pipeline (Lpipe)
- Right instruction execution pipeline (Rpipe)

The V850E2 CPU detects interdependencies among instructions and is configured to enable up to two instructions to be issued at the same time. Figure 8-2 shows the configuration of the V850E2 CPU's pipelines.

**Figure 8-2. Pipeline Configuration**



**(1) Instruction fetch pipeline (Fpipe)**

This pipeline includes the following three units.

**(a) Instruction fetch unit (Bpipe)**

Up to 8 instructions (when there are 16 bits per instruction) can be fetched in one cycle from the 128-bit fetch bus (iLB).

**(b) Dispatch unit**

A 128-bit, two-stage instruction queue is included. This queue is used to detect interdependencies among instructions so that up to 2 instructions can be issued efficiently to the instruction execution pipeline.

**(c) Instruction buffer**

The instruction stores instructions that have been fetched by the instruction fetch unit (Bpipe).

**(2) Left instruction execution pipeline (Lpipe)**

This pipeline includes the following three units.

**(a) Left instruction decode unit**

This unit decodes instructions issued from the dispatch unit.

**(b) ALU unit**

This unit executes instructions that perform integer arithmetic operations and logical operations.

**(c) MEM unit**

This unit executes instructions that perform memory access, such as load instructions and store instructions.

**(3) Right instruction execution pipeline (Rpipe)**

This pipeline includes the following three units.

**(a) Right instruction decode unit**

This unit decodes instructions issued from the dispatch unit.

**(b) ALU unit**

This unit executes instructions that perform integer arithmetic operations and logical operations.

**(c) BSFT unit**

This unit executes data manipulation instructions.

**(4) MUL unit**

This unit executes instructions that perform integer multiplication.

**(5) Write back unit**

This unit controls write back operations to register files.

## 8.2 Pipeline Flow during Execution of Instructions

This section describes the flow of pipeline processing when various instructions are being executed.

### 8.2.1 Load instructions

Load instructions are executed by the left instruction execution pipeline (Lpipe)'s MEM unit.

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

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
Load instruction	IF	DP	ID	EX	AT	DF	WB	
Next instruction		IF	DP	ID	EX	AT	DF	WB

[Description] This pipeline has seven stages: IF, DP, ID, EX, AT, DF, and WB. In the figure above, a load instruction is executed by the Lpipe and then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the load instruction, it can execute its own processing independently. However, immediately after the load instruction is executed, if an instruction that uses the execution result is issued, a data wait period will occur.

Each of these instructions can be issued at the same time as another instruction.

### 8.2.2 Store instructions

Store instructions are executed by the left instruction execution pipeline (Lpipe)'s MEM unit.

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

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
Store instruction	IF	DP	ID	EX	AT	DF	WB	
Next instruction		IF	DP	ID	EX	AT	DF	WB

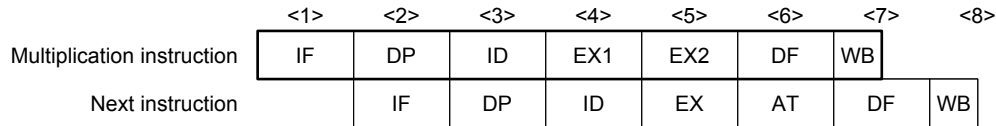
[Description] This pipeline also has seven stages (IF, DP, ID, EX, AT, DF, and WB), but its WB stage does not operate because there is no writing of data to registers. In the figure above, a store instruction is executed by the Lpipe and then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the store instruction, it can execute its own processing independently. Each of these instructions can be issued at the same time as another instruction.

### 8.2.3 Multiplication instructions

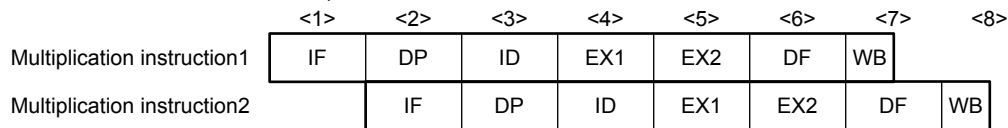
The multiplication instructions are executed by the left instruction execution pipeline (Lpipe)'s MUL unit.

[Target instructions] MUL, MULH, MULHI, and MULU

[Pipeline] (a) If the next instruction is not a multiplication instruction (or a multiplication with addition instruction)



(b) If the next instruction is a multiplication instruction (or a multiplication with addition instruction)



[Description] This pipeline has seven stages: IF, DP, ID, EX, AT, DF, and WB. Although two clock cycles are required by the EX stages, EX1 and EX2 operate independently. Consequently, only one clock cycle is required per instruction even when the multiplication instruction (or multiplication with addition instruction) is repeated.

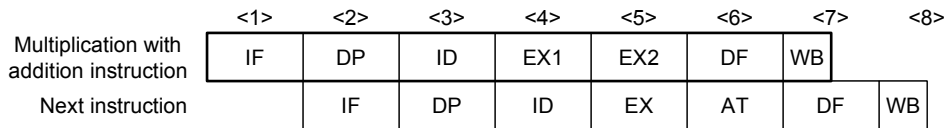
In the figure above, a multiplication instruction is executed by the Lpipe and then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the multiplication instruction, it can execute its own processing independently. However, immediately after the multiplication instruction is executed, if an instruction that uses the execution result is issued, a data wait period will occur. Each of these instructions can be issued at the same time as another instruction.

## 8. 2. 4 Multiplication with addition instructions

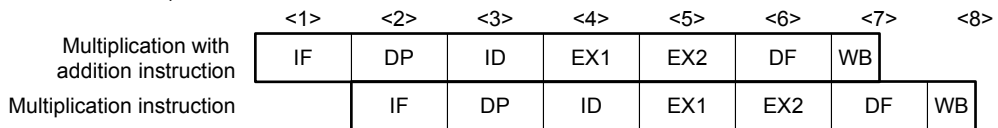
Multiplication with addition instructions are executed by the left instruction execution pipeline (Lpipe)'s MUL unit.

[Target instructions]      MAC and MACU

[Pipeline]      (a) When the next instruction is not a multiplication instruction (or multiplication with addition instruction)



(b) When the next instruction is a multiplication instruction (or multiplication with addition instruction)



[Description]      This pipeline has seven stages: IF, DP, ID, EX, AT, DF, and WB. Although two clock cycles are required by the EX stages, EX1 and EX2 operate independently. Consequently, only one clock cycle is required per instruction even when the multiplication instruction (or multiplication with addition instruction) is repeated.

In the figure above, a multiplication instruction is executed by the Lpipe and then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the multiplication instruction, it can execute its own processing independently. However, immediately after the multiplication instruction is executed, if an instruction that uses the execution result is issued, a data wait period will occur.

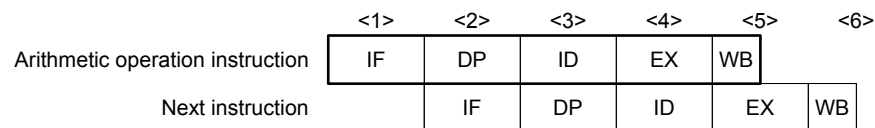
These instructions are issued one at a time.

## 8. 2. 5 Arithmetic operation instructions

Arithmetic operation instructions are executed by the ALU unit of the left instruction execution pipeline or the right instruction execution pipeline (Lpipe or Rpipe).

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

[Pipeline]



[Description]      This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, an arithmetic operation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the Lpipe has no dependency with the arithmetic operation instruction, it can execute its own processing independently.

Each instruction except for the MOV imm32 reg1 instruction can be issued at the same time as another instruction (the MOV imm32 reg1 instruction must be issued by themselves).

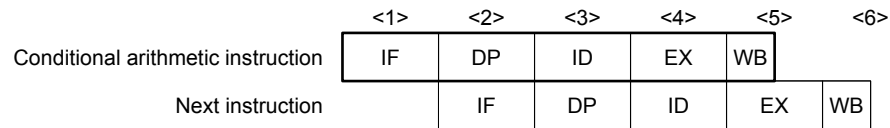


### 8.2.6 Conditional arithmetic instructions

Conditional arithmetic instructions are executed by the ALU unit of the left instruction execution pipeline or right instruction execution pipeline (Lpipe or Rpipe).

[Target instructions]      ADF and SBF

[Pipeline]



[Description]      This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, an arithmetic operation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the Lpipe has no dependency with the arithmetic operation instruction, it can execute its own processing independently.

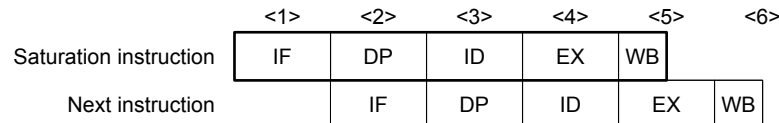
These instructions are issued one at a time.

### 8.2.7 Saturation instructions

Saturation instructions are executed by the ALU unit of the left instruction execution pipeline or the right instruction execution pipeline (Lpipe or Rpipe).

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

[Pipeline]



[Description]      This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, a saturation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the Lpipe has no dependency with the saturation instruction, it can execute its own processing independently.

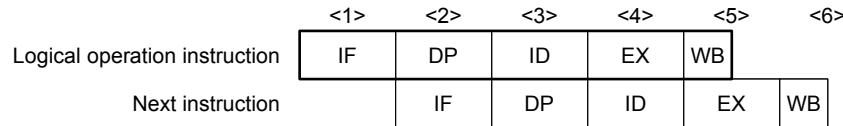
Each instruction except for the SATADD reg1, reg2, reg3 instruction and the SATSUB reg1, reg2, reg3 instruction can be issued at the same time as another instruction (the SATADD reg1, reg2, reg3 instruction and SATSUB reg1, reg2, reg3 instruction must be issued by themselves).

### 8. 2. 8 Logical operation instructions

Logical operation instructions are executed by the ALU unit of the left instruction execution pipeline or right instruction execution pipeline (Lpipe or Rpipe).

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

[Pipeline]



[Description]      This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, a logical operation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the Lpipe has no dependency with the logical operation instruction, it can execute its own processing independently.

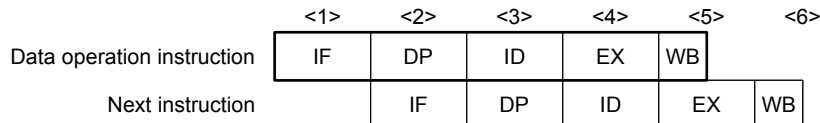
Each of these instructions can be issued at the same time as another instruction.

### 8. 2. 9 Data operation instructions

Data operation instructions are executed by the right instruction execution pipeline (Rpipe)'s BSFT unit.

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

[Pipeline]



[Description]      This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, a data operation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the left instruction execution (Lpipe) has no dependency with the data operation instruction, it can execute its own processing independently.

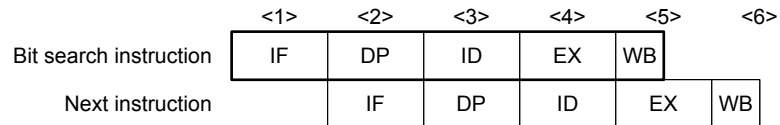
Each of these instructions can be issued at the same time as another instruction.

### 8. 2. 10 Bit search instructions

Bit search instructions are executed by the right instruction execution pipeline (Rpipe)'s BSFT unit.

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

[Pipeline]



[Description] This pipeline has five stages: IF, DP, ID, EX, and WB.

In the figure above, a data operation instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe. If the left instruction execution (Lpipe) has no dependency with the data operation instruction, it can execute its own processing independently.

Each of these instructions can be issued at the same time as another instruction.

### 8. 2. 11 Division instructions

Division instructions are executed by the right instruction execution pipeline (Rpipe)'s ALU unit.

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

[Pipeline]

(a) When DIV or DIVH



:- Idle inserted for wait timing

(b) When DIVU or DIVHU



:- Idle inserted for wait timing

[Description] For the DIV and DIVH instructions, the pipeline has 39 stages: IF, DP, ID, EX1 to EX35, and WB. For the DIVU and DIVHU instructions, it has 38 stages: IF, DP, ID, EX1 to EX34, and WB. In the figure above, a division instruction is executed by the Rpipe and then the next instruction is issued to the Rpipe.

However, the dispatch unit does not issue any instructions to the Rpipe during the time when a division instruction is being decoded in the ID stage or when it is being executed during the EX stages.

These instructions are issued one at a time.

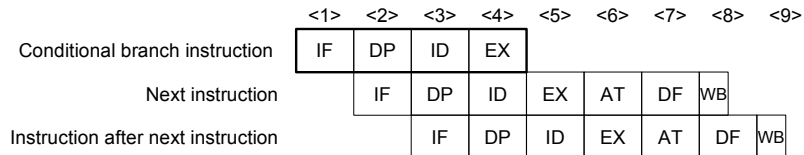
## 8. 2. 12 Branch instructions

Branch instructions are executed by the instruction fetch unit (Bpipe).

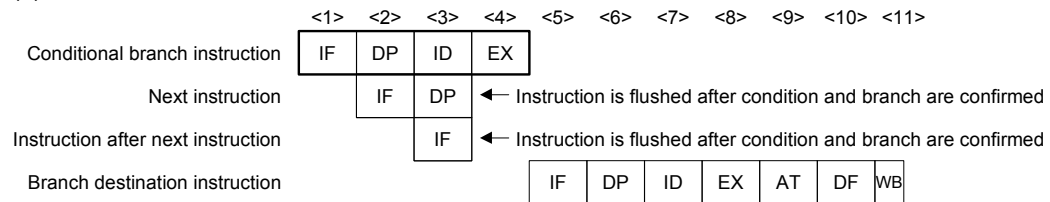
### (1) Conditional branch instruction (excludes BR instruction)

[Target instruction] Bcond

[Pipeline] (a) When condition has not been met



(b) When condition has been met



[Description]

The figure above shows a Bcond instruction being executed by the Bpipe, with all instructions being executed via the left instruction execution pipeline (Lpipe). Each of these instructions can be executed at the same time as another instruction. The numbers of execution clock cycles are listed below.

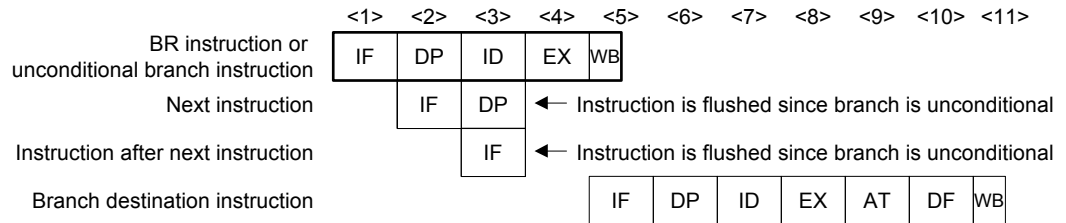
Branch instruction	Execution clock cycles
(a) When condition is not met	1
(b) When condition is met	4 <sup>Note</sup>

**Note** This number is 3 ( $4 - 1 = 3$ ) if there are no target instructions in the instruction buffer. This number is 6 if a PSW write instruction was executed as the previous instruction.

**(2) BR instruction and unconditional branch instructions (excluding JMP instruction)**

[Target instructions] BR, JARL, and JR

[Pipeline]

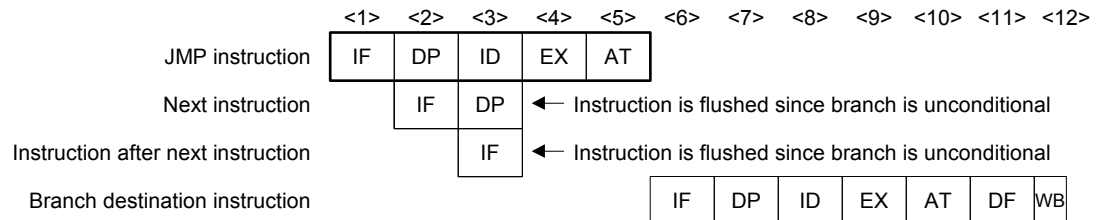


[Description]

The figure above shows a branch instruction being executed by the Bpipe, with all instructions being executed via the left instruction execution pipeline (Lpipe). Each of these instructions, except for the JARL disp32, reg1 instruction and JR disp32 instruction, can be executed at the same time as another instruction. Four clock cycles are required for execution of these instructions (this number is 3 (4 - 1 = 3) if there are no target instructions in the instruction buffer).

**(3) JMP instruction**

[Pipeline]



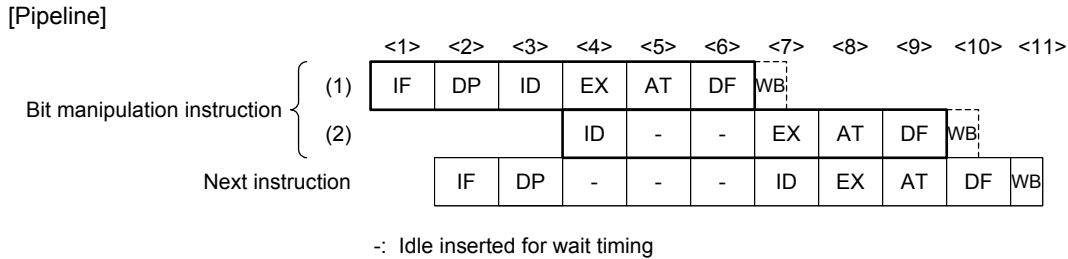
[Description]

The figure above shows a JMP instruction being executed by the Bpipe, with all instructions being executed via the left instruction execution pipeline (Lpipe). The JMP [reg1] instruction can be executed at the same time as another instruction (this is not possible for the JMP disp32 [reg1] instruction). Five clock cycles are required for execution of these instructions (this number is 4 (5 - 1 = 4) if there are no target instructions in the instruction buffer).

## 8. 2. 13 Bit manipulation instructions

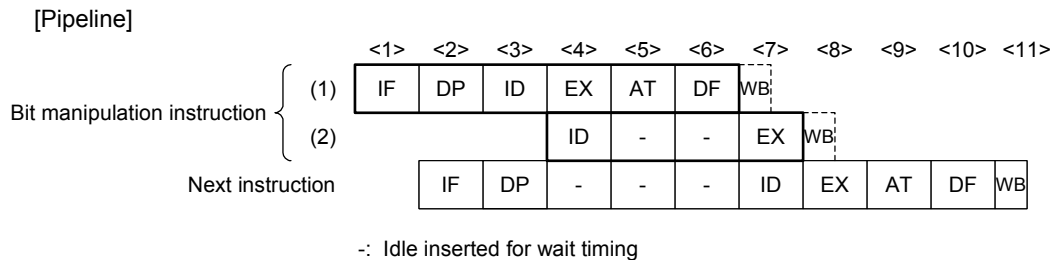
Bit manipulation instructions are executed by the left instruction execution pipeline (Lpipe)'s ALU unit.

### (1) CLR1, NOT1, and SET1 instructions



[Description] Each instruction is divided into two instructions at the ID stage. The load instruction is executed first, then the store instruction that includes bit manipulation is executed. However, since there is no writing of data to registers, nothing occurs at the WB stage. In the figure above, a bit manipulation instruction is executed by the Lpipe and then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the bit manipulation instruction, it can execute its own processing independently. The dispatch unit is not able to issue instructions to the Lpipe during decoding of instructions at the ID stage. These instructions are issued one at a time.

### (2) TST1 instruction



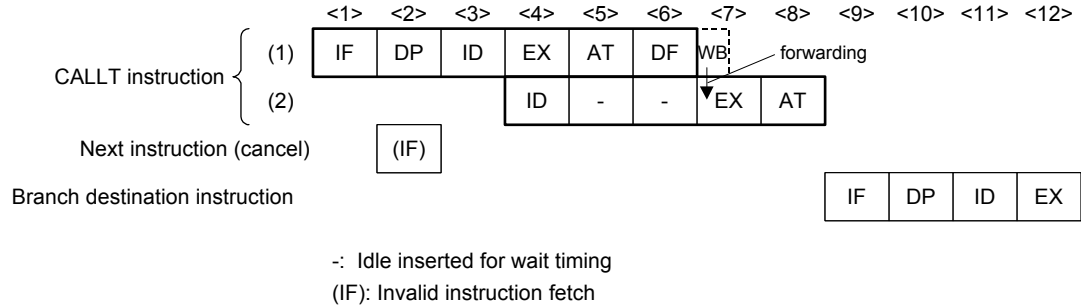
[Description] Each instruction is divided into two instructions at the ID stage. The load instruction is executed first, then the bit manipulation instruction is executed. However, since there is no writing of data to registers, nothing occurs at the WB stage. In the figure above, the TST1 instruction is executed by the Lpipe, then the next instruction is issued to the Lpipe. If the Rpipe has no dependency with the bit manipulation instruction, it can execute its own processing independently. The dispatch unit is not able to issue instructions to the Lpipe during decoding of instructions at the ID stage. These instructions are issued one at a time.

## 8. 2. 14 Special instructions

### (1) CALLT instruction

The CALLT instruction is executed by the left instruction execution pipeline (Lpipe)'s ALU unit.

[Pipeline]



[Description]

This instruction is divided into two instructions at the ID stage. The load instruction is executed first, then the branch instruction corresponding to CTBP is executed. However, since there is no writing of data to registers, nothing occurs at the WB stage. In the above figure, the CALLT instruction is executed by the Lpipe, then an instruction is fetched from the branch destination. If the right instruction execution pipeline (Rpipe) has no dependency with the CALLT instruction, it can execute its own processing independently.

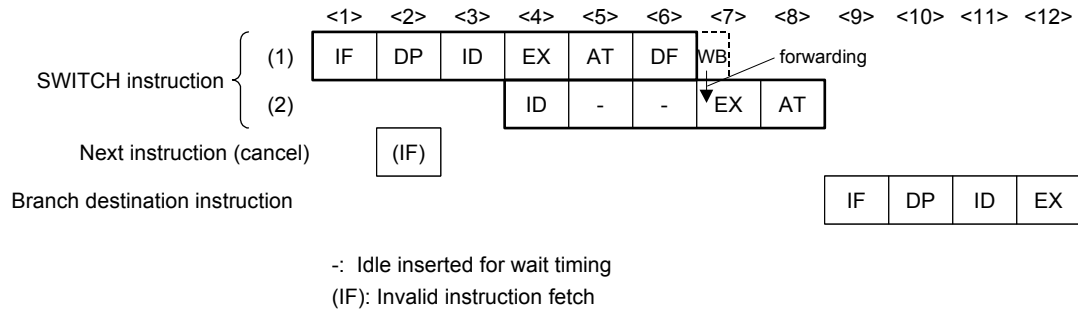
These instruction is issued one at a time.

The number of execution clock cycles is eight.

### (2) SWITCH instruction

The SWITCH instruction is executed by the left instruction execution pipeline (Lpipe)'s ALU unit.

[Pipeline]



[Description]

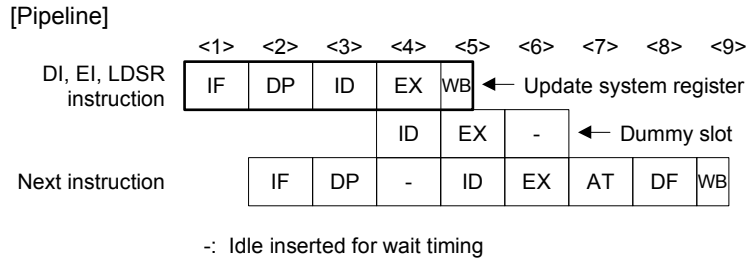
This instruction is divided into two instructions at the ID stage. The load instruction is executed first, then the branch instruction corresponding to PC is executed. However, since there is no writing of data to registers, nothing occurs at the WB stage. In the above figure, the SWITCH instruction is executed by the Lpipe, then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the SWITCH instruction, it can execute its own processing independently.

These instruction is issued one at a time.

The number of execution clock cycles is eight.

### (3) DI, EI, and LDSR instructions

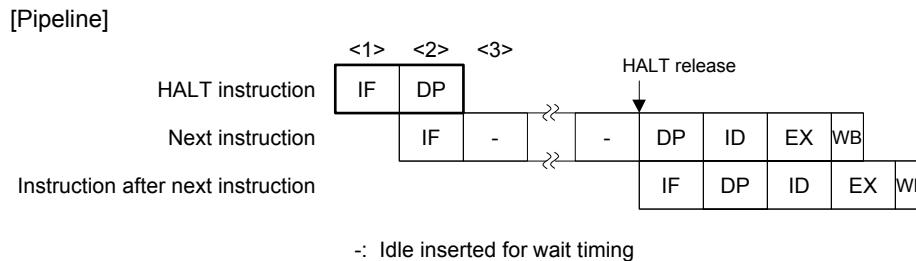
The DI, EI, and LDSR instructions are executed by the right instruction execution pipeline (Rpipe)'s ALU unit.



[Description] This pipeline has five stages: IF, DP, ID, EX, and WB.  
In the above figure, a DI, EI, or LDSR instruction is executed by the Rpipe, and all other instructions are also executed by the Rpipe.  
These instructions are issued one at a time.

### (4) HALT instruction

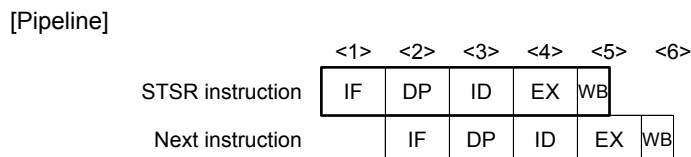
The HALT instruction is executed by the instruction fetch pipeline (Fpipe)'s dispatch unit.



[Description] Once a HALT instruction is detected at the DP stage, instructions cannot be issued to the ID stage until the HALT instruction has been canceled. Consequently, when the next instruction is issued, the ID stage is delayed for that instruction until the HALT instruction is canceled. In the above figure, the right instruction execution pipeline (Rpipe) executes the HALT instruction, then the next instruction is issued to the Rpipe.  
These instruction is issued one at a time.

### (5) STSR instruction

The STSR instruction is executed by the right instruction execution pipeline (Rpipe)'s ALU unit.



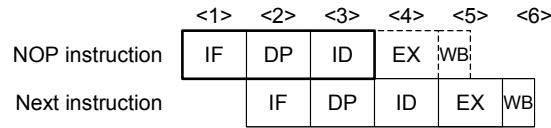
[Description] This pipeline has five stages: IF, DP, ID, EX, and WB.  
In the above figure, the STSR instruction is executed by the Rpipe, then the next instruction is issued to the Rpipe. If the left instruction execution pipeline (Lpipe) has no dependency with the STSR instruction, it can execute its own processing independently.  
These instruction is issued one at a time.



### (6) NOP instruction

The NOP instruction is executed by the ALU unit of the left instruction execution pipeline or the right instruction execution pipeline (Lpipe or Rpipe).

[Pipeline]



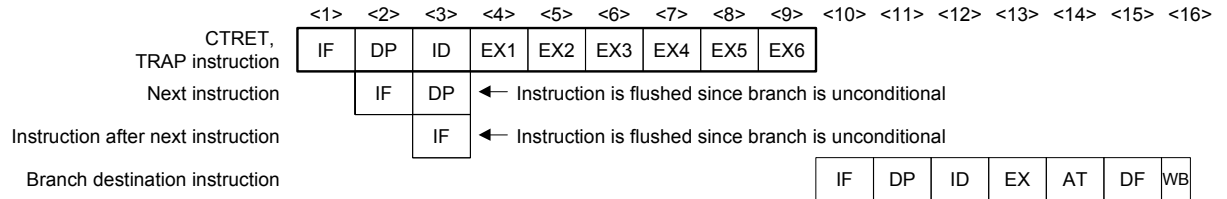
[Description]

This pipeline has five stages: IF, DP, ID, EX, and WB, but since there are no processing and no writing of data to registers, there are no operations at the EX and WB stages. In the above figure, the NOP instruction is executed by the Rpipe, then the next instruction is issued to the Rpipe. If the Lpipe has no dependency with the NOP manipulation instruction, it can execute its own processing independently. This instruction can be issued at the same time as another instruction.

### (7) CTRET and TRAP instructions

The CTRET and TRAP instructions are executed by the instruction fetch unit (Bpipe).

[Pipeline]



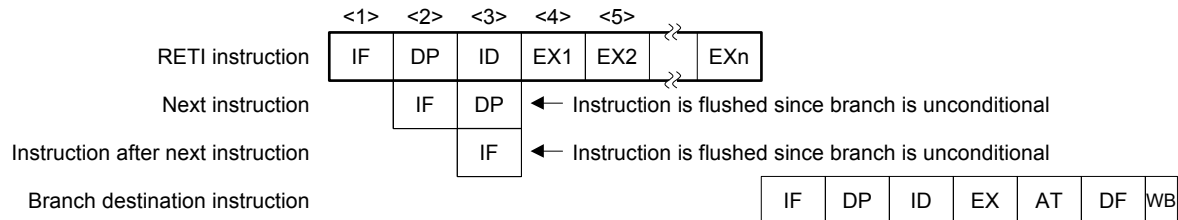
[Description]

In the above figure, a CTRET or TRAP instruction is executed by the Bpipe, with all instructions executed via the Lpipe. These instructions are issued one at a time. The number of execution clock cycles is nine.

### (8) RETI instruction

The RETI instruction is executed by the instruction fetch unit (Bpipe).

[Pipeline]

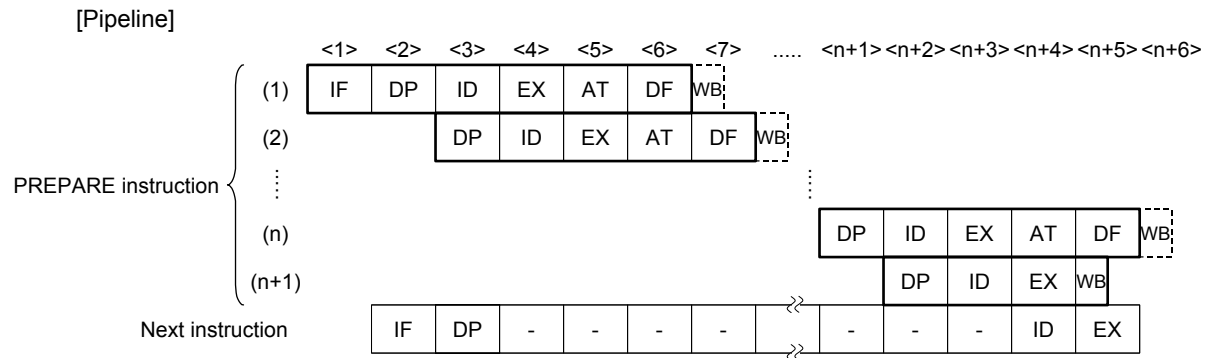


[Description]

In the above figure, a RETI instruction is executed by the Bpipe, with all instructions executed via the left instruction execution pipeline (Lpipe). These instructions are issued one at a time. The number of execution clock cycles is varies according to the system (it depends on the interrupt controller's operation specifications).

### (9) PREPARE instruction

The PREPARE instruction is executed by the left instruction execution pipeline (Lpipe)'s ALU unit.



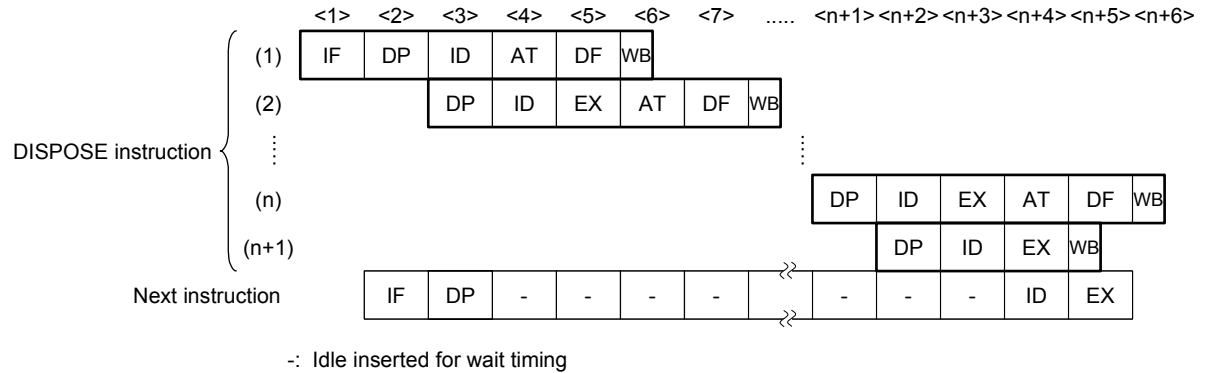
**Remark** n is the number of registers specified in the register list (list12).

[Description] This instruction is divided into n + 1 instructions at the DP stage, and the store instruction of the first n instruction is executed first, then an instruction that writes to the stack pointer (SP) is executed. However, since the store instruction does not write any data to registers, no operations occur at the WB stage. In the above figure, the PREPARE instruction is executed by the Lpipe, then the next instruction is issued to the Rpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the PREPARE instruction, it can execute its own processing independently. The dispatch unit does not issue any instructions to the Lpipe when an instruction is being decoded in the DP stage. This instruction is issued one at a time.

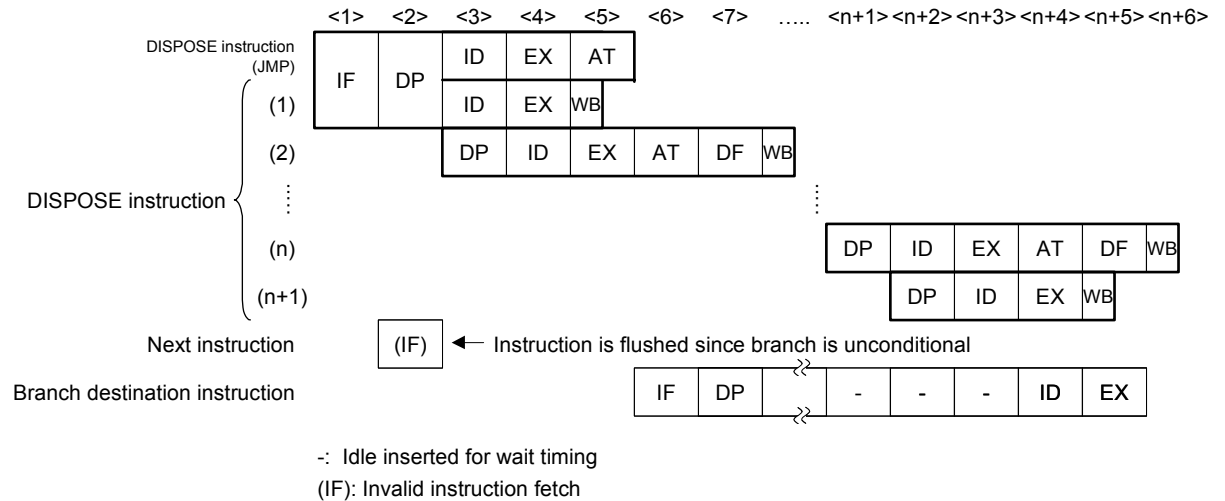
### (10) DISPOSE instruction

The DISPOSE instruction is executed by the right instruction execution pipeline (Rpipe)'s ALU unit.

[Pipeline] (a) When branch does not occur



(b) When branch occurs



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

[Description] This instruction is divided into n + 1 instructions at the DP stage, and the load instruction of the first n instruction is executed first, then an instruction that writes to the stack pointer (SP) is executed. In the above figure, DISPOSE instruction is executed by the Lpipe, then the next instruction is issued to the Lpipe. If the right instruction execution pipeline (Rpipe) has no dependency with the DISPOSE instruction, it can execute its own processing independently.

The dispatch unit does not issue any instructions to the Lpipe when an instruction is being decoded in the DP stage.

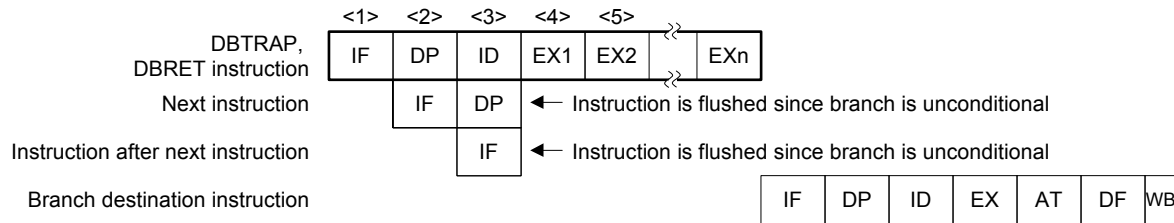
This instruction is issued one at a time.

## 8. 2. 15 Instructions for debug function

All instructions for the debug function are executed by the instruction fetch unit (Bpipe).

[Target instructions] DBTRAP and DBRET

[Pipeline]



[Description] In the above figure, a DBTRAP or DBRET instruction is executed by the Bpipe, and all other instructions are executed by the left instruction execution pipeline (Lpipe). These instructions are issued one at a time. Since this instruction is retained in the CPU, the branch destination instruction is not executed before completion of this instruction's processing.

# CHAPTER 9 SHIFTING TO DEBUG MODE

When a debug trap, exception trap, or debug break occurs, the V850E2 CPU sets the handler address (00000060H) to the program counter (PC) and shifts to debug mode.

The mode can be shifted to debug mode each time a single-step operation is set and an instruction is executed.

**Caution** When the mode is shifted to debug mode, the data cache (dCACHE) is set to Hold mode and its data and tags are not updated. If a cacheable area of external memory is accessed during debug mode, cohesion may be lost even when the dCACHE is valid and access is only to external memory. Therefore, before manipulating any data in a cacheable area as part of a debug monitor routine, be sure to first return to user mode and clear the dCACHE (for write through) or flush and clear it (for write back).

## 9. 1 Methods for Shifting to Debug Mode

### (1) Debug trap

When the DBTRAP instruction is executed, a debug trap occurs and the mode is shifted to debug mode.

### (2) Exception trap

When an instruction is executed incorrectly (or illegally), an exception trap occurs and the mode is shifted to debug mode.

### (3) Debug break

There are three types of debug breaks, as described below.

- Break set by breakpoint (4 channels)
- Break triggered by misaligned access exception
- Break triggered by alignment error exception

The following system registers are used to set debug breaks.

- Debug Interface register (DIR)
- Breakpoint Control registers 0 to 3 (BPC0 to BPC3)
- Breakpoint Address Setup registers 0 to 3 (BPAV0 to BPAV3)
- Breakpoint Address Mask registers 0 to 3 (BPAM0 to BPAM3)
- Breakpoint Data Setup registers 0 to 3 (BPDV0 to BPDV3)
- Breakpoint Data Mask registers 0 to 3 (BPDM0 to BPDM3)

#### (a) Break set by breakpoint (4 channels)

The mode is shifted to debug mode based on breakpoints that are set (for 4 channels) when the following break conditions are met. These conditions are set via the BPCn registers (n = 0 to 3).

**Caution** When the BPCn register's IE bit is set, if the Program IDs set via the BP ASID bit and ASID register do not match, the mode will not be shifted to debug mode even if the break condition has been met.

Table 9-1. Break Conditions

Type	Break condition		Break timing	BPxxn register's setting <sup>Note 2</sup>				BPCn register's MD, FE, RE, and WE bit settings		
	Address <sup>Note 1</sup>	Data		BP AVn	BP AMn	BP DVn	BP DMn	MD	FE	RE, WE
Execution-related trap	Any execution address	-	Before execution	<1>	<1>	<1>	<1>	1	1	0 <sup>Note 4</sup>
	Designated execution address	-		✓	<0>	<1>	<1>			
	Designated execution address range	-		✓	✓	<1>	<1>			
Access-related trap	Any access address	Designated data	After execution <sup>Note 3</sup>	<1>	<1>	✓	<0>	0	0	0/1 <sup>Note 5</sup>
		Designated data range	Immediately After execution	<1>	<1>	✓	✓			
	Designated access address	Any data	After execution <sup>Note 3</sup>	✓	<0>	<1>	<1>	Don't care		
		Designated data		✓	<0>	✓	<0>	0		
		Designated data range		✓	<0>	✓	✓			
	Designated access address range	Any data	Immediately After execution	✓	✓	<1>	<1>	Don't care		
		Designated data	After execution <sup>Note 3</sup>	✓	✓	✓	<0>	0		
		Designated data range		✓	✓	✓	✓			

**Notes**1. Execution addresses refers to addresses used during instruction fetch operations and access addresses refer to addresses where access occurs during instruction execution.

2. Enter the following settings.

✓: Set a break condition.

<0>: Zero-clear all bits.

<1>: A condition setting is not required, but since the initial value is undefined, all bits should be set (= 1) (except for bits 31 to 29 in the BPAVn and BPAMn registers, which are fixed to zero).

3. When writing data: After execution

When reading data: Execution after several instruction (slip)

4. Bit value must be 0 (operation is not guaranteed if any of these bits are set (= 1)).

5. Set according to type of access (read only, write only, or read/write-accessible).

**Remarks**1. n = 0 to 3

2. If several break conditions have been set, the mode will be shifted to debug mode if any of the previously set break conditions is met.

The following two types of operations can be performed sequentially between channels 0 and 1, or channels 2 and 3 (these operations cannot be performed at the same time).

**<1> Break triggered by sequential execution (sequential break mode)**

The mode is shifted to debug mode only when break conditions are met in the order of channels 0 → 1 or channels 2 → 3. To make this setting, set a "1" to the SQ0 bit (when using channels 0 and 1) or the SQ1 bit (when using channels 2 and 3) in the Debug Interface register (DIR).

**<2> Break triggered by simultaneous execution (latency break mode)**

The mode is shifted to debug mode only when break conditions are met at the same time in channels 0 and 1, or in channels 2 or 3. To make this setting, set a "1" to the RE0 bit (when using channels 0 and 1), or the RE1 bit (when using channels 2 and 3) in the Debug Interface register (DIR).

- Cautions 1. The timing by which break conditions are met differs between execution-related traps and access-related traps. Consequently, even when sequential break mode has been set, normal operation may not occur if an execution-related trap occurs after an access-related trap.**
- 2. When in latency break mode, set either execution-related traps or access-related traps (when using channels 0 and 1 or channels 2 and 3).**

**(b) Break triggered by misaligned access exception**

To make this setting, set "1" to the MT bit in the Debug Interface register (DIR).

If misaligned access occurs during execution of a load instruction or store instruction, the mode is shifted to debug mode (this does not depend on enabled/prohibited setting for misaligned access, set by the level of input to the IFIMAEN pin).

**(c) Break triggered by alignment error exception**

To make this setting, set "1" to the AT bit in the Debug Interface register (DIR).

The mode is shifted to debug mode when an alignment error occurs.

An alignment error can occur in the following cases.

- When the stack pointer (SP) is forcibly aligned so as not to be at the word boundary during execution of a PREPARE or DISPOSE instruction

When a debug break occurs, the address of the instruction that triggered the break is stored in the DBPC, except when there is an access-related trap while using channels 0 to 3 (since the mode is shifted to debug mode before execution of the instruction is completed). Therefore, after shifting from debug mode to user mode, the instruction which caused a break is re-detected, and an additional debug break occurs (not ignored).

**(4) Single-step operation**

When the PSW's SS flag is set (= 1), single-step operation is set, in which case a debug break can occur each time an instruction is executed. Single-step operation is set and canceled via the following steps.

**(a) Steps for setting single-step operation**

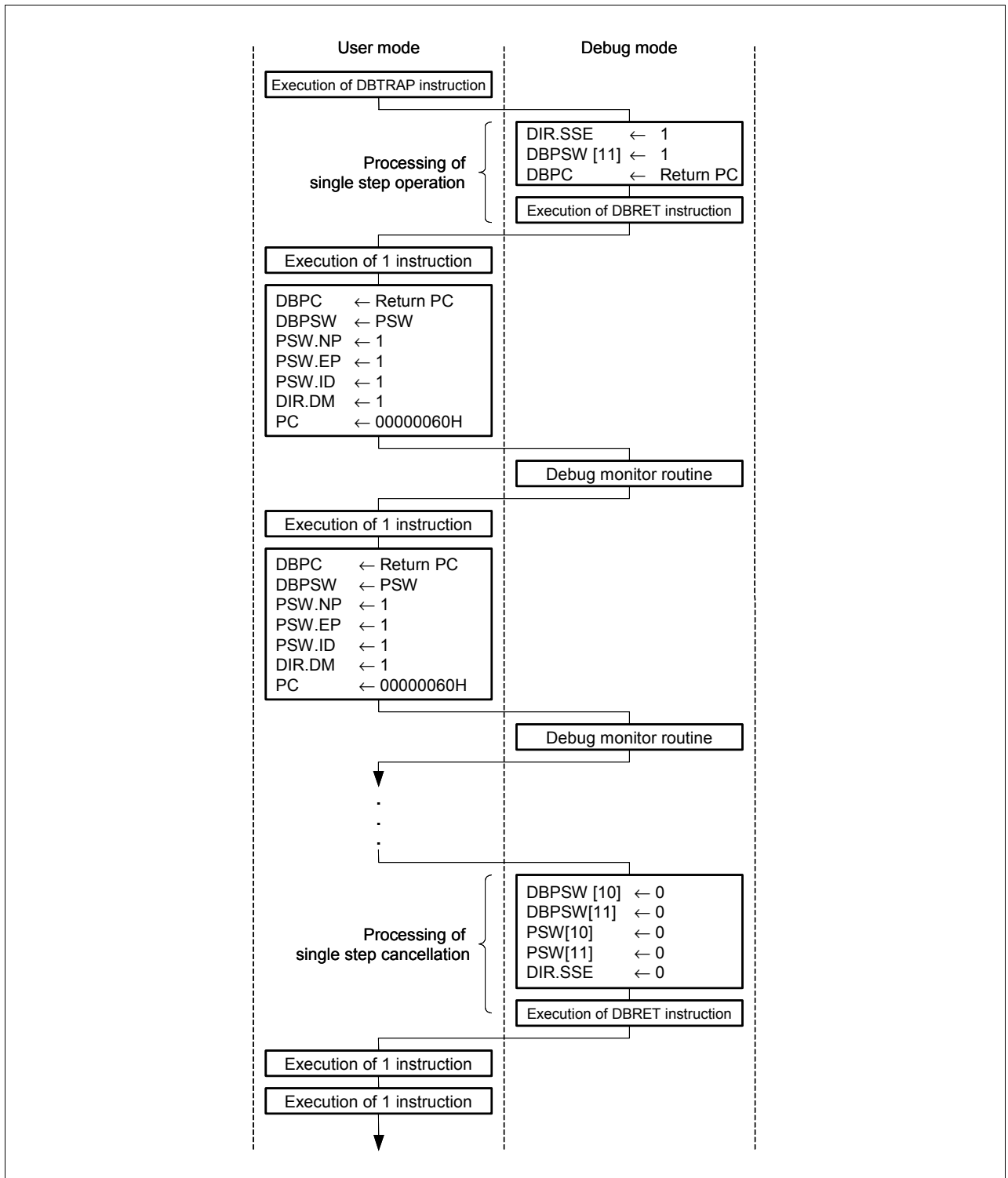
- <1> Use a debug trap (execute the DBTRAP instruction) to shift to debug mode.
- <2> Set a "1" to the SSE bit in the DIR register to control the PSW's SS flag.
- <3> Set a "1" to bit 11 in the DBPSW register to set (= 1) the PSW's SS flag when shifting to user mode.
- <4> Transfer the return PC value to the DBPC register.
- <5> Execute the DBRET instruction to shift to user mode (when shifting, set a "1" to the PSW's SS flag to set single-step operation).

**(b) Steps for canceling single-step operation**

- <1> When operating in the debug mode processing routine, clear bits both 10 (SB bit) and 11 (SS bit) of the DBPSW register to 0.
- <2> When operating in the debug mode processing routine, clear bits both 10 (SB bit) and 11 (SS bit) of the PSW register to 0.
- <3> When operating in the debug mode processing routine, clear bit 8 (SSE bit) of the DIR register to 0.
- <4> Return to the user mode via the DBRET instruction.



Figure 9-1. Single-step Operation Execution Flow

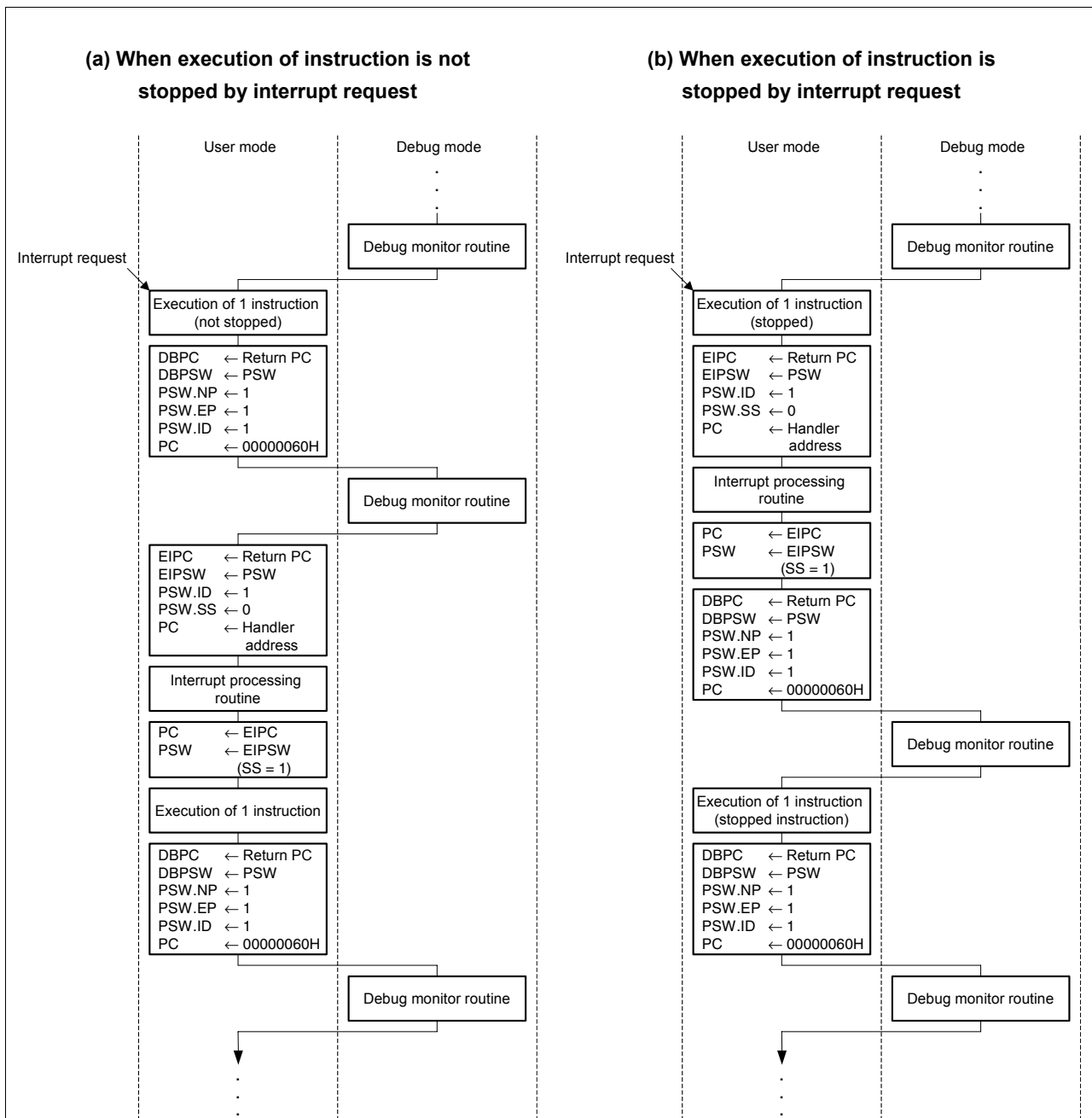


**Remark** During single-step operation, if an interrupt request occurs while in user mode, the setting of the PSW's SB flag is transferred to the SS flag before shifting to the interrupt processing routine. Accordingly, if the SB flag is set (= 1), the interrupt processing routine is performed as a single-step operation. If the SB flag is cleared (= 0), the interrupt processing routine is not performed as a single-step operation (compatible with V850E1 CPU).

The SS flag is set (= 1) again by the processing to return from the interrupt processing routine (EIPSW → PSW) (regardless of the SB flag's value).

The processing flow differs according to the instruction that is being executed when the interrupt occurs (see **Figure 9-2**).

Figure 9-2. Processing Flow when Interrupt Request Occurs during Single-step Operation



**Remarks 1.** When execution is stopped by an interrupt request (See **Table 6-1 Interrupt/Exception Codes**), interrupt processing is performed without waiting for execution of the instruction to be completed. After returning from the interrupt processing routine, the mode is shifted to debug mode without executing any instructions.

**2.** The above figure shows an example where the PSW's SB flag have been cleared (= 0).

## 9.2 Caution Points

When using channels 0 to 3, if access is performed using bit manipulation instructions, the settings in the BPDVn register differ depending on the address being accessed (n = 0 to 3).

The following shows examples of break condition settings for access addresses corresponding to various access sizes.

**Table 9-2. Break Condition Setting Examples**

Access size (Data example is in parentheses)	Access size <sup>Note 1</sup>	Bus cycle	TY bit of BPCn register	BPAVn register <sup>Note1</sup>	BPDVn register Note2
Word (44332211H)	0H	W	1, 1 (Word access)	0H	44332211H
	1H			1H	
	2H			2H	
	3H			3H	
Half word (2211H)	0H	HW	1, 0 (Half word access)	0H	xxxx2211H
	1H			1H	
	2H			2H	
	3H			3H	
Byte (11H)	0H	B	0, 1 (Byte access)	0H	xxxxxx11H
	1H			1H	
	2H			2H	
	3H			3H	
Bit (11H)	0H	B	0, 1 (Byte access)	0H	xxxxxx11H
	1H			1H	xxxx11xxH
	2H			2H	xx11xxxxH
	3H			3H	11xxxxxxH

**Notes 1.** The value of the lower two bits is shown.

**2.** "x" indicates that the value is masked by the BPDMn register.

**Remarks 1.** W: Word data transfer cycle

HW: Half word data transfer cycle

B: Byte data transfer cycle

**2.** n = 0 to 3

# APPENDIX A LIST OF INSTRUCTION

Table A-1 lists the instruction function in alphabetical order. Table A-2 lists instruction format in numeral order.

**Table A-1. Instruction Function (in Alphabetical Order) (1 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
ADD	reg1, reg2	I	0/1	0/1	0/1	0/1	--	Add: Adds the reg1 word data to the reg2 word data and stores the result in reg2.
ADD	imm5, reg2	II	0/1	0/1	0/1	0/1	--	Add: Adds the 5-bit immediate data, sign-extended to word length, to the reg2 word data, and stores the result in reg2.
ADDI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	--	Add Immediate: Adds the 16-bit immediate data, sign-extended to word length, to the reg1 word data, and stores the result in reg2.
ADF	cccc, reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	--	Conditional addition. Either 1 (if the addition result meets the condition specified in condition code "cccc") or 0 (if the addition result does not meet the condition specified in condition code "cccc") is added to the result of adding the word data in reg2 to the value of reg1, and the final result is stored in reg3.
AND	reg1, reg2	I	--	0	0/1	0/1	--	And: ANDs the reg2 word data with the reg1 word data and stores the result in reg2.
ANDI	imm16, reg1, reg2	VI	--	0	0	0/1	--	And: ANDs the reg1 word data with the 16-bit immediate data, zero-extended to word length, and stores the result in reg2.
Bcond	disp9	III	--	--	--	--	--	Conditional Branching (If Carry): Checks the condition flag specified by an instruction, branches if a condition is met and executes the next instruction if not. The branch destination PC holds the sum of the current PC value and the 9-bit displacement data (=8-bit immediate, shifted by 1 and sign-extended to word length).

Table A-1. Instruction Function (in Alphabetical Order) (2 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
BSH	reg2, reg3	XII	0/1	0	0/1	0/1	--	Half-Word Byte Swap: Executes endian conversion.
BSW	reg2, reg3	XII	0/1	0	0/1	0/1	--	Word Byte Swap: Executes endian conversion.
CALLT	imm6	II	--	--	--	--	--	Call with Table Look Up: Updates PC and transfers the control based on CTBP contents.
CLR1	bit#3, disp16 [reg1]	VIII	--	--	--	0/1	--	Clear Bit: Adds the reg1 data to the 16-bit displacement, sign-extended to word length, to generate a 32-bit address; clears the bit data specified by the bit #3 referenced by the generated address.
CLR1	reg2, [reg1]	IX	--	--	--	0/1	--	Clear Bit: Reads the reg1 data to generate a 32-bit address, and clears the bit specified by the lower 3 bits of the reg2 byte data referenced by the generated address.
CMOV	cccc, reg1, reg2, reg3	XI	--	--	--	--	--	Conditional Move: The reg 1 data is set to reg3 if a condition specified by condition code "cccc" is met and reg2 data are set if not.
CMOV	cccc, imm5, reg2, reg3	XII	--	--	--	--	--	Conditional Move: The 5-immediate data, sign-extended to word length, is set to reg3 if a condition specified by condition code "cccc" is met and the reg2 data is set if not.
CMP	reg1, reg2	I	0/1	0/1	0/1	0/1	--	Compare: Subtracts the reg1 word data from the reg2 word data for reg1-reg2 comparison and results via PSW flags.
CMP	imm5, reg2	II	0/1	0/1	0/1	0/1	--	Compare: Subtracts the 5-bit immediate data, sign-extended to word length, from the reg2 word data for imm5-reg2 comparison and results via PSW flags.
CTRET	--	X	0/1	0/1	0/1	0/1	0/1	Return from CALLT: Fetches the return PC and PSW from the appropriate system register and returns from a routine under CALLT.
DBRET	--	X	0/1	0/1	0/1	0/1	0/1	Return from Debug Trap: Fetches the return PC and PSW from the appropriate system register and returns from a debug monitor routine.
DBTRAP	--	I	--	--	--	--	--	Debug Trap: Saves the return PC and PSW in the system register and transfers the control by setting a handler address (00000060H) to PC.
DI	--	X	--	--	--	--	--	Disables Interrupt: Sets "1" to the ID flag of PSW to immediately disable the maskable interrupts acknowledgement; interrupts are disabled beginning this instruction execution.

**Table A-1. Instruction Function (in Alphabetical Order) (3 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
DISPOSE	imm5, list12	XIII	--	--	--	--	--	Dispose: Adds the 5-bit immediate data, logically left-shifted by 2, and zero-extended to word length, to sp; pop the general registers listed in list 12 by adding 4 to sp.
DISPOSE	imm5, list12, [reg1]	XIII	--	--	--	--	--	Dispose. Adds the 5-bit immediate data, logically left-shifted by 2, zero-extended to word length, to sp; pop the general registers listed in list12 by loading data from the address specified by sp and adding 4 to sp; and transfers the control to the address specified by reg1.
DIV	reg1, reg2, reg3	XI	--	0/1	0/1	0/1	--	Divide: Divides the reg2 word data by the reg1 word data and stores the quotient in reg2 and the remainder in reg3.
DIVH	reg1, reg2	I	--	0/1	0/1	0/1	--	Divide Half-Word: Divides the reg2 word data by the reg1 lower half-word data and stores the quotient in reg2.
DIVH	reg1, reg2, reg3	XI	--	0/1	0/1	0/1	--	Divide Half-Word: Divides the reg2 word data by the reg1 lower half-word data and stores the quotient in reg2 and the remainder in reg3.
DIVHU	reg1, reg2, reg3	XI	--	0/1	0/1	0/1	--	Divide Half-Word Unsigned: Divides the reg2 word data by the reg1 lower half-word data and stores the quotient in reg2 and the remainder in reg3.
DIVU	reg1, reg2, reg3	XI	--	0/1	0/1	0/1	--	Divide Unsigned: Divides the reg2 word data by the reg1 word data and stores the quotient in reg2 and the remainder in reg3.
EI	--	X	--	--	--	--	--	Enables Interrupt: Clears the ID flag of PSW to "0" to enable the maskable interrupts acknowledgement beginning the next instruction.
HALT	--	X	--	--	--	--	--	Halt: Stops the CPU operating clock to set CPU in the HALT mode.
HSH	reg2, reg3	XII	0/1	0	0/1	0/1	--	Half word swap of half word data. The value of reg2 is stored in reg3 and the flag judgment result is stored in the PSW.
HSW	reg2, reg3	XII	0/1	0	0/1	0/1	--	Half-Word Swap Word: Executes endian conversion.

Table A-1. Instruction Function (in Alphabetical Order) (4 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
JARL	disp22, reg2	V	--	--	--	--	--	Jump and Register Link: Saves the current PC value+4 in reg2, adds the 22-bit displacement data, sign-extended to word length, to PC, and transfers the control to PC. Bit 0 of 22-bit displacement is masked to "0."
JARL	disp32, reg1	VI	--	--	--	--	--	Jump and Register Link: Saves the current PC value+6 in reg1, adds the 32-bit displacement data to PC, and transfers the control to PC. Bit 0 of 32-bit displacement is masked to "0."
JMP	[reg1]	I	--	--	--	--	--	Jump: Transfers the control to the address specified by reg1. Bit 0 of the address is masked to "0."
JMP	disp32 [reg1]	VI	--	--	--	--	--	Jump: Adds the 32-bit displacement data to reg1, load the result to PC, and transfers the control to PC. Bit 0 of the address is masked to "0."
JR	disp22	V	--	--	--	--	--	Jump Relative: Adds the 22-bit displacement data, sign-extended to word length, to the current PC, and transfer the control to PC. Bit 0 of 22-bit displacement is masked to "0."
JR	disp32	VI	--	--	--	--	--	Jump Relative: Adds the 32-bit displacement data to the current PC and transfer the control to PC. Bit 0 of 32-bit displacement is masked to "0."
LD.B	disp16 [reg1] , reg2	VII	--	--	--	--	--	Load Byte: Adds the reg1 data 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 reg2.
LD.BU	disp16 [reg1] , reg2	VII	--	--	--	--	--	Load Unsigned Byte: Adds the reg1 data 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 reg2.
LD.H	disp16 [reg1], reg2	VII	--	--	--	--	--	Load Half-Word: Adds the reg1 data to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, sign-extended to word length, and stored in reg2.
LD.HU	disp16 [reg1] , reg2	VII	--	--	--	--	--	Load Half-Word Unsigned: Adds the reg1 data to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Half-word data is read from the generated address, zero-extended to word length, and stored in reg2.



**Table A-1. Instruction Function (in Alphabetical Order) (5 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
LD.W	disp16 [reg1] , reg2	VII	--	--	--	--	--	Load Word: Adds the reg1 data to the 16-bit displacement data, sign-extended to word length, to generate a 32-bit address. Word data is read from the generated address, and stored in reg2.
LDSR	reg2, regID	IX	--	--	--	--	--	Load System Register: Sets the reg2 word data to the system register specified by regID. If regID is PSW, the corresponding reg2 bits values are set to the PSW flags.
MAC	reg1, reg2, reg3, reg4	XI	--	--	--	--	--	Multiplication with addition of (signed) word data. The word data in reg2 is multiplied by the word data in reg1 and the result is added to the 64-bit data created by combining reg3 with reg3 + 1, then this sum (64-bit data) is stored in reg4 and reg4 + 1.
MACU	reg1, reg2, reg3, reg4	XI	--	--	--	--	--	Multiplication with addition of (unsigned) word data. The word data in reg2 is multiplied by the word data in reg1 and the result is added to the 64-bit data created by combining reg3 with reg3 + 1, then this sum (64-bit data) is stored in reg4 and reg4 + 1.
MOV	reg1, reg2	I	--	--	--	--	--	Move: Transfers the reg1 word data to reg2.
MOV	imm5, reg2	II	--	--	--	--	--	Move: Transfers the 5-bit immediate data, sign-extended to word length, to reg2.
MOV	imm32, reg1	VI	--	--	--	--	--	Move: Transfers the 32-bit immediate data to reg1.
MOVEA	imm16, reg1, reg2	VI	--	--	--	--	--	Move Effective Address: Adds the 16-bit immediate data, sign-extended to word length, to the reg1 word data and stores the result in reg2.
MOVHI	imm16, reg1, reg2	VI	--	--	--	--	--	Move High Half-word: Adds the word data, where the higher 16 bits are defined by the 16-bit immediate data with the lower 16 bits set to "0," to the reg1 word data and stores the result in reg2.
MUL	reg1, reg2, reg3	XI	--	--	--	--	--	Multiply Word: Multiplies the reg2 word data by the reg1 word data and stores the result in reg2 and reg3 as double-word data.
MUL	imm9, reg2, reg3	XII	--	--	--	--	--	Multiply Word: Multiplies the reg2 word data by the 9-bit immediate data, sign-extended to word length, and stores the result in reg2 and reg3.
MULH	reg1, reg2	I	--	--	--	--	--	Multiply Half-Word: Multiplies the reg2 lower half-word data by the reg1 lower half-word data, and stores the result in reg2 as word data.

Table A-1. Instruction Function (in Alphabetical Order) (6 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
MULH	imm5, reg2	II	--	--	--	--	--	Multiply Half-Word: Multiplies the reg2 lower half-word data by the 5-bit immediate data, sign-extended to half-word length, and stores the result in reg2 as word data.
MULHI	imm16, reg1, reg2	VI	--	--	--	--	--	Multiply Half-word Immediate: Multiplies the reg1 lower half-word data by the 16-bit immediate data and stores the result in reg2.
MULU	reg1, reg2, reg3	XI	--	--	--	--	--	Multiply Word Unsigned: Multiplies the reg2 word data by the reg1 word data and stores the result in reg2 and reg3 as double-word data. reg1 not affected.
MULU	imm9, reg2, reg3	XII	--	--	--	--	--	Multiply Word Unsigned. Multiplies the reg2 word data by the 9-bit immediate data, zero-extended to word length, and store the result in reg2 and reg3.
NOP	--	I	--	--	--	--	--	No Operation.
NOT	reg1, reg2	I	--	0	0/1	0/1	--	Not: Logically negates the reg1 word data by 1's complement and stores the result in reg2.
NOT1	bit#3, disp16 [reg1]	VIII	--	--	--	0/1	--	Not Bit: Adds the reg1 data to the 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Bit specified by the bit #3 is inverted at the byte data location referenced by the generated address.
NOT1	reg2, [reg1]	IX	--	--	--	0/1	--	Not Bit: Reads reg1 to generate a 32-bit address. Bit specified by the lower 3 bits of the reg2 byte data of the generated address is inverted.
OR	reg1, reg2	I	--	0	0/1	0/1	--	Or: ORs the reg2 word data with the reg1 word data and stores the result in reg2.
ORI	imm16, reg1, reg2	VI	--	0	0/1	0/1	--	Or Immediate: ORs the reg1 word data with the 16-bit immediate data, zero-extended to word length and stores the result in reg2.
PREPARE	list12, imm5	XIII	--	--	--	--	--	Prepare: Saves the general register in list12 by subtracting 4 from sp and stores the data in that address; subtracts from sp the 5-bit immediate data, logically left-shifted by 2, zero-extended to word length.
PREPARE	list12, imm5, sp/imm	XIII	--	--	--	--	--	Prepare: Saves the general register in list12 by subtracting 4 from sp and the data in that address; subtract from sp the 5-bit immediate data, logically left-shifted by 2, zero-extended to word length; and loads the data specified by the third operand to ep.

Table A-1. Instruction Function (in Alphabetical Order) (7 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
RETI	--	X	0/1	0/1	0/1	0/1	0/1	Return from Trap or Interrupt: Reads the return PC and PSW from the system register to return from interrupt or exception processing.
SAR	reg1, reg2	IX	0/1	0	0/1	0/1	--	Shift Arithmetic Right: Arithmetically right-shifts the reg2 word data by 'n' positions, where 'n' is specified by the lower 5 bits of reg1 (the pre-shift MSB value is copied and set as the new MSB); writes the result to reg2.
SAR	imm5, reg2	II	0/1	0	0/1	0/1	--	Shift Arithmetic Right. Arithmetically right-shifts the reg2 word data by 'n' positions, where "n" is specified by the lower 5-bit immediate data, zero-extended to word length (the pre-shift MSB value is copied and set as the new MSB); writes the result to reg2.
SAR	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	--	Arithmetic right shift. The word data in reg2 is arithmetically right-shifted the number of times indicated by the lower 5 bits of reg1 (MSB value prior to shifting is copied in MSB-first order) and the result is written to reg3.
SASF	cccc, reg2	IX	--	--	--	--	--	Shift and Set Flag Condition: reg2 is logically left-shifted by 1; "1" is set to LSB if a "cccc"-specified condition is met and "0" is set to LSB if not.
SATADD	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturate Add: Adds the reg1 word data to the reg2 word data and stores the result in reg2; if the result exceeds the maximum positive value, the maximum positive value is stored in reg2 and if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. "1" is set to the SAT flag.
SATADD	imm5, reg2	II	0/1	0/1	0/1	0/1	0/1	Saturate Add: Adds the 5-bit immediate data, sign-extended to word length, to the reg2 word data and stores the result in reg2; if the result exceeds the maximum positive value, the maximum positive value is stored to reg2 and if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. "1" is set to the SAT flag.

Table A-1. Instruction Function (in Alphabetical Order) (8 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SATADD	reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	0/1	<p>Saturation addition.</p> <p>The word data in reg2 is added to the word data in reg1 and the result is stored in reg3. However, if the result exceeds the maximum positive value, the maximum positive value is instead stored in reg3 (or if it exceeds the maximum negative value, the maximum negative value is instead stored in reg3) and the SAT flag is set (= 1).</p>
SATSUB	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	<p>Saturate Subtract: Subtracts the reg1 word data from the reg2 word data and stores the result in reg2; if the result exceeds the maximum positive value, the maximum positive value is stored in reg2 and if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. "1" is set to the SAT flag.</p>
SATSUB	reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	0/1	<p>Saturation multiplication.</p> <p>The word data in reg1 is subtracted from the word data in reg2, and the result is stored in reg3. However, if the result exceeds the maximum positive value, the maximum positive value is instead stored in reg3 (or if it exceeds the maximum negative value, the maximum negative value is instead stored in reg3) and the SAT flag is set (= 1).</p>
SATSUBI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	0/1	<p>Saturate Subtract Immediate: Subtracts the 16-bit immediate data, sign-extended to word length, from the reg1 word data and stores the result in reg2; if the result exceeds the maximum positive value, the maximum positive value is stored in reg2 and if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. "1" is set to the SAT flag.</p>
SATSUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	<p>Saturate Subtract Reverse: Subtracts the reg2 word data from the reg1 word data and stores the result in reg2; if the result exceeds the maximum positive value, the maximum positive value is stored in reg2 and if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. "1" is set to the SAT flag.</p>
SBF	cccc, reg1, reg2, reg3	XI	0/1	0/1	0/1	0/1	--	<p>Conditional subtraction.</p> <p>The value of reg1 is subtracted from the word data in reg2, and either 1 (if the subtraction result meets the condition set by condition code "cccc") or 0 (if the subtraction result does not meet the condition set by condition code "cccc") is subtracted from the result, then the final result is stored in reg3.</p>

Table A-1. Instruction Function (in Alphabetical Order) (9 of 12)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SCH0L	reg2, reg3	IX	0/1	0	0	0/1	--	Search "0" from MSB side. The word data in reg2 is searched starting from the left side (MSB side) and the position where the first "0" bit is found is written in hexadecimal format to reg3. If this "0" bit is not found, "0" is written to reg3 and the Z flag is set (= 1). Lastly, if this "0" bit is found, the CY flag is set (= 1).
SCH0R	reg2, reg3	IX	0/1	0	0	0/1	--	Search zero from LSB side. The word data in reg2 is searched starting from the right side (LSB side) and the position where the first "0" bit is found is written in hexadecimal format to reg3. If this "0" bit is not found, "0" is written to reg3 and the Z flag is set (= 1). Lastly, if this "0" bit is found, the CY flag is set (= 1).
SCH1L	reg2, reg3	IX	0/1	0	0	0/1	--	Search "1" from MSB side. The word data in reg2 is searched starting from the left side (MSB side) and the position where the first "1" bit is found is written in hexadecimal format to reg3. If this "1" bit is not found, "0" is written to reg3 and the Z flag is set (= 1). Lastly, if this "1" bit is found, the CY flag is set (= 1).
SCH1R	reg2, reg3	IX	0/1	0	0	0/1	--	Search "1" from LSB side. The word data in reg2 is searched starting from the left side (LSB side) and the position where the first "1" bit is found is written in hexadecimal format to reg3. If this "1" bit is not found, "0" is written to reg3 and the Z flag is set (= 1). Lastly, if this "1" bit is found, the CY flag is set (= 1).
SET1	bit#3, disp16 [reg1]	VIII	--	--	--	0/1	--	Set Bit: Adds the 16-bit displacement, sign-extended to word length, to the reg1 data to generate a 32-bit address. Bit specified by the bit #3 is set to "1" at the byte data location specified by the generated address.
SET1	reg2, [reg1]	IX	--	--	--	0/1	--	Set Bit: Reads the reg1 data to generate a 32-bit address. Bit specified by the reg2 data of the lower 3 bits is set to "1" at the byte data location referenced by the generated address.
SETF	cccc, reg2	IX	--	--	--	--	--	Set Flag Condition: "1" is set to reg2 if a "cccc"-specified condition is met and "0" is stored in reg2 if not.

**Table A-1. Instruction Function (in Alphabetical Order) (10 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SHL	reg1, reg2	IX	0/1	0	0/1	0/1	--	Shift Logical Left: Logically left-shifts the reg2 word data by 'n' positions, where 'n' is specified by the lower 5 bits of reg1 ("0" is shifted to the LSB side), and writes the result in reg2.
SHL	imm5, reg2	II	0/1	0	0/1	0/1	--	Shift Logical Left: Logically left-shifts the reg2 word data by 'n' positions, where 'n' is specified by the 5-bit immediate data, zero-extended to word length ("0" is shifted to the LSB side) and writes the result in reg2.
SHL	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	--	Logical left shift. The word data in reg2 is logically left-shifted (toward LSB side) the number of times indicated by the lower 5 bits of reg1, and the result is written to reg3.
SHR	reg1, reg2	IX	0/1	0	0/1	0/1	--	Shift Logical Right. Logically right-shifts the reg2 word data by 'n' positions, where 'n' is specified by the lower 5 bits of reg1("0" is shifted to the MSB side), and writes the result in reg2.
SHR	imm5, reg2	II	0/1	0	0/1	0/1	--	Shift Logical Right. Logically right-shifts the reg2 word data by 'n' positions, where 'n' is specified by the 5-bit immediate data, zero-extended to word length ("0" is shifted to the MSB side), and writes the result in reg2.
SHR	reg1, reg2, reg3	XI	0/1	0	0/1	0/1	--	Logical right shift. The word data in reg2 is logically right-shifted (toward MSB side) the number of times indicated by the lower 5 bits of reg1, and the result is written to reg3.
SLD.B	disp7 [ep], reg2	IV	--	--	--	--	--	Byte Load. Adds the 7-bit displacement, 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.
SLD.BU	disp4 [ep], reg2	IV	--	--	--	--	--	Byte Load Unsigned. Adds the 4-bit displacement, 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.
SLD.H	disp8 [ep], reg2	IV	--	--	--	--	--	Half-Word Load. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Half-word data is read from the generated address with bit 0 masked to "0," sign-extended to word length, and stored in reg2.

**Table A-1. Instruction Function (in Alphabetical Order) (11 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SLD.HU	disp5 [ep], reg2	IV	--	--	--	--	--	Half-Word Load Unsigned. Adds the 5-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Half-word data is read from the generated address with bit 0 masked to "0," zero-extended to word length, and stored in reg2.
SLD.W	disp8 [ep], reg2	IV	--	--	--	--	--	Load Word: Adds the 8-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address. Word data is read from the generated address with bits 0 and 1 masked to "0" and stored in reg2.
SST.B	reg2, disp7 [ep]	IV	--	--	--	--	--	Store Byte: Adds the 7-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the reg2 lowest byte data in the generated address.
SST.H	reg2, disp8 [ep]	IV	--	--	--	--	--	Store Half-Word: Adds the 8-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address and stores the reg2 lower half-word in the generated address with bit 0 masked to "0."
SST.W	reg2, disp8 [ep]	IV	--	--	--	--	--	Store Word: Adds the 8-bit displacement data, zero-extended to word length, to the element pointer to generate a 32-bit address and stores the reg2 word data in the generated address with bits 0 and 1 masked to "0."
ST.B	reg2, disp16 [reg1]	VII	--	--	--	--	--	Store Byte: Adds the 16-bit displacement data, sign-extended to word length, to the reg1 data to generate a 32-bit address and stores the reg2 lowest byte data in the generated address.
ST.H	reg2, disp16 [reg1]	VII	--	--	--	--	--	Store Half-Word: Adds the 16-bit displacement data, sign-extended to word length, to the reg1 data to generate a 32-bit address, and stores the reg2 lower half-word in the generated address with bit 0 masked to "0."
ST.W	reg2, disp16 [reg1]	VII	--	--	--	--	--	Store Word: Adds the 16-bit displacement data, sign-extended to word length, to the reg1 data to generate a 32-bit address, and stores the reg2 word data in the generated address with bits 0 and 1 masked to "0."
STSR	regID, reg2	IX	--	--	--	--	--	Store System Register: Stores the system register contents specified by regID in reg2.

**Table A-1. Instruction Function (in Alphabetical Order) (12 of 12)**

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SUB	reg1, reg2	I	0/1	0/1	0/1	0/1	--	Subtract: Subtracts the reg1 word data from the reg2 word data and stores the result in reg2.
SUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	--	Subtract Reverse. Subtracts the reg2 word data from the reg1 word data and stores the result in reg2.
SWITCH	reg1	I	--	--	--	--	--	Jump with Table Look Up: Adds the table entry address (the one next to SWITCH instruction) to the reg1 data, logically left-shifted by 1, to generate a target address; loads the half-word entry data specified by the table entry address; logically left-shifts the loaded data by 1, and sign-extend it to word length, branches to the target address.
SXB	reg1	I	--	--	--	--	--	Sign Extend Byte: Sign-extends the reg1 lowest byte to word length.
SXH	reg1	I	--	--	--	--	--	Sign Extend Half-word: Sign-extends the reg1 lower half-word to word length.
TRAP	vector	X	--	--	--	--	--	Trap: Saves the return PC and PSW; sets the exception code and the PSW flags; jumps to the trap handler address corresponding to the vector-specified trap vector to begin exception processing.
TST	reg1, reg2	I	--	0	0/1	0/1	--	Test: ANDs the reg2 word data with the reg1 word data. The result is not stored, changing the flags only.
TST1	bit#3, disp16 [reg1]	VIII	--	--	--	0/1	--	Test Bit: Adds the reg1 data to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address; checks the bit specified by the bit #3 at the byte data location referenced by the generated address. "1" is set to Z flag if the specified bit =0 and Z flag cleared to "0" if the bit=1.
TST1	reg2, [reg1]	IX	--	--	--	0/1	--	Test Bit: Reads the reg1 data to generate a 32-bit address. "1" is set to the Z flag If the bits indicated by the lower 3 bits of the reg2 byte data of the generated address are 0, and the Z flag is cleared to "0" if they are 1.
XOR	reg1, reg2	I	--	0	0/1	0/1	--	Exclusive Or: Exclusively ORs the reg2 word data with the reg1 word data and stores the result in reg2.
XORI	imm16, reg1, reg2	VI	--	0	0/1	0/1	--	Exclusive Or Immediate: Exclusively ORs the reg1 word data with a 16-bit immediate data, zero-extended to word length, and stores the result in reg2.
ZXB	reg1	I	--	--	--	--	--	Zero Extend Byte: Zero-extends the reg1 lowest byte to word length.
ZXH	reg1	I	--	--	--	--	--	Zero Extend Half-word. Zero-extends the reg1 lower half-word to word length.



Table A-2. Instruction Format (in Numeral Order) (1 of 4)

Format	Opcode				Mnemonic	Operand
	15	0	31	16		
I	0000000000000000		--		NOP	--
	rrrrrr000000RRRRR		--		MOV	reg1, reg2
	rrrrrr000001RRRRR		--		NOT	reg1, reg2
	rrrrrr000010RRRRR		--		DIVH	reg1, reg2
	00000000010RRRRR		--		SWITCH	reg1
	00000000011RRRRR		--		JMP	[reg1]
	rrrrrr000100RRRRR		--		SATSUBR	reg1, reg2
	rrrrrr000101RRRRR		--		SATSUB	reg1, reg2
	rrrrrr000110RRRRR		--		SATADD	reg1, reg2
	rrrrrr000111RRRRR		--		MULH	reg1, reg2
	00000000100RRRRR		--		ZXB	reg1
	00000000101RRRRR		--		SXB	reg1
	00000000110RRRRR		--		ZXH	reg1
	00000000111RRRRR		--		SXH	reg1
	rrrrrr001000RRRRR		--		OR	reg1, reg2
	rrrrrr001001RRRRR		--		XOR	reg1, reg2
	rrrrrr001010RRRRR		--		AND	reg1, reg2
	rrrrrr001011RRRRR		--		TST	reg1, reg2
	rrrrrr001100RRRRR		--		SUBR	reg1, reg2
	rrrrrr001101RRRRR		--		SUB	reg1, reg2
	rrrrrr001110RRRRR		--		ADD	reg1, reg2
	rrrrrr001111RRRRR		--		CMP	reg1, reg2
	1111100001000000		--		DBTRAP	--
II	rrrrrr010000iiii		--		MOV	imm5, reg2
	rrrrrr010001iiii		--		SATADD	imm5, reg2
	rrrrrr010010iiii		--		ADD	imm5, reg2
	rrrrrr010011iiii		--		CMP	imm5, reg2
	0000001000iiii		--		CALLT	imm6
	rrrrrr010100iiii		--		SHR	imm5, reg2
	rrrrrr010101iiii		--		SAR	imm5, reg2
	rrrrrr010110iiii		--		SHL	imm5, reg2
	rrrrrr010111iiii		--		MULH	imm5, reg2
III	dddd1011dddCCCC		--		Bcond	disp9
IV	rrrrrr0000110dddd		--		SLD.BU	disp4 [ep], reg2
	rrrrrr0000111dddd		--		SLD.HU	disp5 [ep], reg2
	rrrrrr0110ddddddd		--		SLD.B	disp7 [ep], reg2
	rrrrrr0111ddddddd		--		SST.B	reg2, disp7 [ep]
	rrrrrr1000ddddddd		--		SLD.H	disp8 [ep], reg2
	rrrrrr1001ddddddd		--		SST.H	reg2, disp8 [ep]
	rrrrrr1010ddddddd0		--		SLD.W	disp8 [ep], reg2

Table A-2. Instruction Format (in Numeral Order) (2 of 4)

Format	Opcode				Mnemonic	Operand
	15	0	31	16		
IV	rrrrr1010	dddddd1		--	SST.W	reg2, disp8 [ep]
V	rrrrr11110	dddddd	dddddddddddddd0		JARL	disp22, reg2
	0000011110	dddddd	dddddddddddddd0		JR	disp22
VI	00000010111	00000		<b>Note1</b>	JR	disp32
	00000010111	RRRRR		<b>Note1</b>	JARL	disp32, reg1
	rrrrr110000	RRRRR	iiiiiiiiiiiiiiii		ADDI	imm16, reg1, reg2
	rrrrr110001	RRRRR	iiiiiiiiiiiiiiii		MOVEA	imm16, reg1, reg2
	rrrrr110010	RRRRR	iiiiiiiiiiiiiiii		MOVHI	imm16, reg1, reg2
	rrrrr110011	RRRRR	iiiiiiiiiiiiiiii		SATSUBI	imm16, reg1, reg2
	00000110001	RRRRR		<b>Note2</b>	MOV	imm32, reg1
	rrrrr110100	RRRRR	iiiiiiiiiiiiiiii		ORI	imm16, reg1, reg2
	rrrrr110101	RRRRR	iiiiiiiiiiiiiiii		XORI	imm16, reg1, reg2
	rrrrr110110	RRRRR	iiiiiiiiiiiiiiii		ANDI	imm16, reg1, reg2
	rrrrr110111	RRRRR	iiiiiiiiiiiiiiii		MULHI	imm16, reg1, reg2
	00000110111	RRRRR		<b>Note1</b>	JMP	disp32 [reg1]
VII	rrrrr111000	RRRRR	dddddddddddddd		LD.B	disp16 [reg1], reg2
	rrrrr111001	RRRRR	dddddddddddddd0		LD.H	disp16 [reg1], reg2
	rrrrr111001	RRRRR	dddddddddddddd1		LD.W	disp16 [reg1], reg2
	rrrrr111010	RRRRR	dddddddddddddd		ST.B	reg2, disp16 [reg1]
	rrrrr111011	RRRRR	dddddddddddddd0		ST.H	reg2, disp16 [reg1]
	rrrrr111011	RRRRR	dddddddddddddd1		ST.W	reg2, disp16 [reg1]
	rrrrr11110b	RRRRR	dddddddddddddd1		LD.BU	disp16 [reg1], reg2
	rrrrr111111	RRRRR	dddddddddddddd1		LD.HU	disp16 [reg1], reg2
VIII	00bbb111110	RRRRR	dddddddddddddd		SET1	bit#3, disp16 [reg1]
	01bbb111110	RRRRR	dddddddddddddd		NOT1	bit#3, disp16 [reg1]
	10bbb111110	RRRRR	dddddddddddddd		CLR1	bit#3, disp16 [reg1]
	11bbb111110	RRRRR	dddddddddddddd		TST1	bit#3, disp16 [reg1]

**Notes1.** 32-bit displacement data. The higher 32 bits (bits 16 to 47) are as follows.

31	16	47	32
dddddddddddddd0	DDDDDDDDDDDDDDDD		

**2.** 32-bit immediate data. The higher 32 bits (bits 16 to 47) are as follows.

31	16	47	32
iiiiiiiiiiiiiiii	IIIIIIIIIIIIIIII		

Table A-2. Instruction Format (in Numeral Order) (3 of 4)

Format	Opcode				Mnemonic	Operand
	15	0	31	16		
IX	rrrrr111110cccc		0000000000000000		SETF	cccc, reg2
	rrrrr11111RRRRR		0000000000100000		LDSR	reg2, regID
	rrrrr11111RRRRR		0000000001000000		STSR	regID, reg2
	rrrrr11111RRRRR		0000000010000000		SHR	reg1, reg2
	rrrrr11111RRRRR		0000000010100000		SAR	reg1, reg2
	rrrrr11111RRRRR		0000000011000000		SHL	reg1, reg2
	rrrrr11111RRRRR		0000000011100000		SET1	reg2, [reg1]
	rrrrr11111RRRRR		0000000011100010		NOT1	reg2, [reg1]
	rrrrr11111RRRRR		0000000011100100		CLR1	reg2, [reg1]
	rrrrr11111RRRRR		0000000011100110		TST1	reg2, [reg1]
	rrrrr111110cccc		0000001000000000		SASF	cccc, reg2
	rrrrr1111100000	wwwww01101100000			SCH0R	reg2, reg3
	rrrrr1111100000	wwwww01101100010			SCH1R	reg2, reg3
	rrrrr1111100000	wwwww01101100100			SCH0L	reg2, reg3
	rrrrr1111100000	wwwww01101100110			SCH1L	reg2, reg3
X	0000011111iiii		0000000100000000		TRAP	vector
	000001111100000		0000000100100000		HALT	--
	000001111100000		0000000101000000		RETI	--
	000001111100000		0000000101000100		CTRET	--
	000001111100000		0000000101000110		DBRET	--
	000001111100000		0000000101100000		DI	--
	100001111100000		0000000101100000		EI	--
XI	rrrrr11111RRRRR	wwwww00010000010			SHR	reg1, reg2, reg3
	rrrrr11111RRRRR	wwwww00010100010			SAR	reg1, reg2, reg3
	rrrrr11111RRRRR	wwwww00011000010			SHL	reg1, reg2, reg3
	rrrrr11111RRRRR	wwwww01000100000			MUL	reg1, reg2, reg3
	rrrrr11111RRRRR	wwwww01000100010			MULU	reg1, reg2, reg3
	rrrrr11111RRRRR	wwwww01010000000			DIVH	reg1, reg2, reg3

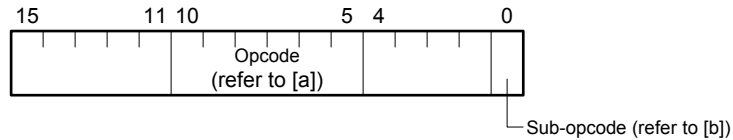
**Table A-2. Instruction Format (in Numeral Order) (4 of 4)**

Format	Opcode				Mnemonic	Operand
	15	0	31	16		
XI	rrrrr11111RRRRR		wwwww01010000010		DIVHU	reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww01011000000		DIV	reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww01011000010		DIVU	reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww011001cccc0		CMOV	cccc, reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww011100cccc0		SBF	cccc, reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww01110011010		SATSUB	reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww011101cccc0		ADF	cccc, reg1, reg2, reg3
	rrrrr11111RRRRR		wwwww01110111010		SATADD	reg1, reg2, reg3
	rrrrr11111RRRRR		www0011110mmmm0		MAC	reg1, reg2, reg3, reg4
	rrrrr11111RRRRR		www0011111mmmm0		MACU	reg1, reg2, reg3, reg4
XII	rrrrr11111iiiiL		wwwww01001IIII00		MUL	imm9, reg2, reg3
	rrrrr11111iiiiL		wwwww01001IIII10		MULU	imm9, reg2, reg3
	rrrrr11111iiiiL		wwwww011000cccc0		CMOV	cccc, imm5, reg2, reg3
	rrrrr1111100000		wwwww01101000000		BSW	reg2, reg3
	rrrrr1111100000		wwwww01101000010		BSH	reg2, reg3
	rrrrr1111100000		wwwww01101000100		HSW	reg2, reg3
	rrrrr1111100000		wwwww01101000110		HSH	reg2, reg3
XIII	0000011001iiiiL		LLLLLLLLLLLLRRRRR		DISPOSE	imm5, list12, [reg1]
	0000011001iiiiL		LLLLLLLLLLLL00000		DISPOSE	imm5, list12
	0000011110iiiiL		LLLLLLLLLLLL00001		PREPARE	list12, imm5
	0000011110iiiiL		LLLLLLLLLLLLff011		PREPARE	list12, imm5, sp/imm

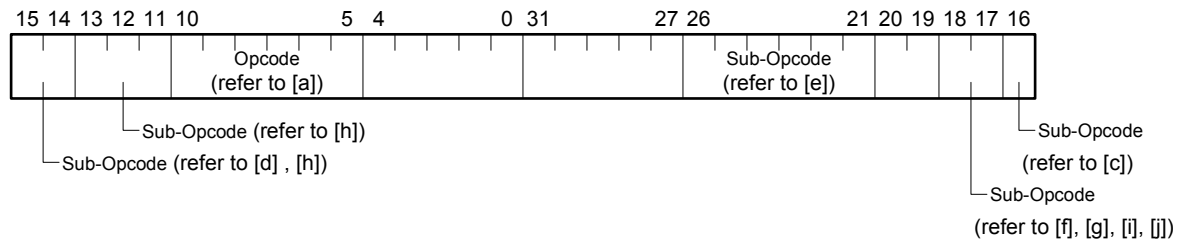
## APPENDIX B INSTRUCTION OPCODE MAP

The instruction opcode map is as follows:

### (1) 16-bit format instruction



### (2) 32-bit format instruction



### Operand convention

Symbol	Meaning
R	reg1: General register (as source register).
r	reg2: General register (Primarily as destination register with some used as source registers).
w	reg3: General register (Primarily to store the remainder of division results or the higher 32 bits of multiply results).
bit#3	3-bit data to specify bit number.
imm×	×-bit immediate data.
disp×	×-bit displacement data.
cccc	4-bit data to specify condition code.

[a] Opcode

Bit10	Bit9	Bit8	Bit7	Bits 6, 5				Format
				0,0	0,1	1,0	1,1	
0	0	0	0	MOV R, r NOP <sup>Note1</sup>	NOT	DIVH SWITCH <sup>Note2</sup> DBTRAP <sup>Note3</sup> Undefined <sup>Note3</sup>	JMP [reg1] <sup>Note4</sup> SLD.BU <sup>Note5</sup> SLD.HU <sup>Note6</sup>	I, IV
0	0	0	1	SATSUBR ZXB <sup>Note4</sup>	SATSUB SXB <sup>Note4</sup>	SATADD R, r ZXH <sup>Note4</sup>	MULH SXH <sup>Note4</sup>	I
0	0	1	0	OR	XOR	AND	TST	
0	0	1	1	SUBR	SUB	ADD R, r	CMP R, r	
0	1	0	0	MOV imm5, r ----- CALLT <sup>Note4</sup>	SATADD imm5, r	ADD imm5, r	CMP imm5, r	II, VI
0	1	0	1	SHR imm5, r	SAR imm5, r	SHL imm5, r	MULH imm5, r JR disp32 <sup>Note1</sup> JARL disp32, reg1 <sup>Note2</sup>	
0	1	1	0	SLD.B				IV
0	1	1	1	SST.B				
1	0	0	0	SLD.H				
1	0	0	1	SST.H				
1	0	1	0	SLD.W <sup>Note7</sup> SST.W <sup>Note7</sup>				
1	0	1	1	Bcond				III
1	1	0	0	ADDI	MOVEA MOV imm32, R <sup>Note4</sup>	MOVHI ----- DISPOSE <sup>Note4</sup>	SATSUBI	VI, XIII
1	1	0	1	ORI	XORI	ANDI	MULHI JMP disp32 [reg1] <sup>Note4</sup>	
1	1	1	0	LD.B	LD.H <sup>Note8</sup> LD.W <sup>Note8</sup>	ST.B	ST.H <sup>Note8</sup> ST.W <sup>Note8</sup>	VII
1	1	1	1	JR disp22 JARL disp22, reg2 LD.BU <sup>Note10</sup> PREPARE <sup>Note11</sup>		Bit manipulation 1 <sup>Note9</sup>	LD.HU <sup>Note10</sup> Undefined <sup>Note11</sup> Expansion 1 <sup>Note12</sup>	V, VII, VIII, IX, X, XI, XII, XIII,

- Notes**
1. If R (reg1) = r0 and r (reg2) = r0 (instruction without reg1 and reg2).
  2. If R (reg1) ≠ r0 and r (reg2) = r0 (instruction with reg1 and without reg2).
  3. If R (reg1) = r0 and r (reg2) ≠ r0 (instruction without reg1 and with reg2).
  4. If r (reg2) = r0 (instruction without reg2).
  5. If bit 4 = 0 and r (reg2) ≠ r0 (instruction with reg2).
  6. If bit 4 = 1 and r (reg2) ≠ r0 (instruction with reg2).
  7. Refer to [b].
  8. Refer to [c].
  9. Refer to [d].

- Notes**
10. If bit 16 = 1 and r (reg2)  $\neq$  r0 (instruction with reg2).
  11. If bit 16 = 1 and r (reg2) = r0 (instruction without reg2).
  12. Refer to [e].

[b] Short format load/store instructions (displacement/sub-opcode)

Bit10	Bit9	Bit8	Bit7	Bit 0	
				0	1
0	1	1	0	SLD.B	
0	1	1	1	SST.B	
1	0	0	0	SLD.H	
1	0	0	1	SST.H	
1	0	1	0	SLD.W	SST.W

[c] Load/store instructions (displacement/sub-opcode)

Bit 6	Bit 5	Bit 16	
		0	1
0	0	LD.B	
0	1	LD.H	LD.W
1	0	ST.B	
1	1	ST.H	ST.W

[d] Bit manipulation instructions 1 (sub-opcode)

Bit 15	Bit 14	
	0	1
0	SET1 bit#3, disp16 [R]	NOT1 bit#3, disp16 [R]
1	CLR1 bit#3, disp16 [R]	TST1 bit#3, disp16 [R]

[e] Expansion 1 (sub-opcode)

Bit26	Bit25	Bit24	Bit23	Bits: 22 and 21				Format
				0,0	0,1	1,0	1,1	
0	0	0	0	SETF	LDSR	STSR	Undefined	IX
0	0	0	1	SHR	SAR	SHL	Bit manipulation 2 Note1	
0	0	1	0	TRAP	HALT	RETI <sup>Note2</sup> CTRET <sup>Note2</sup> DBRET <sup>Note2</sup> Undefined	EI <sup>Note3</sup> DI <sup>Note3</sup> Undefined	X
0	0	1	1	Undefined		Undefined		--
0	1	0	0	SASF	MUL R, r, w MULU R, r, w <sup>Note4</sup>	MUL imm9, r, w MULU imm9, r, w <sup>Note4</sup>		IX, XI, XII
0	1	0	1	DIVH DIVHU <sup>Note4</sup>	DIV DIVU <sup>Note4</sup>			XI
0	1	1	0	CMOV cccc, imm5, r, w	CMOV cccc, R, r, w	BSW <sup>Note5</sup> BSH <sup>Note5</sup> HSW <sup>Note5</sup> HSH <sup>Note5</sup>	SCH0R <sup>Note6</sup> SCH1R <sup>Note6</sup> SCH0L <sup>Note6</sup> SCH1L <sup>Note6</sup>	IX, XI, XII
0	1	1	1	SBF SATSUB R, r, w <sup>Note7</sup>	ADF SATADD R, r, w <sup>Note7</sup>	MAC	MACU	XI
1	x	x	x	Illegal instruction				--

- Notes**
1. Refer to [f].
  2. Refer to [g].
  3. Refer to [h].
  4. If bit 17 = 1.
  5. Refer to [i].
  6. Refer to [j].
  7. If bit 20 to 17 = 1, 1, 0, 1.

[f] Bit manipulation instructions 2 (sub-opcode)

Bit 18	Bit 17	
	0	1
0	SET1 r, [R]	NOT1 r, [R]
1	CLR1 r, [R]	TST1 r, [R]

[g] Return instructions (sub-opcode)

Bit 18	Bit 17	
	0	1
0	RETI	Undefined
1	CTRET	DBRET



[h] PSW operation instructions (sub-opcode)

Bit 15	Bit 14	Bits: 13, 12, and 11							
		0,0,0	0,0,1	0,1,0	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
0	0	DI	Undefined						
0	1	Undefined							
1	0	EI	Undefined						
1	1	Undefined							

[i] Endian conversion instructions (sub-opcode)

Bit 18	Bit 17	
	0	1
0	BSW	BSH
1	HSW	HSH

[j] Bit search instructions (sub-opcode)

Bit 18	Bit 17	
	0	1
0	SCH0R	SCH1R
1	SCH0L	SCH1L

## APPENDIX C DIFFERENCES WITH ARCHITECTURE OF V850 CPU AND V850E1 CPU

(1/3)

Item		V850E2 CPU	V850E1 CPU	V850 CPU
Instructions (Operand included)	ADF cccc, reg1, reg2, reg3	Provided	Not provided	Not 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 reg2, reg3			
	SCH0R reg2, reg3			
	SCH1L reg2, reg3			
	SCH1R reg2, reg3			
	SHL reg1, reg2, reg3			
	SHR reg1, reg2, reg3			
	BSH reg2, reg3			
	BSW reg2, reg3			
	CALLT imm6			
	CLR1 reg2, [reg1]			
	CMOV cccc, imm5, reg2, reg3			
	CMOV cccc, reg1, reg2, reg3			
	CTRET			
	DBRET			
	DBTRAP			
	DISPOSE imm5, list12			
	DISPOSE imm5, list12 [reg1]			
	DIV reg1, reg2, reg3			
	DIVH reg1, reg2, reg3			
	DIVHU reg1, reg2, reg3			
	DIVU reg1, reg2, reg3			
	HSW reg2, reg3			
			Provided	

(2/3)

Item		V850E2 CPU	V850E1 CPU	V850 CPU
Instructions (Operand included)	LD.BU disp16 [reg1], reg2	Provided		Not provided
	LD.HU disp16 [reg1], reg2			
	MOV imm32, reg1			
	MUL imm9, reg2, reg3			
	MUL reg1, reg2, reg3			
	MULU reg1, reg2, reg3			
	MULU imm9, reg2, reg3			
	NOT1 reg2, [reg1]			
	PREPARE list12, imm5			
	PREPARE list12, imm5, sp/imm			
	SASF cccc, reg2			
	SET1 reg2, [reg1]			
	SLD.BU disp4 [ep], reg2			
	SLD.HU disp5 [ep], reg2			
	SWITCH reg1			
	SXB reg1			
	SXH reg1			
	TST1 reg2, [reg1]			
	ZXB reg1			
	ZXH reg1			
Instruction format	Format IV	Format varies for some instructions.		
	Format XI	Provided		Not provided
	Format XII			
	Format XIII			
Instruction execution clocks		Value varies for some instructions.		
Program space		512M bytes	64M bytes	16M bytes
Valid bits of program counter (PC)		Lower 29 bits	Lower 26 bits	Lower 24 bits
System register	Program Status Word (PSW)	Functions differ.		
	CALLT execution status-saving registers (CTPC, CTPSW)	Provided		Not provided
	Exception/debug trap status-saving registers (DBPC, DBPSW)			
	CALLT base pointer (CTBP)			

(3/3)

Item		V850E2 CPU	V850E1 CPU	V850 CPU
System register	Debug-interface register (DIR)	Provided (Functions differ.)		
	Breakpoint control registers 0 and 1 (BPC0, BPC1)			
	Program ID register (ASID)	Provided		
	Breakpoint address setting registers 0 and 1 (BPAV0, BPAV1)	Provided (Initial values differ.)		
	Breakpoint address mask registers 0 and 1 (BPAM0, BPAM1)			
	Breakpoint data setting registers 0 and 1 (BPDV0, BPDV1)	Provided		
	Breakpoint data mask registers 0 and 1 (BPDM0, BPDM1)			
	Breakpoint control register 2 to 6 (BPC2 to BPC6)	Provided	Not provided	
	Breakpoint address setting register 2, 3 (BPAV2, BPAV3)			
	Breakpoint address mask register 2,3 (BPAM2, BPAM3)			
	Breakpoint data setting register 2 to 5 (BPDV2 to BPDV5)			
	Breakpoint data mask register 2, 3 (BPDM2, BPDM3)			
	Exception trap status-saving registers	DBPC, DBPSW		EIPC, EIPSW
Illegal instruction code		Instruction code areas vary.		
Misaligned access enable/disable setting		Can be set depending on the product.		Cannot be set (misaligned access disabled).
Non-maskable interrupt (NMI)	Input	3		1
	Exception code	0010H, 0020H, 0030H		0010H
	Handler address	00000010H, 00000020H, 00000030H		00000010H
Debug trap		Provided		Not provided
Debug break		Provided (4 factors)	Provided (2 factors) <sup>Note</sup>	Not provided
Pipeline		7 steps	5 steps	
		Pipeline flow differs for each instruction.		

**Note** An equivalent function can be performed by BPC0, BPC1 register setup.

## APPENDIX D INSTRUCTIONS ADDED FOR V850E2 CPU COMPARED WITH V850 CPU AND V850E1 CPU

V850E2 differs from its predecessors in that (1) V850E2 CPU instruction codes are upward-compatible at the object code level and (2) instructions, such as those requiring the r0 register write, are extended as additional instructions. Table D-1 compares the V850E2 CPU instructions with the V850 CPU instructions, Table D-2 compares the V850E2 CPU instructions with the V850E1 CPU instructions, listing the new additions to V850E2 CPU in the left column and the existing V850 CPU and V850E1 CPU instructions in the right column.

**Table D-1. Instructions Added to V850E2 CPU and Compatible V850 CPU Instructions (1 of 2)**

Instructions Added to V850E2CPU	Compatible V850 CPU Instructions
CALLT imm6	MOV imm5, r0 or SATADD imm5, r0
DISPOSE imm5, list12	MOVHI imm16, reg1, r0 or SATSUBI imm16, reg1, r0
DISPOSE imm5, list12 [reg1]	MOVHI imm16, reg1, r0 or SATSUBI imm16, reg1, r0
MOV imm32, reg1	MOVEA imm16, reg1, r0
SWITCH reg1	DIVH reg1, r0
SXB reg1	SATSUB reg1, r0
SXH reg1	MULH reg1, r0
ZXB reg1	SATSUBR reg1, r0
ZXH reg1	SATADD reg1, r0
JARL disp32, reg1	MULH imm5, r0
JMP disp32 [reg1]	MULHI imm16, reg1, r0
JR disp32	MULH 0x0, r0
ADF cccc, reg1, reg2, reg3	Illegal instruction
BSH reg2, reg3	
BSW reg2, reg3	
CMOV cccc, imm5, reg2, reg3	
CMOV cccc, reg1, reg2, reg3	
CTRET	
DIV reg1, reg2, reg3	
DIVH reg1, reg2, reg3	
DIVHU reg1, reg2, reg3	
DIVU reg1, reg2, reg3	
HSH reg2, reg3	
HSW reg2, reg3	
MAC reg1, reg2, reg3, reg4	
MACU reg1, reg2, reg3, reg4	
MUL imm9, reg2, reg3	

**Table D-1. Instructions Added to V850E2 CPU and Compatible V850 CPU Instructions (2 of 2)**

Instructions Added to V850E2CPU	Compatible V850 CPU Instructions
MUL reg1, reg2, reg3	Illegal instruction
MULU reg1, reg2, reg3	
MULU imm9, reg2, reg3	
SASF cccc, reg2	
SATADD reg1, reg2, reg3	
SATSUB reg1, reg2, reg3	
SBF cccc, reg1, reg2, reg3	
SCH0L reg2, reg3	
SCH0R reg2, reg3	
SCH1L reg2, reg3	
SCH1R reg2, reg3	
CLR1 reg2, [reg1]	Undefined
DBRET	
DBTRAP	
LD.BU disp16 [reg1], reg2	
LD.HU disp16 [reg1], reg2	
NOT1 reg2, [reg1]	
PREPARE list12, imm5	
PREPARE list12, imm5, sp/imm	
SAR reg1, reg2, reg3	
SET1 reg2, [reg1]	
SHL reg1, reg2, reg3	
SHR reg1, reg2, reg3	
SLD.BU disp4 [ep], reg2	
SLD.HU disp5 [ep], reg2	
TST1 reg2, [reg1]	

**Table D-2. Instructions Added to V850E2 CPU and Compatible V850E1 CPU Instructions**

Instructions Added to V850E2CPU	Compatible V850E1 CPU Instructions
ADF cccc, reg1, reg2, reg3	Illegal instruction
SATADD reg1, reg2, reg3	
SATSUB reg1, reg2, reg3	
SBF cccc, reg1, reg2, reg3	
HSH reg2, reg3	Undefined
JARL disp32, reg1	
JMP disp32 [reg1]	
JR disp32	
MACU reg1, reg2, reg3, reg4	
SAR reg1, reg2, reg3	
SCH0L reg2, reg3	
SCH0R reg2, reg3	
SCH1L reg2, reg3	
SCH1R reg2, reg3	
SHL reg1, reg2, reg3	
SHR reg1, reg2, reg3	
MAC reg1, reg2, reg3, reg4	
	Undefined, or a part of product-sum operation instructions

## APPENDIX E LIST OF CAUTION POINTS

(1/4)

Location		Description
2. 2 System Registers		After bit 0 in the EIPC, FEPC, or CTPC register is set (= 1) by the LDSR instruction, if interrupt processing occurs and a RETI instruction is used to recover, the value of bit 0 is ignored (since the PC bit 0 value is fixed to 0). When setting values to the EIPC, FEPC, or CTPC register, always set an even number (bit 0 = 0) unless there is a particular reason to do otherwise.
2. 2. 8 Debug Interface register (DIR)		Either one of the DIR register's SQ1 and RE1 bits must be set (= 1) or both must be cleared (= 0). Operation is not guaranteed if both are set (= 1).
2. 2. 9 Breakpoint Control registers 0 to 3 (BPC0 to BPC3)		Be sure to zero-clear bits 31 to 27, 14 to 12, 6, and 5 in the BPCn register (n = 0 to 3). Operation is not guaranteed if any of these bits are set (= 1).
		Only "0" can be written to bits FB2 to FB0 in the BPCn register (n = 0 to 3). To update the values of these bits, clear the bits to zero. Operation is not guaranteed if any of these bits are set (= 1).
5. 1 Instruction Formats		Some instructions have an unused field (RFU). This field is reserved for future expansion only and must be fixed to "0."
5. 3 Instruction Set	Bcond	The branch condition loses its meaning If a conditional branch instruction is executed on a signed integer (BGE, BGT, BLE, or BLT) when the saturate instruction sets "1" to the SAT flag. In normal operations, if an overflow occurs, the S flag is inverted (0 → 1 or 1 → 0). This is because the result is a negative value if it exceeds the maximum positive value and it is a positive value if it exceeds the maximum negative value. However, when a saturate 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.
	CALLT	When an interrupt occurs during the CALLT instruction execution, the execution is aborted after the end of the read/write cycle.
	DBRET DBTRAP	Because this instruction is for debugging, it is essentially used by debug tools. When a debug tool is using this instruction, therefore, use of it in the application may cause a malfunction.
	DISPOSE	When an interrupt occurs during the DISPOSE instruction execution, the execution is aborted after the end of the read/write cycle, due to stack operation.
	LD.H LD.HU	Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting: <ul style="list-style-type: none"> <li>•Bit0 is masked to "0" and address is generated (when misaligned access is disabled).</li> <li>•Bit0 is not masked and address is generated (when misaligned access is enabled).</li> </ul> For details on misaligned access, see 3.3 Data Alignment.



Location		Description
5. 3 Instruction Set	LD.W	<p>Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>• Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).</li> <li>• Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see 3.3 Data Alignment.</p>
	LDSR	<p>The fields to define reg1 and reg2 are swapped in this instruction. "RRR" is normally used for reg1 and is the source operand with "rrr" representing reg2 and the destination operand. In this instruction, "RRR" is still the source operand, but is represented by reg2 with "rrr" being as the register destination, as labeled below:</p> <p>rrrrr: reg1D specification RRRRR: reg2 specification</p> <p>The system register number reg1D is to identify a system register. Accessing system registers that are reserved or write-prohibited is prohibited.</p>
	MAC MACU	The 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.
	MUL MULU	<p>In the "MUL reg1, reg2, reg3", "MULU reg1, reg2, reg3" instruction, do not use registers in combinations that satisfy all the following conditions. If the instruction is executed with all the following conditions satisfied, the operation is not guaranteed.</p> <ul style="list-style-type: none"> <li>• reg1 = reg3</li> <li>• reg1 ≠ reg2</li> <li>• reg1 ≠ r0</li> <li>• reg3 ≠ r0</li> </ul>
	PREPARE	When an interrupt occurs during execution of an instruction, the stack operation may cause execution of the instruction to be stopped after the read/write cycles and register overwrite operations have been completed.
	RETI	<p>In order to correctly restore the PC and PSW values when using a RETI instruction to recover from non-maskable interrupt processing or software exception processing, the NP and EP flags must be set as follows before executing the RETI instruction.</p> <ul style="list-style-type: none"> <li>• When using RETI instruction to recover from non-maskable interrupt processing: NP = 1 and EP = 0</li> <li>• When using RETI instruction to recover from software exception processing: EP = 1</li> </ul> <p>The LDSR instruction is used for program-based settings. Due to the operation of the interrupt controller, interrupts cannot be received in the ID stage during the second half of this instruction.</p>
	SATADD SATSUB SATSUBI SATSUBR	Use LDSR instruction to clear the SAT flag to "0."

Location		Description
5. 3 Instruction Set	SLD.H SLD.HU SST.H	<p>Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>•Bit0 is masked to "0" and address is generated (when misaligned access is disabled).</li> <li>•Bit0 is not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see 3.3 Data Alignment.</p>
	SLD.W SST.W	<p>Adding the element pointer to the 8-bit displacement data, zero-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>•Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).</li> <li>•Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see 3.3 Data Alignment.</p>
	ST.H	<p>Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>•Bit0 is masked to "0" and address is generated (when misaligned access is disabled).</li> <li>•Bit0 is not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see 3.3 Data Alignment.</p>
	ST.W	<p>Adding the data of the general register reg1 to the 16-bit displacement data, sign-extended to word length, can generate two types of results. It depends on the misaligned mode setting:</p> <ul style="list-style-type: none"> <li>• Bit0 and bit1 are masked to "0" and address is generated (when misaligned access is disabled).</li> <li>• Bit0 and bit1 are not masked and address is generated (when misaligned access is enabled).</li> </ul> <p>For details on misaligned access, see 3.3 Data Alignment.</p>
	STSR	The system register number regID is to identify a system register. Accessing reserved system registers is prohibited.
6. 2. 2 Exception trap		The operation using the instruction undefined is not guaranteed.
CHAPTER 9 Shifting to Debug Mode		<p>When the mode is shifted to debug mode, the data cache (dCACHE) is set to Hold mode and its data and tags are not updated. If a cacheable area of external memory is accessed during debug mode, cohesion may be lost even when the dCACHE is valid and access is only to external memory. Therefore, before manipulating any data in a cacheable area as part of a debug monitor routine, be sure to first return to user mode and clear the dCACHE (for write through) or flush and clear it (for write back).</p>

Location	Description
9. 1 Methods for Switching to Debug Mode	When the BPCn register's IE bit is set, if the Program IDs set via the BP ASID bit and ASID register do not match, the mode will not be switched to debug mode even if the break condition has been met.
	The timing by which break conditions are met differs between execution-related traps and access-related traps. Consequently, even when sequential break mode has been set, normal operation may not occur if an execution-related trap occurs after an access-related trap.
	When in latency break mode, set either execution-related traps or access-related traps (when using channels 0 and 1 or channels 2 and 3).
9. 2 Caution Points	(Refer to the text)

## APPENDIX F INSTRUCTION INDEX

[A]	[J]	[R]	[T]
ADD ..... 61	JARL ..... 90	RETI ..... 117	TRAP ..... 153
ADDI ..... 62	JMP ..... 91		TST ..... 154
ADF ..... 63	JR ..... 92	[S]	TST1 ..... 155
AND ..... 64		SAR ..... 119	
ANDI ..... 65	[L]	SASF ..... 120	[X]
	LD.B ..... 93	SATADD ..... 121	XOR ..... 156
[B]	LD.BU ..... 94	SATSUB ..... 123	XORI ..... 157
Bcond ..... 66	LD.H ..... 95	SATSUBI ..... 124	
BSH ..... 68	LD.HU ..... 96	SATSUBR ..... 125	[Z]
BSW ..... 69	LD.W ..... 97	SBF ..... 126	ZXB ..... 158
	LDSR ..... 98	SCH0L ..... 127	ZXH ..... 159
[C]		SCH0R ..... 128	
CALLT ..... 70	[M]	SCH1L ..... 129	
CLR1 ..... 71	MAC ..... 99	SCH1R ..... 130	
CMOV ..... 72	MACU ..... 100	SET1 ..... 131	
CMP ..... 74	MOV ..... 101	SETF ..... 132	
CTRET ..... 75	MOVEA ..... 102	SHL ..... 134	
	MOVHI ..... 103	SHR ..... 135	
[D]	MUL ..... 104	SLD.B ..... 136	
DBRET ..... 76	MULH ..... 106	SLD.BU ..... 137	
DBTRAP ..... 77	MULHI ..... 107	SLD.H ..... 138	
DI ..... 78	MULU ..... 108	SLD.HU ..... 139	
DISPOSE ..... 79		SLD.W ..... 140	
DIV ..... 81	[N]	SST.B ..... 141	
DIVH ..... 82	NOP ..... 110	SST.H ..... 142	
DIVHU ..... 84	NOT ..... 111	SST.W ..... 143	
DIVU ..... 85	NOT1 ..... 112	ST.B ..... 144	
		ST.H ..... 145	
[E]	[O]	ST.W ..... 146	
EI ..... 86	OR ..... 113	STSR ..... 147	
	ORI ..... 114	SUB ..... 148	
[H]		SUBR ..... 149	
HALT ..... 87	[P]	SWITCH ..... 150	
HSH ..... 88	PREPARE ..... 115	SXB ..... 151	
HSW ..... 89		SXH ..... 152	