NEC

User's Manual

V850E/MS1™, V850E/MS2™

32-Bit Single-Chip Microcontrollers

Architecture

V850E/MS1:

- μPD703100
- μPD703100A
- μPD703101
- μPD703101A
- μPD703102
- μPD703102A
- μPD70F3102
- μPD70F3102A

V850E/MS2:

- μPD703130

Document No. U12197EJ6V0UM00 (6th edition)
Date Published November 2002 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

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 once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build 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 bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

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 in this document is current as of August, 2002. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with 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 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 NEC Electronics 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 customer's equipment shall be done under the full responsibility of 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 NEC Electronics product 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 NEC Electronics sales representative in advance to determine NEC Electronics's 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).

M8E 02. 11

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.

NEC Electronics America, Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
  Fax: 091-504 28 60

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
  Fax: 01-30-67 58 99

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
  Fax: 02-66 75 42 99

- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-244 58 45
  Fax: 040-244 45 80

- **Tyskland Filial**
  Taeby, Sweden
  Tel: 08-63 80 820
  Fax: 08-63 80 388

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133
  Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Shanghai, Ltd.
Shanghai, P.R. China
Tel: 021-6841-1138
Fax: 021-6841-1137

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 6253-8311
Fax: 6250-3583

J02.11

Major Revisions in This Edition

Page	Description
p.60	Modification of description of **CLR1 instruction** in **5.3 Instruction Set**
p.90	Modification of description of **NOT1 instruction** in **5.3 Instruction Set**
p.105	Modification of description of **SET1 instruction** in **5.3 Instruction Set**
p.110	Modification of description of **SLD1 instruction** in **5.3 Instruction Set**
p.112	Addition of description of **SST instruction** in **5.3 Instruction Set**
p.185	Addition of **APPENDIX F REVISION HISTORY**

The mark ⋆ shows major revised points.

PREFACE

Readers

This manual is intended for users who wish to understand the functions of the V850E/MS1 and V850E/MS2 for designing systems using the V850E/MS1 and V850E/MS2. The following products are described.

- V850E/MS1: μPD703100, 703100A, 703101A, 703102, 703102A, 70F3102, 70F3102A
- V850E/MS2: μPD703130

Purpose

This manual presents information on the architecture and instruction set of the V850E/MS1 and V850E/MS2.

Organization

This manual contains the following information:

- Register set
- Data type
- Instruction format and instruction set
- Interrupts and exceptions
- Pipeline flow

How to Read This Manual

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

To learn about the hardware functions,

→ Read **V850E/MS1 Hardware User's Manual** and **V850E/MS2 Hardware User's Manual**.

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

→ Read **CHAPTER 5 INSTRUCTIONS**.

To learn about the electrical specifications,

→ Read the **DATA SHEET** of each device.

To understand the overall functions of the V850E/MS1 and V850E/MS2,

→ Read this manual in the order of the contents.

With the V850E/MS1 and V850E/MS2, data consisting of 2 bytes is called a halfword, and data consisting of 4 bytes is called a word.
In this manual, the V850E/MS1 is explained as the typical product unless there are any functional differences.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low:	$\overline{\times\times\times}$ (overscore over pin or signal name)
Memory map addresses:	Higher addresses on the top and lower addresses on the bottom
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Binary ... ×××× or ××××B
	Decimal ... ××××
	Hexadecimal ... ××××H

Prefix indicating the power of 2 (address space, memory capacity):

K (Kilo):	$2^{10} = 1024$
M (Mega):	$2^{20} = 1024^2$
G (Giga):	$2^{30} = 1024^3$

Data type:	Word…32 bits
	Halfword…16 bits
	Byte…8 bits

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

- **Documents related to devices**

Document Name	Document Number
μPD703100-33, 703100-40, 703101-33, 703102-33 Data Sheet	U13995E
μPD703100A-33, 703100A-40, 703101A-33, 703102A-33 Data Sheet	U14168E
μPD70F3102-33 Data Sheet	U13844E
μPD70F3102A-33 Data Sheet	U13845E
μPD703130 Data Sheet	U15390E
V850E/MS1 Hardware User's Manual	U12688E
V850E/MS2 Hardware User's Manual	U14985E
V850E/MS1, V850E/MS2 Architecture User's Manual	This manual
V850E/MS1 Hardware Application Note	U14214E

- **Documents related to development tools (user's manuals)**

Document Name		Document Number
IE-703102-MC (In-circuit emulator)		U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A (In-circuit emulator option board)		U13876E
CA850 (Ver.2.30 or Later) (C compiler package)	Operation	U14568E
	C language	U14566E
	Project manager	U14569E
	Assembly language	U14567E
ID850 (Integrated debugger) Ver.2.20	Operation Windows™ based	U14580E
SM850 (System simulator) Ver.2.20	Operation Windows based	U14782E
SM850 System Simulator (Ver. 2.00 or Later)	External Part User Open Interface Specifications	U14873E
RX850 (Ver.3.13 or Later) (Real-time OS)	Fundamental	U13430E
	Installation	U13410E
	Technical	U13431E
RX850 Pro (Ver.3.13) (Real-time OS)	Fundamental	U13773E
	Installation	U13774E
	Technical	U13772E
RD850 (Ver.3.01) (Task debugger)		U13916E
RD850 Pro. (Ver.3.01) (Task debugger)		U13737E
AZ850 (Ver.3.0) (System performance analyzer)		U11181E
PG-FP3 (Flash memory programmer)		U13502E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

The V850 Series™ is a collection of NEC Electronics single-chip microcontrollers that have a CPU core that uses the RISC microprocessor technology of the V800 Series™, and incorporate functions such as internal ROM/RAM and peripheral I/O.

The V850 Series of microcontrollers provides a migration path to NEC Electronics' existing 78K Series of original single-chip microcontrollers, and boasts a higher cost-performance.

The V850 Series includes products that incorporate the V850 CPU and products that incorporate the V850E CPU. The V850E/MS1 is one of the latter.

This chapter briefly outlines the V850 Series.

1.1 General

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,
- Automotive electronics such as engine control systems and ABSs (Antilock Braking Systems)
- 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 design. 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 provides maximum performance with simpler hardware, allowing 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 saturation, bit manipulation, and multiply/divide (executed by a hardware multiplier), which are especially well suited to digital servo control systems. Moreover, instruction formats are designed for maximum compiler coding efficiency, allowing the reduction in the object code size.

1.2 Features

- High-performance 32-bit architecture for embedded control
  - Number of instructions: 81
  - 32-bit general-purpose registers: 32
  - Load/store instructions in long/short format
  - 3-operand instruction
  - 5-stage pipeline of 1 clock cycle per stage
  - Hardware interlock on register/flag hazards
  - Memory space Program space: 64 MB linear
    Data space: 4 GB linear
- Special instructions
  - Saturation operation instructions
  - Bit manipulation instructions
  - On-chip multiplier executing multiplication in 1 to 2 clocks
    - 16 bits $\times$ 16 bits $\rightarrow$ 32 bits
    - 32 bits $\times$ 32 bits $\rightarrow$ 32 or 64 bits

1.3 Product Development

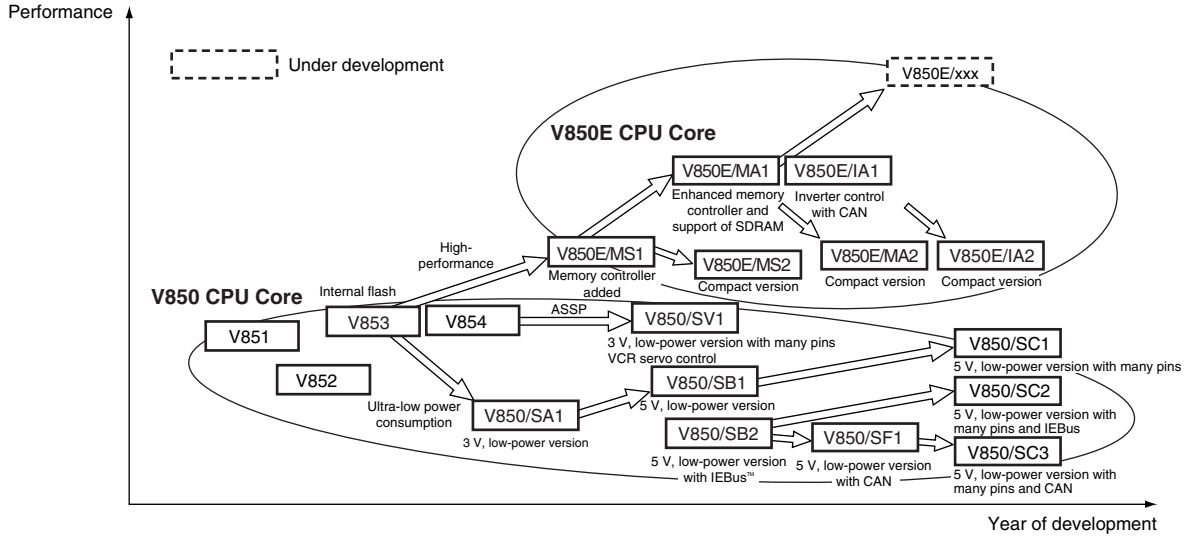
The V850 Series is part of the V800 Series and consists of single-chip microcontrollers using a RISC microprocessor core.

The members of V850 Series are the V851™, V852™, V853™, V854™, V850/SV1™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SF1™, V850/SC1™, V850/SC2™, and V850/SC3™, which incorporate the V850 CPU, and the V850E/MS1, V850E/MS2, V850E/MA1™, V850E/MA2™, V850E/IA1™, V850E/IA2™, and V850E/xxx, which incorporate the V850E CPU.

The versions incorporating the V850 CPU are single-chip microcontrollers for control, and the versions incorporating the V850E CPU are single-chip microcontrollers that feature an enhanced bus interface and are suitable for data processing in addition to control.

Moreover, the V850E CPU differs from the V850 in that it provides additional instructions mainly for high-level languages, such as C language switch statement processing, table lookup branching, stack frame generation/deletion, and data conversion. The instruction code is upwardly compatible at the object code level with the V850 CPU, allowing the software resources contained in the V850 CPU to be used as is.

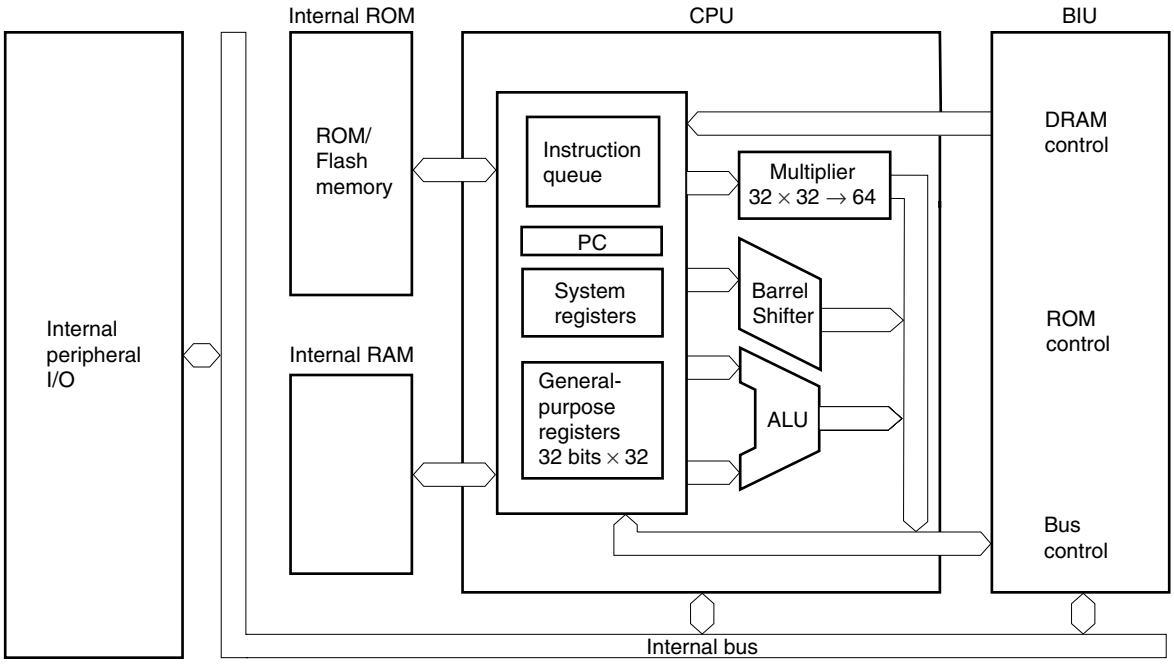
Figure 1-1. V850 Series Lineup



1.4 CPU Configuration

Figure 1-2 shows the internal configuration of the V850E/MS1.

Figure 1-2. Internal Configuration



The function of each hardware block is as follows.

CPU ..... Executes almost all instructions such as address calculations, arithmetic and logical operations, and data transfers in one clock by using a 5-stage pipeline. Contains dedicated hardware such as a multiplier (32 × 32 bits) and a barrel shifter (32 bits/clock) to execute complicated instructions at high speeds.

Internal ROM ..... <V850E/MS1>
ROM or flash memory mapped from address 00000000H. Can be accessed by the CPU in one clock during instruction fetch.
<V850E/MS2>
Internal ROM is not provided.

Internal RAM ..... RAM mapped to a space preceding address FFFFEFFFH. Can be accessed by the CPU in one clock during data access.

Internal peripheral I/O ..... Peripheral I/O area mapped from address FFFFF000H.

BIU ..... Starts a necessary bus cycle based on a physical address obtained by the CPU.

1.5 Differences with Architecture of V850 CPU

The differences between the architecture of the V850E CPU and that of the V850 CPU are listed below.

Table 1-1. Differences Between V850E CPU and V850 CPU (1/2)

Item		V850E CPU	V850 CPU
Instructions (including operand)	BSH reg2, reg3	Provided	Not provided
	BSW reg2, reg3		
	CALLT imm6		
	CLR1 reg2, [reg1]		
	CMOV cccc, imm5, reg2, reg3		
	CMOV cccc, reg1, reg2, reg3		
	CTRET		
	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		
	LD.BU disp16 [reg1] , reg2		
	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		

Table 1-1. Differences Between V850E CPU and V850 CPU (2/2)

Item		V850E CPU	V850 CPU
Instruction format	Format IV	Format is different for some instructions.	
	Format XI	Provided	Not provided
	Format XII		
	Format XIII		
Instruction execution clocks		Value differs for some instructions.	
Program space		64 MB linear	16 MB linear
Valid bits of program counter (PC)		Lower 26 bits	Lower 24 bits
System registers	CALLT execution status save registers (CTPC, CTPSW)	Provided	Not provided
	CALLT base pointer (CTBP)		
	Exception trap status save registers	DBPC, DBPSW	EIPC, EIPSW
Instruction code of illegal instruction code trap		Instruction code areas differ.	
Misalign access enable/disable setting		Can be set.	Cannot be set. (misalign access prohibited)
Access time (No. of clocks)	Internal RAM (at instruction fetch)	1 or 2	3
	External memory	2^{Note} + No. of waits	3 + No. of waits
Pipeline		At next instruction, pipeline flow differs. • Arithmetic instruction (except multiply instruction) • Branch instruction • Bit manipulation instruction • Special instruction (TRAP, RETI)	

Note When external memory type is set to SRAM, I/O

CHAPTER 2 REGISTER SET

The registers of the V850 Series 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 the registers consist of 32 bits.

2.1 Program Registers

2.1.1 Program register set

(1) General-purpose registers

The V850 Series has thirty-two general-purpose registers, r0 through r31. All these registers can be used for data or address storage.

However, r0 and r30 are implicitly used by instructions, and care must be exercised in using these registers. r0 is a register that always holds 0, and is used for operations and offset 0 addressing. r30 is used as a base pointer when accessing memory using the SLD and SST instructions. r1, r3, r4, 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. The real-time OS may use r2. When real-time OS does not use r2, r2 can be used as a variable register.

Figure 2-1. Program Registers

31 0

r0	Zero register
r1	Reserved for address generation
r2	
r3	Stack pointer (SP)
r4	Global pointer (GP)
r5	Text pointer (TP)
r6	
r7	
r8	
r9	
r10	
r11	
r12	
r13	
r14	
r15	
r16	
r17	
r18	
r19	
r20	
r21	
r22	
r23	
r24	
r25	
r26	
r27	
r28	
r29	
r30	Element pointer (EP)
r31	Link pointer (LP)

PC	Program counter

Figure 2-2. Program Register Operations

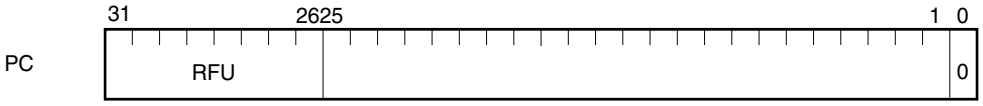
Name	Usage	Operation
r0	Zero register	Always holds 0.
r1	Assembler-reserved register	Used as working register for address generation.
r2	Address/data variable registers (when the real-time OS does not use r2)	
r3	Stack pointer	Used for stack frame generation when function is called.
r4	Global pointer	Used to access global variable in data area.
r5	Text pointer	Used as register for pointing start address of text area[Note]
r6 to r29	Address/data variable registers	
r30	Element pointer	Used as base pointer for address generation when memory is accessed.
r31	Link pointer	Used when compiler calls function.
PC	Program counter	Holds instruction address during program execution.

Note Text area: Area where program code is placed.

Remark For detailed descriptions of r1, r3, r4, r5, r31 used by the assembler and C compiler, see the **CA850 (C Compiler Package) User's Manual**.

(2) Program counter

This register holds an instruction address during program execution. The lower 26 bits of this register are valid, and bits 31 through 26 are reserved fields (fixed to 0). If a carry occurs from bit 25 to 26, it is ignored.
Bit 0 is always fixed to 0, and execution cannot branch to an odd address.



Remark RFU: Reserved field (Reserved for Future Use)

2.2 System Registers

The system registers control the status of the V850 Series and hold information on interrupts.

Figure 2-3. System Registers

31	0
EIPC	Exception/Interrupt PC
EIPSW	Exception/Interrupt PSW
FEPC	Fatal Error PC
FEPSW	Fatal Error PSW
ECR	Exception Cause Register
PSW	Program Status Word
CTPC	CALLT Caller PC
CTPSW	CALLT Caller PSW
DBPC	ILGOP Caller PC
DBPSW	ILGOP Caller PSW
CTBP	CALLT Base Pointer

2.2.1 Interrupt status saving registers

Two interrupt status saving registers are provided: EIPC and EIPSW.

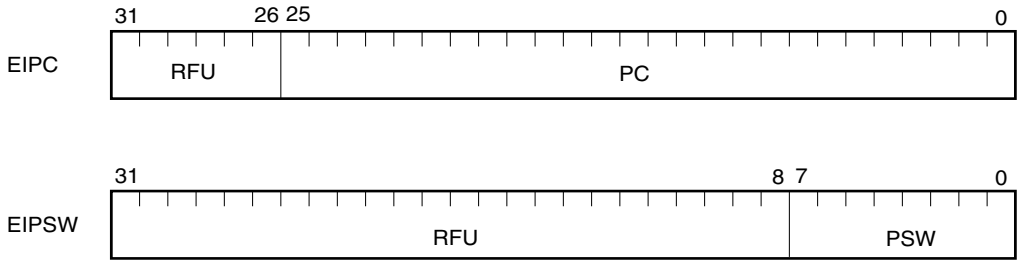
The contents of the PC and PSW are respectively saved in these registers if a software exception or interrupt occurs. If an NMI occurs, however, the contents of the PC and PSW are saved to the NMI status saving registers.

When a software exception or interrupt occurs, the address of the following instruction is saved in the EIPC register. If an interrupt occurs while a division (DIV/DIVH/DIVU) instruction is being executed, the address of the division instruction currently being executed is saved.

The current value of the PSW is saved to the EIPSW.

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.

Bits 26 through 31 of the EIPC and bits 8 through 31 of the EIPSW are fixed to 0.



2.2.2 NMI status saving registers

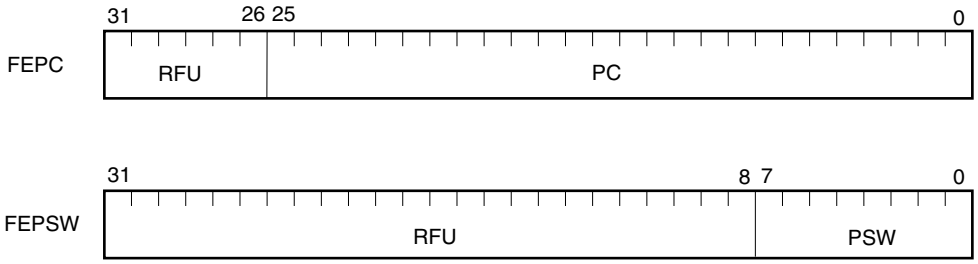
The V850 Series is provided with two NMI status saving registers: FEPC and FEPSW.

The contents of the PC and PSW are respectively saved in these registers when an NMI occurs.

The value saved to the FEPC is, like the EIPC, the address of the instruction next to the one executed when the NMI has occurred (if the NMI occurs while a division (DIVH/DIV/DIVU) instruction is being executed, the address of the division instruction under execution is saved).

The current value of the PSW is saved to the FEPSW.

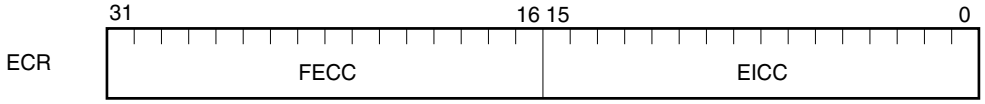
Bits 26 through 31 of the FEPC and bits 8 through 31 of the FEPSW are fixed to 0.



2.2.3 Exception cause register

The exception cause register (ECR) holds the cause information of an exception, maskable interrupt, or NMI when any of these events occur. The ECR holds a code which identifies each interrupt source.

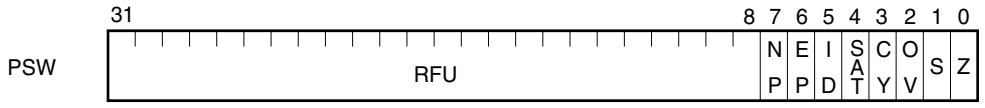
This is a read-only register, and therefores no data can be written to it by using the LDSR instruction.



Bit Position	Field	Function
31 to 16	FECC	Fatal Error Cause Code NMI code
15 to 0	EICC	Exception/Interrupt Cause Code Exception/interrupt code

2.2.4 Program status word

The program status word is a collection of flags that indicate the status of the program (result of instruction execution) and the status of the CPU. If the contents of the PSW register are modified by the LDSR instruction, the PSW will assume the new value immediately after the LDSR instruction has been executed. In setting the ID flag to 1, however, interrupts are already disabled even while the LDSR instruction is being executed.



Bit Position	Flag	Function
31 to 8	RFU	Reserved for Future Use Reserved field (fixed to 0).
7	NP	NMI Pending Indicates that NMI processing is in progress. This flag is set when an NMI is acknowledged. The NMI request is then masked, and multiple interrupts are disabled. NP = 0: NMI processing is not in progress NP = 1: NMI processing is in progress
6	EP	Exception Pending Indicates that exception processing is in progress. This flag is set when an exception occurs. Even when this bit is set, interrupt requests can be acknowledged. EP = 0: Exception processing is not in progress EP = 1: Exception processing is in progress
5	ID	Interrupt Disable Indicates whether external interrupt request can be acknowledged. ID = 0: Interrupt can be acknowledged ID = 1: Interrupt cannot be acknowledged
4	SAT[Note]	Saturated Indicates that an overflow has occurred in a saturated operation and the result is saturated. This is a cumulative flag. Once the result is saturated, the flag is set to 1 and is not reset to 0 even if the next result is not saturated. To reset this flag, load data to the PSW. This flag is neither set nor reset by general arithmetic operation instruction. SAT = 0: Not saturated SAT = 1: Saturated
3	CY	Carry Indicates whether a carry or borrow occurred as a result of the operation. CY = 0: Carry or borrow did not occur CY = 1: Carry or borrow occurred
2	OV[Note]	Overflow Indicates whether an overflow occurred as a result of the operation. OV = 0: Overflow did not occur OV = 1: Overflow occurred
1	S[Note]	Sign Indicates whether the result of the operation is negative S = 0: Result is positive or zero S = 1: Result is negative
0	Z	Zero Indicates whether the result of the operation is zero Z = 0: Result is not zero Z = 1: Result is zero

Note In the case of saturation instructions, the SAT, S, and OV flags will be set according to the result of the operation as shown in the table below. Note that the SAT flag is set to 1 only when the OV flag has been set due to an overflow condition caused by a saturation instruction.

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

2.2.5 CALLT caller status saving registers

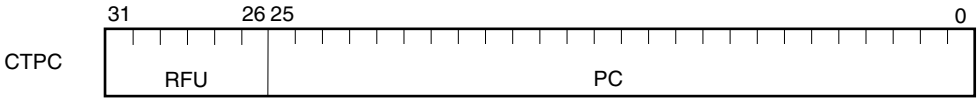
The V850E Series is provided with two CALLT caller status saving registers: CTPC and CTPSW.

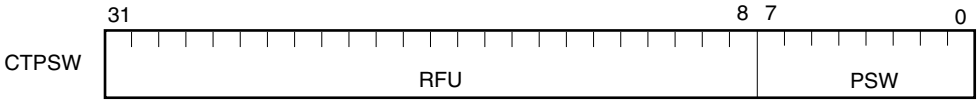
The contents of the PC and PSW are respectively saved in these registers when a CALLT instruction is executed.

The value saved to CTPC is, like the EIPC, the address of the instruction next to the one executed.

The current value of the PSW is saved to CTPSW.

Bits 26 through 31 of CTPC and bits 8 through 31 of CTPSW are fixed to 0.





2.2.6 ILGOP caller status saving registers

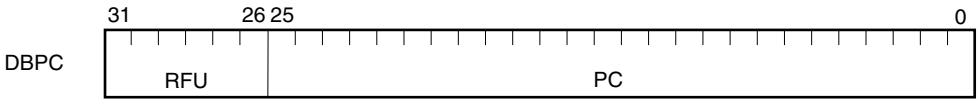
The V850E Series is provided with two ILGOP caller status saving registers: DBPC and DBPSW.

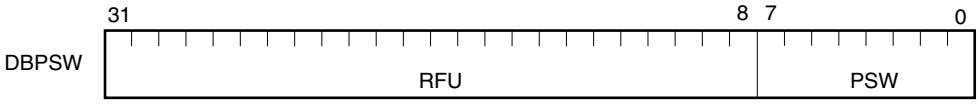
The contents of the PC and PSW are respectively saved in these registers when ILGOP is detected.

The value saved to DBPC is, like the EIPC, the address of the instruction next to the one executed.

The current value of the PSW is saved to DBPSW.

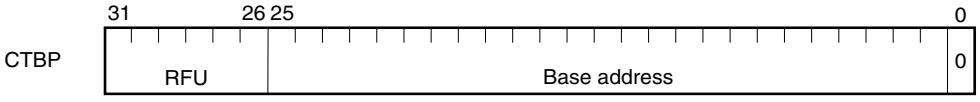
Bits 26 through 31 of DBPC and bits 8 through 31 of DBPSW are fixed to 0.





2.2.7 CALLT base pointer

The CALLT base pointer CTBP is used to specify a table address and to generate a target address.



2.2.8 System register number

Data in the system registers is accessed by using the load/store system register instructions, LDSR and STSR. Each register is assigned a unique number which is referenced by the LDSR and STSR instructions.

Table 2-1. System Register Number

Number	System Register	Operand Specification	
		LDSR	STSR
0	EIPC	○	○
1	EIPSW	○	○
2	FEPC	○	○
3	FEPSW	○	○
4	ECR	—	○
5	PSW	○	○
16	CTPC	○	○
17	CTPSW	○	○
18	DBPC	○	○
19	DBPSW	○	○
20	CTBP	○	○
6 to 15 21 to 31	Reserved	—	—

—: Access prohibited

○: Access enabled

Reserved: Accessing registers in this range is prohibited and will lead to undefined results.

Caution When using the LDSR instruction with the EIPC, FEPC and CTPC registers, only even address values should be specified. After interrupt processing has ended with a RETI instruction, bit 0 in the EIPC, FEPC and CTPC registers will be ignored and assumed to be zero when the PC is restored.

CHAPTER 3 DATA TYPES

3.1 Data Format

The V850 Series supports the following data types.

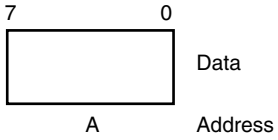
- Integer (8, 16, 32 bits)
- Unsigned integer (8, 16, 32 bits)
- Bit

3.1.1 Data type and addressing

The V850 Series supports three types of data lengths: word (32 bits), halfword (16 bits), and byte (8 bits). Byte 0 of any data is always the least significant byte (this is called little endian) and is shown at the rightmost position in figures throughout this manual. The following paragraphs describe the data format where data of a fixed length is in the memory.

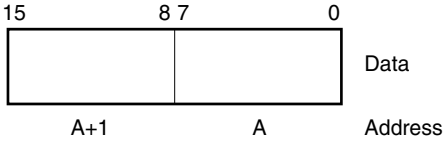
(1) Byte (BYTE)

A byte is 8-bit contiguous data that starts from any byte boundary[Note]. Each bit is assigned a number from 0 to 7. The LSB (Least Significant Bit) is bit 0 and the MSB (Most Significant Bit) is bit 7. A byte is specified by its address A.



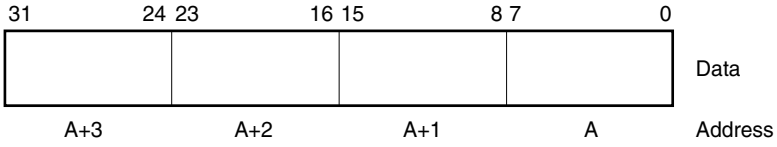
(2) Halfword (HALF-WORD)

A halfword is 2 byte (16-bit) contiguous data that starts from any halfword boundary[Note]. 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 when misalign access is disabled)[Note], and occupies 2 bytes, A and A+1.



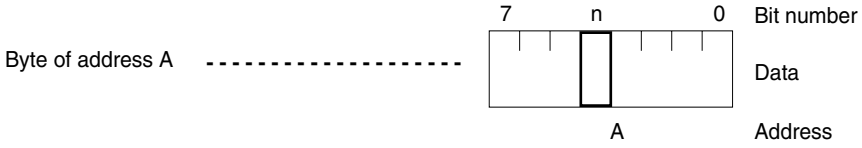
(3) Word (WORD)

A word is 4-byte (32-bit) contiguous data that starts from any word boundary[Note]. Each bit is assigned a number from 0 to 31. The LSB is bit 0 and the MSB is bit 31. A word is specified by its address A (with the 2 lowest bits fixed to 0 when misalign access is disabled)[Note], and occupies 4 bytes, A, A+1, A+2, and A+3.



(4) Bit (BIT)

A bit is 1-bit data at the nth bit position in 8-bit data that starts from any byte boundary[Note]. A bit is specified by its address A and bit number n.



Note The V850E Series can access any byte boundary whether access is in halfword or word units when misalign access is enabled.
Refer to **3.3 Data Alignment**.

3.2 Data Representation

3.2.1 Integer

With the V850 Series, an integer is expressed as a binary number of 2's complement and is 8, 16, or 32 bits long. Regardless of its length, bit 0 of an integer is the least significant bit. The higher the bit number, the more significant the bit. Because 2's complement is used, the most significant bit is used as a sign bit.

Data Length		Range
Byte	8 bits	–128 to +127
Halfword	16 bits	–32768 to +32767
Word	32 bits	–2147483648 to +2147483647

3.2.2 Unsigned integer

While an integer is data that can take either a positive or a negative value, an unsigned integer is an integer that is not negative. Like an integer, an unsigned integer is also expressed as 2's complement and is 8, 16, or 32 bits 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. However, no sign bit is used.

Data Length		Range
Byte	8 bits	0 to 255
Halfword	16 bits	0 to 65535
Word	32 bits	0 to 4294967295

3.2.3 Bit

The V850 Series can handle 1-bit data that can take a value of 0 (cleared) or 1 (set). Bit manipulation can only be performed on 1-byte data in the memory space in the following four ways.

- Set
- Clear
- Invert
- Test

3.3 Data Alignment

With the V850E Series, data to be allocated in memory must be aligned at an appropriate boundary when misalign access is disabled. Therefore, word data must be aligned at a word boundary (the lower 2 bits of the address are 0), and halfword data must be aligned at a halfword boundary (the lower 1 bit of the address is 0). If data is not aligned at a boundary and misalign access disabled, the data is accessed with the lowest bit(s) of the address (lower 2 bits in the case of word data and lowest 1 bit in the case of halfword data) automatically masked. This will cause loss of data and truncation of the least significant bytes.

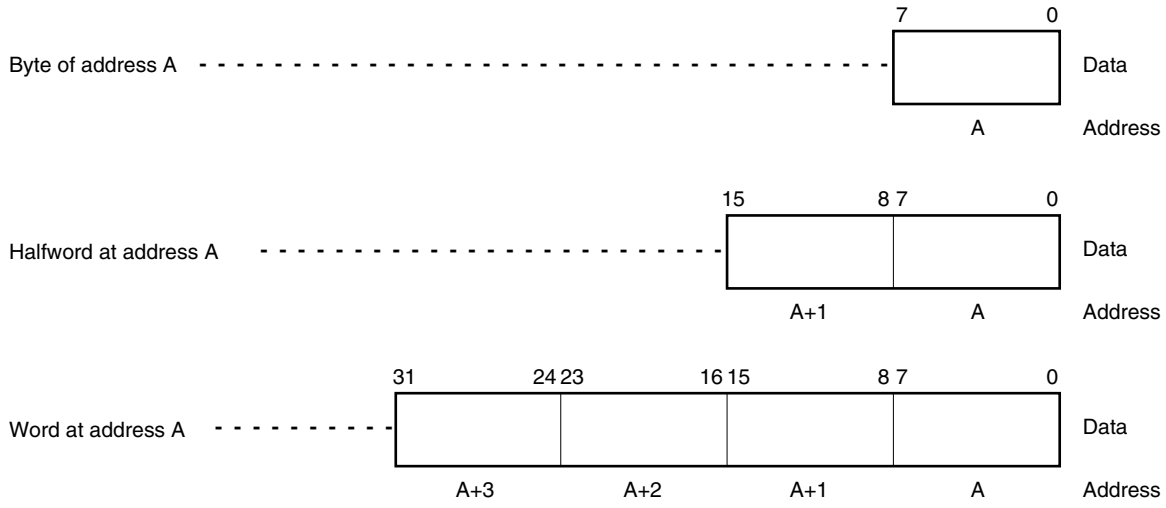
When misalign access is enabled, it is possible to place any data at any address, irrespective of the data format when data is word or halfword and is not aligned at a boundary, however one or more bus cycles is generated, which lowers the bus efficiency.

CHAPTER 4 ADDRESS SPACE

The V850 Series supports a 4 GB linear address space. Both memory and I/O are mapped to this address space (**memory-mapped I/O**). The V850 Series outputs 32-bit addresses to the memory and I/O. The maximum address is $2^{32}-1$.

Byte ordering is little endian. Byte data allocated at each address is defined with bit 0 as LSB and bit 7 as MSB. In regards to multiple-byte data, the byte with the lowest address value is defined to have the LSB and the byte with the highest address value is defined to have the MSB.

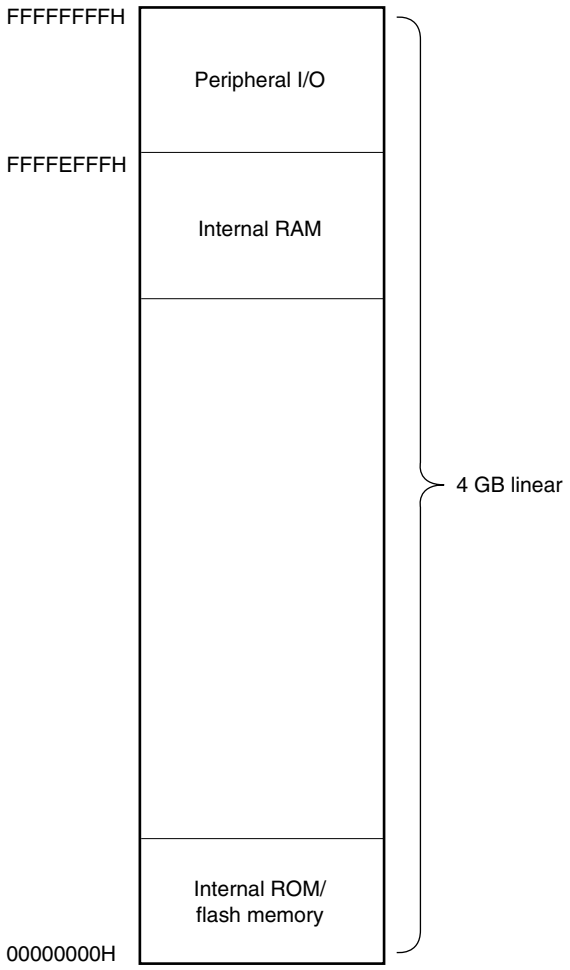
Data consisting of 2 bytes is called a halfword, and 4-byte data is called a word. In this user's manual, data consisting of 2 or more bytes is illustrated as shown below, with the lower address shown on the right and the higher address on the left.



4.1 Memory Map

The V850 Series employs a 32-bit architecture and supports a linear address space (data space) of up to 4 GB. It supports a linear address space (program space) of up to 64 MB for instruction addressing.
Figure 4-1 shows the memory map of the V850 Series.

Figure 4-1. Memory Map



4.2 Addressing Mode

The CPU generates two 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 (+2) according to the number of bytes of an instruction to be fetched each time an instruction has been executed. When a branch instruction is executed, the branch destination address is loaded into the PC using one of the following two addressing modes.

(1) Relative addressing (PC relative)

The signed 9- or 22-bit data of an instruction code (displacement: disp) is added to the value of the program counter (PC). At this time, the displacement is treated as 2's complement data with bits 8 and 21 serving as sign bits.

This addressing is used for Bcond disp9, JR disp22, and JARL disp22, reg2 instructions.

Figure 4-2. Relative Addressing (JR disp22/JARL disp22, reg2)

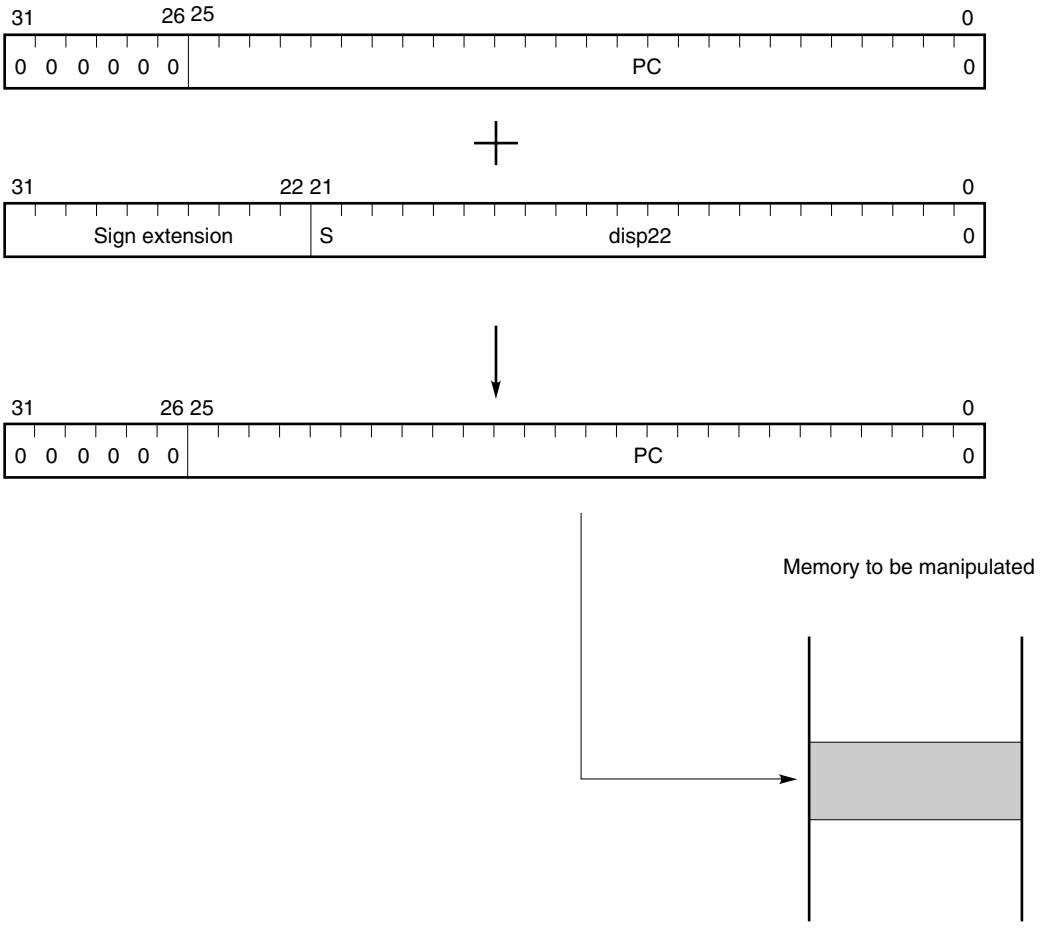
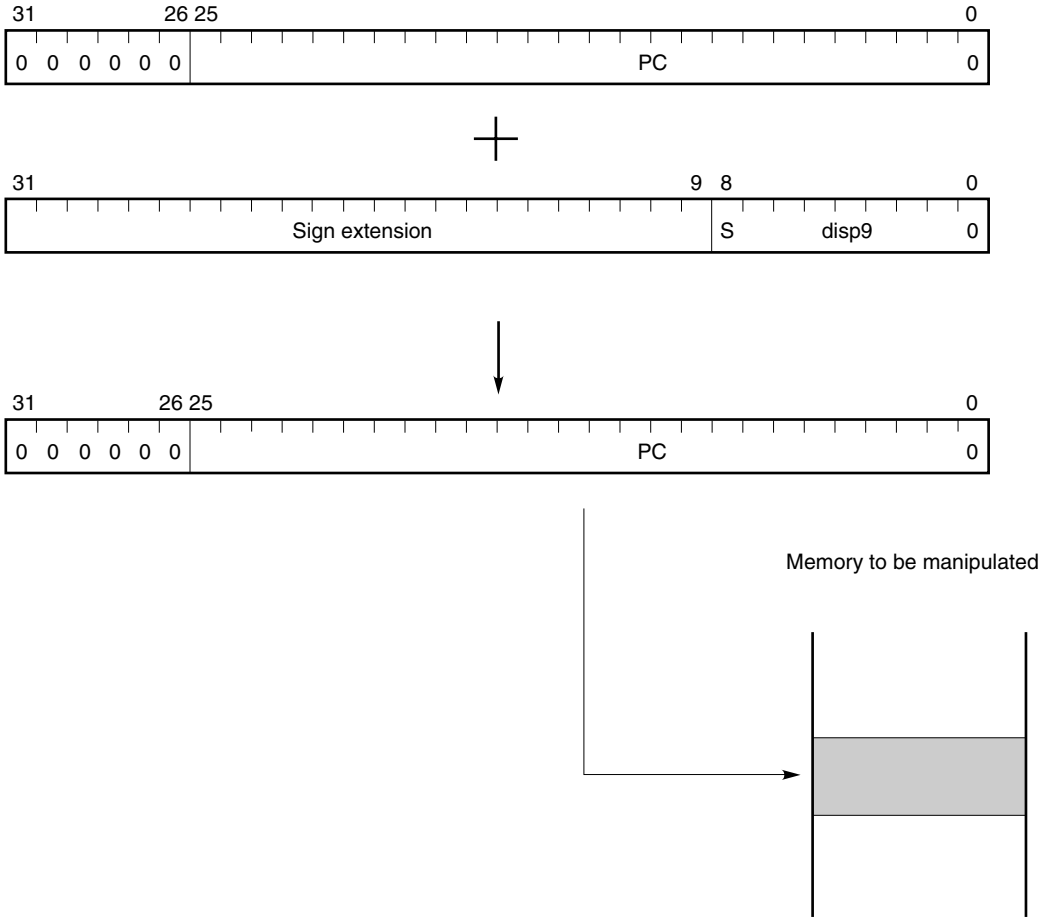


Figure 4-3. Relative Addressing (Bcond disp9)

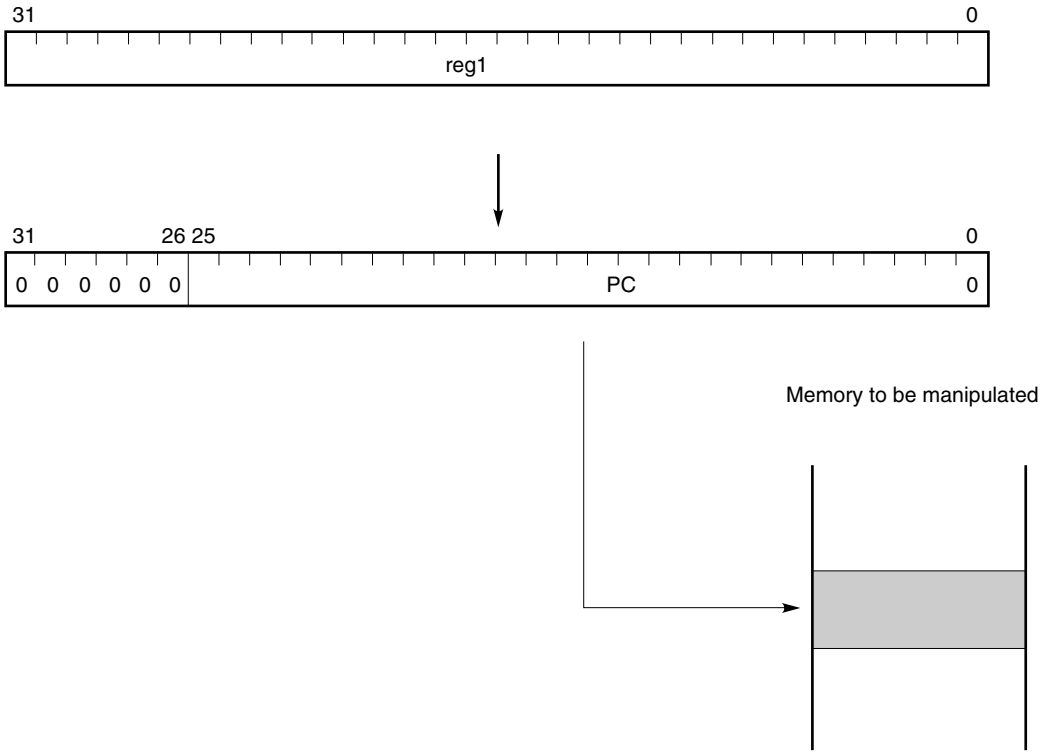


(2) Register addressing (address indirect)

The contents of a general-purpose register (r0 to r31) specified by an instruction are transferred to the program counter (PC).

This addressing is applied to the JMP [reg1] instruction.

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



4.2.2 Operand address

When an instruction is executed, the register or memory area to be accessed is specified in one of the following four addressing modes.

(1) Register addressing

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

(2) Immediate addressing

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

Remark vector: An operand that is 5-bit immediate data that specifies the trap vector (00H to 1FH), and is used by the TRAP instruction.

cccc: An operand consisting of 4-bit data used by the SETF and CMOV instructions to specify the condition code. Assigned as part of the instruction code as 5-bit immediate data by appending a 1-bit 0 above the highest bit.

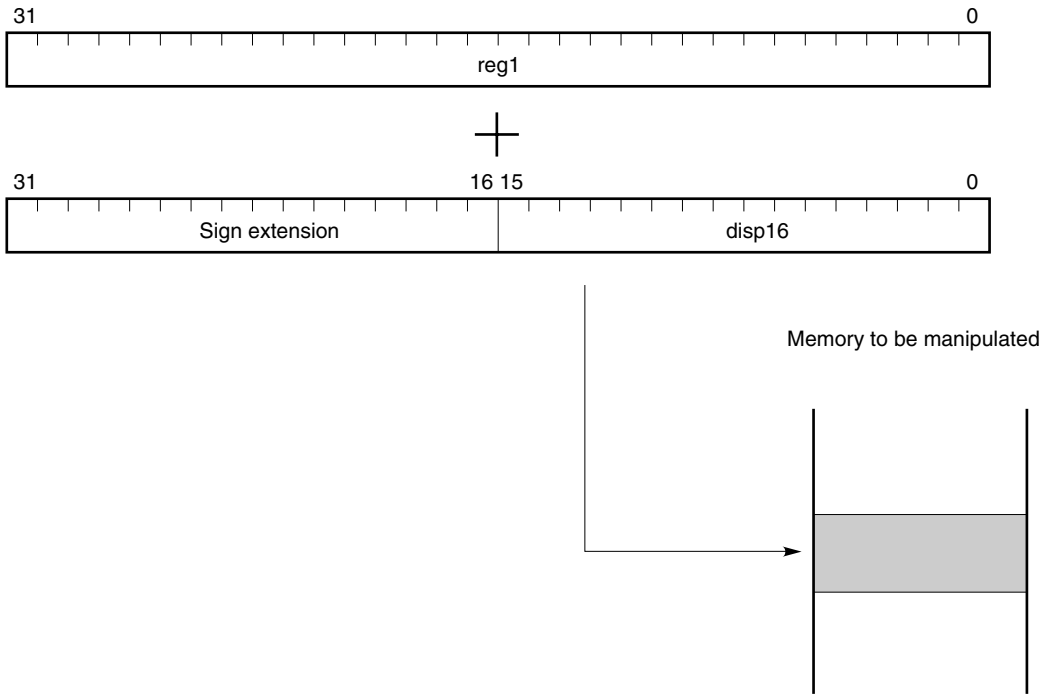
(3) Based addressing

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-purpose register to the 16-bit displacement value contained in the instruction. This addressing mode applies to instructions using the operand format disp16 [reg1].

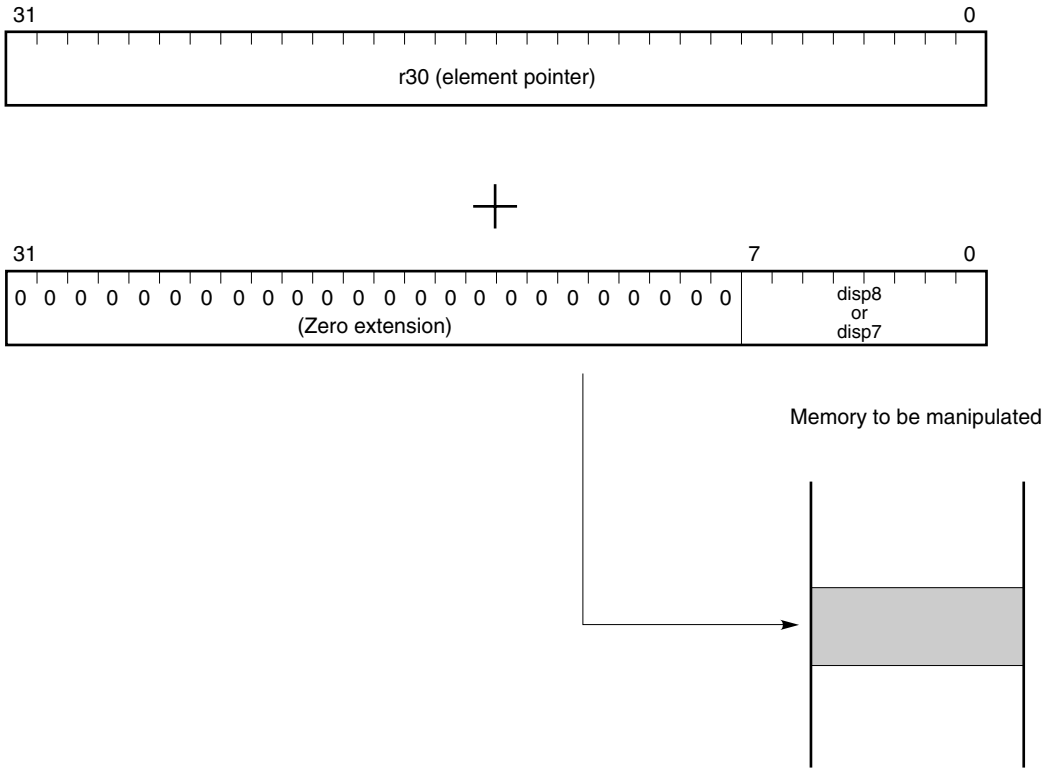
Figure 4-5. Based Addressing (Type 1)



(b) Type 2

The address of the data memory location to be accessed is determined by adding the value in the 32-bit element pointer (r30) to the 7- or 8-bit displacement value contained in the instruction. This addressing mode applies to SLD and SST instructions.

Figure 4-6. Based Addressing (Type 2)



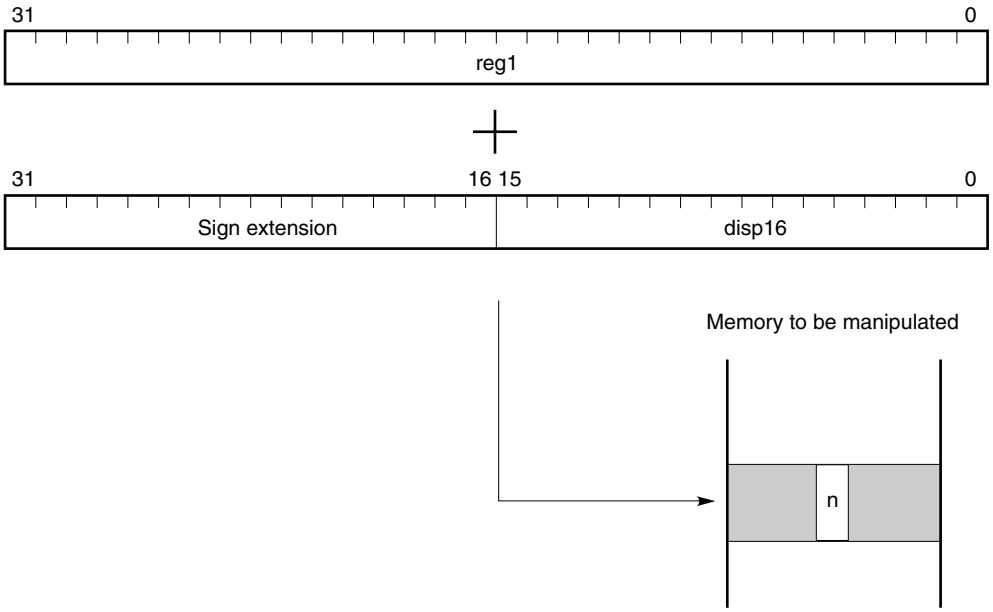
Byte access = disp7

Halfword access and word access = disp8

(4) Bit addressing

This addressing is used to access 1 bit (specified with bit#3 of 3-bit data) in 1 byte of the memory space to be manipulated by using an operand address which is the sum of the contents of a general-purpose register and a 16-bit displacement sign-extended to word length. This addressing mode applies only to bit manipulation instructions.

Figure 4-7. Bit Addressing



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

CHAPTER 5 INSTRUCTIONS

5.1 Instruction Format

The V850 Series has two types of instruction formats: 16-bit and 32-bit. The 16-bit instructions include binary operation, control, and conditional branch instructions, and the 32-bit instructions include load/store, jump, and instructions that handle 16-bit immediate data.

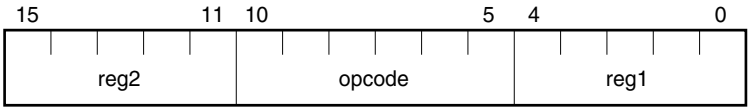
Some instructions have an unused field (RFU). This field is reserved for future expansion and must be fixed to 0.

An instruction is actually stored in memory as follows.

- Lower bytes of instruction (including bit 0) → Lower address
- Higher bytes of instruction (including bit 15 or bit 31) → Higher address

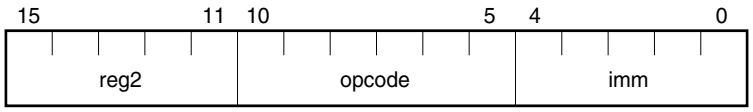
(1) reg-reg instruction (Format I)

A 16-bit instruction format having a 6-bit opcode field and two general-purpose register specification fields for operand specification.



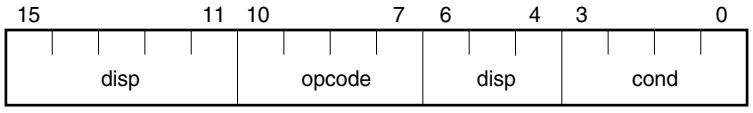
(2) imm-reg instruction (Format II)

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



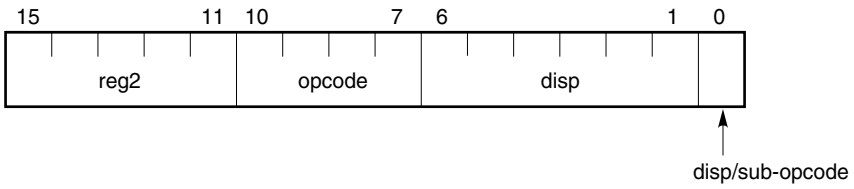
(3) Conditional branch instruction (Format III)

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

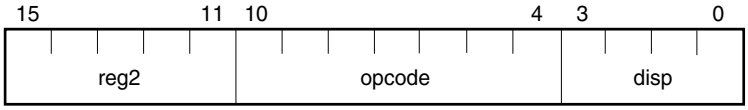


(4) 16-bit load/store instruction (Format IV)

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

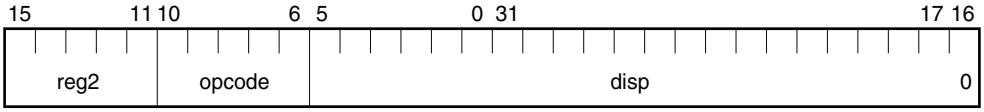


A 16-bit instruction format having a 7-bit opcode field, a general-purpose register specification field, and 4-bit displacement.



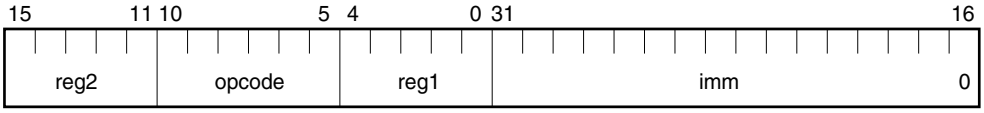
(5) Jump instruction (Format V)

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



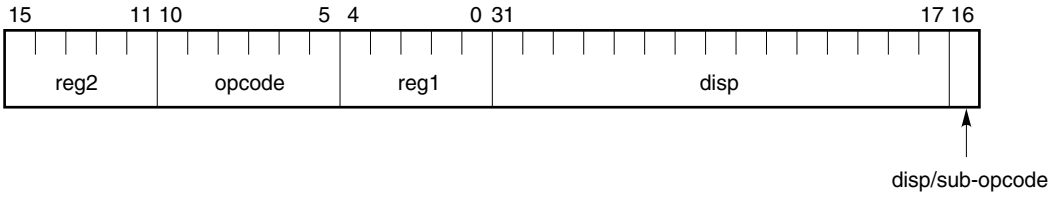
(6) 3-operand instruction (Format VI)

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



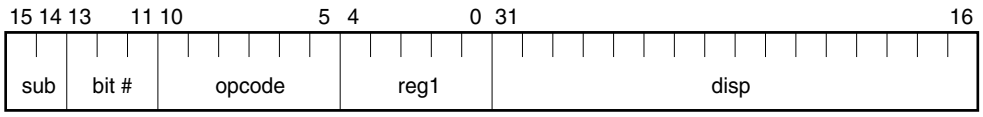
(7) 32-bit load/store instruction (Format VII)

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



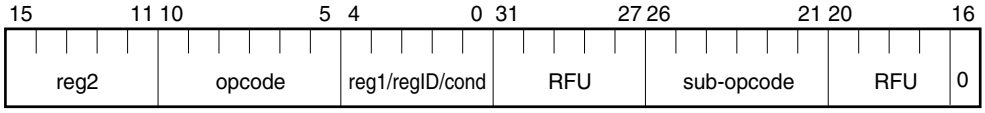
(8) Bit manipulation instruction (Format VIII)

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



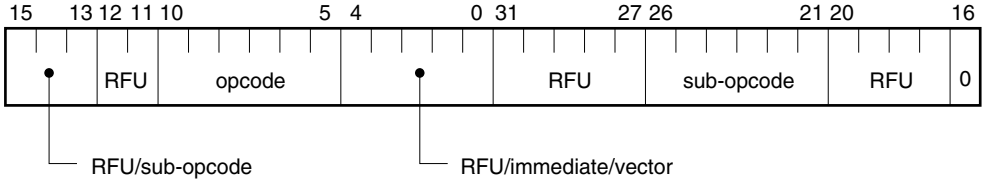
(9) Extended instruction format 1 (Format IX)

A 32-bit instruction format having a 6-bit opcode field, a 6-bit sub-opcode, and two general-purpose register specification fields (one field may be regID or cond).



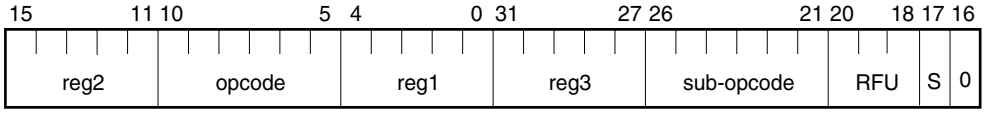
(10) Extended instruction format 2 (Format X)

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



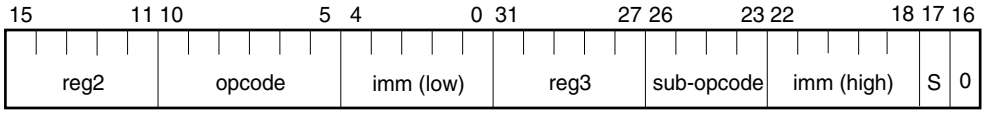
(11) Extended instruction format 3 (Format XI)

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



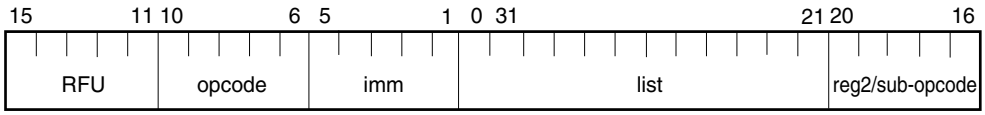
(12) Extended instruction format 4 (Format XII)

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



(13) Stack manipulation instruction (Format XIII)

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



Remark RFU: Reserved field (Reserved for Future Use)

5.2 Outline of Instructions

Load/store instructions ..... Transfer data from memory to a register or from a register to memory.

Table 5-1. Load/Store Instructions

SLD
LD
SST
ST

Arithmetic operation instructions ..... Add, subtract, multiply, divide, transfer, or compare data between registers.

Table 5-2. Arithmetic Operation Instructions

MOV
MOVHI
MOVEA
ADD
ADDI
SUB
SUBR
MUL
MULH
MULHI
MULU
DIV
DIVH
DIVHU
DIVU
CMP
CMOV
SETF
SASF

Saturated operation instructions..... Execute saturation addition or subtraction. If the result of the operation exceeds the maximum positive value (7FFFFFFFH), 7FFFFFFFH is returned. If the result exceeds the negative value (80000000H), 80000000H is returned.

Table 5-3. Saturated Operation Instructions

SATADD
SATSUB
SATSUBI
SATSUBR

Logical operation instructions..... These instructions include logical operation instructions, shift instructions and data type transfer. The shift instructions include arithmetic shift and logical shift instructions. Operands can be shifted by two or more bit positions in one clock cycle by the universal barrel shifter.

Table 5-4. Logical Operation Instructions

TST
OR
ORI
AND
ANDI
XOR
XORI
NOT
SHL
SHR
SAR
ZXB
ZXH
SXB
SXH
BSH
BSW
HSW

Branch instructions ..... Branch instruction include unconditional branch along with conditional branch instructions which alter the flow of control, depending on the status of conditional flags in the PSW. Program control can be transferred to the address specified by a branch instruction.

Table 5-5. Branch Instructions

JMP
JR
JARL
BGT
BGE
BLT
BLE
BH
BNL
BL
BNH
BE
BNE
BV
BNV
BN
BP
BC
BNC
BZ
BNZ
BR
BSA

Bit manipulation instructions..... Execute a logical operation to bit data in memory. Only the specified bit is affected as a result of executing a bit manipulation instruction.

Table 5-6. Bit Manipulation Instructions

SET1
CLR1
NOT1
TST1

Special instructions..... These instructions are special in that they do not fall into any of the categories of instructions described above.

Table 5-7. Special Instructions

LDSR
STSR
SWITCH
PREPARE
DISPOSE
CALLT
CTRET
TRAP
RETI
HALT
DI
EI
NOP

5.3 Instruction Set

Example of instruction description

Mnemonic of instruction

Meaning of instruction

Instruction format Indicates the description and operand of the instruction. The following symbols are used in the description of an operand.

Symbol	Meaning
reg1	General-purpose register (used as source register)
reg2	General-purpose register (mainly used as destination register. Some are also used as source registers)
reg3	General-purpose register (mainly used as remainder or higher 32 bits of multiply results)
bit#3	3-bit data for specifying bit number
imm×	×-bit immediate
disp×	×-bit displacement
regID	System register number
vector	5-bit data for trap vector (00H to1FH) specification
cccc	4-bit data for condition code specification
ep	Element Pointer (r30)
list×	Lists of registers (× is the maximum number of registers)

Operation Describes the function of the instruction. The following symbols are used.

Symbol	Meaning
←	Assignment
GR [ ]	General-purpose register
SR [ ]	System register
zero-extend (n)	Zero-extends n to word
sign-extend (n)	Sign-extends n to word
load-memory (a, b)	Reads data of size b from address a
store-memory (a, b, c)	Writes data b of size c to address a
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
saturated (n)	Performs saturation processing of n. If n ≥ 7FFFFFFFH as result of calculation, 7FFFFFFFH. If n ≤ 80000000H as result of calculation, 80000000H.
result	Reflects result on flag
Byte	Byte (8 bits)
Half-word	Halfword (16 bits)
Word	Word (32 bits)
+	Add
–	Subtract
||	Bit concatenation
×	Multiply
÷	Divide
%	Remainder (Divide)
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

Format Indicates instruction format number.

Opcode Describes the separate bit fields of the instruction opcode.
The following symbols are used.

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
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
bbb	3-bit data for bit number specification
L	1-bit data of code specifying register list

Flag Indicates the flags that are altered after executing the instruction.

CY – ← Indicates that the flag is not affected.
OV 0 ← Indicates that the flag is cleared to 0.
S 1 ← Indicates that the flag is set to 1.
Z –
SAT –

Instruction Describes the function of the instruction.

Explanation Explains the operation of the instruction.

Remark Supplementary information on the instruction

Caution Important cautions regarding use of this instruction

Instruction List

Mnemonic	Function
	Load/store instructions
SLD.B	Load Byte
SLD.H	Load Half-word
SLD.W	Load Word
SLD.BU	Load Byte Unsigned
SLD.HU	Load Half-word Unsigned
LD.B	Load Byte
LD.H	Load Half-word
LD.W	Load Word
LD.BU	Load Byte Unsigned
LD.HU	Load Half-word Unsigned
SST.B	Store Byte
SST.H	Store Half-word
SST.W	Store Word
ST.B	Store Byte
ST.H	Store Half-word
ST.W	Store Word
	Arithmetic instructions
MOV	Move
MOVHI	Move High half-word
MOVEA	Move Effective Address
ADD	Add
ADDI	Add Immediate
SUB	Subtract
SUBR	Subtract Reverse
MUL	Multiply Word
MULH	Multiply Half-word
MULHI	Multiply Half-word Immediate
MULU	Multiply Word Unsigned
DIV	Divide Word
DIVH	Divide Half-word
DIVHU	Divide Half-word Unsigned
DIVU	Divide Word Unsigned
CMP	Compare
CMOV	Conditional Move
SETF	Set Flag Condition
SASF	Shift And Set Flag Condition
	Saturate instructions
SATADD	Saturated Add
SATSUB	Saturated Subtract
SATSUBI	Saturated Subtract Immediate
SATSUBR	Saturated Subtract Reverse

Mnemonic	Function
	Logical operation instructions
TST	Test
OR	Or
ORI	Or Immediate
AND	And
ANDI	And Immediate
XOR	Exclusive-Or
XORI	Exclusive-Or Immediate
NOT	Not
SHL	Shift Logical Left
SHR	Shift Logical Right
SAR	Shift Arithmetic Right
ZXB	Zero Extend Byte to Word
ZXH	Zero Extend Half-word to Word
SXB	Sign Extend Byte to Word
SXH	Sign Extend Half-word to Word
BSH	Byte Swap Half-word
BSW	Byte Swap Word
HSW	Half-word Swap Word
	Branch instructions
JMP	Jump
JR	Jump Relative
JARL	Jump and Register Link
Bcond	Branch on Condition Code
	Bit manipulation instructions
SET1	Set Bit
CLR1	Clear Bit
NOT1	Not Bit
TST1	Test Bit
	Special instructions
LDSR	Load System Register
STSR	Store System Register
SWITCH	Jump with Table Look Up
PREPARE	Function Initial Operation
DISPOSE	Function Close Operation
CALLT	Call with Table Look Up
CTRET	Return from CALLT
TRAP	Trap
RETI	Return from Trap or Interrupt
HALT	Halt
DI	Disable Interrupt
EI	Enable Interrupt
NOP	No Operation

ADD

Add

Instruction format
(1) ADD reg1, reg2
(2) ADD imm5, reg2

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

Format
(1) Format I
(2) Format II

Opcode

(1)

15	0
rrrrr001110RRRRR	

(2)

15	0
rrrrr010010iiiii	

Flag

CY	1 if a carry occurs from MSB; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise 0.
SAT	–

Instruction
(1) ADD Add Register
(2) ADD Add Immediate (5-bit)

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

ADDI

Add Immediate

Instruction format ADDI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110000RRRRR	iiiiiiiiiiiiiiii

Flag

CY	1 if a carry occurs from MSB; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise 0.
SAT	–

Instruction ADDI Add immediate

Explanation Adds 16-bit immediate data, sign-extended to word length, to the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

AND

And

Instruction format AND reg1, reg2

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

Format Format I

Opcode

15															0
r	r	r	r	r	0	0	1	0	1	0	R	R	R	R	R

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise 0.
SAT	–

Instruction AND And

Explanation ANDs the word data of general-purpose register reg2 with the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

ANDI

And Immediate

Instruction format ANDI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110110RRRRR	iiiiiiiiiiiiiiii

Flag

CY	–
OV	0
S	0
Z	1 if the result of an operation is 0; otherwise 0.
SAT	–

Instruction ANDI And Immediate (16-bit)

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

Bcond

Branch on Condition Code

Instruction format Bcond disp9

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

Format Format III

Opcode

15	0
ddddd1011dddcccc	

dddddddd is the higher 8 bits of disp9.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction Bcond Branch on Condition Code with 9-bit displacement

Explanation Tests the condition flag specified by the instruction. Branches if the specified condition is satisfied; otherwise, executes the next instruction. The branch destination PC holds the sum of the current PC value and 9-bit displacement, which is 8-bit immediate shifted 1 bit and sign-extended to word length.

Remark 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. If the displacement value is 0, therefore, the branch destination is this instruction itself.

Table 5-8. Conditional Branch Instructions

Instruction		Condition Code (cccc)	Status of Condition Flag	Branch Condition
Signed integer	BGT	1111	((S xor OV) or Z) = 0	Greater than signed
	BGE	1110	(S xor OV) = 0	Greater than or equal signed
	BLT	0110	(S xor OV) = 1	Less than signed
	BLE	0111	((S xor OV) or Z) = 1	Less than or equal signed
Unsigned integer	BH	1011	(CY or Z) = 0	Higher (Greater than)
	BNL	1001	CY = 0	Not lower (Greater than or equal)
	BL	0001	CY = 1	Lower (Less than)
	BNH	0011	(CY or Z) = 1	Not higher (Less than or equal)
Common	BE	0010	Z = 1	Equal
	BNE	1010	Z = 0	Not equal
Others	BV	0000	OV = 1	Overflow
	BNV	1000	OV = 0	No overflow
	BN	0100	S = 1	Negative
	BP	1100	S = 0	Positive
	BC	0001	CY = 1	Carry
	BNC	1001	CY = 0	No carry
	BZ	0010	Z = 1	Zero
	BNZ	1010	Z = 0	Not zero
	BR	0101	–	Always (unconditional)
	BSA	1101	SAT = 1	Saturated

Caution If executing a conditional branch instruction of a signed integer (BGT, BGE, BLT, or BLE) when the SAT flag is set to 1 as a result of executing a saturated operation instruction, the branch condition loses its meaning. In ordinary arithmetic operations, if an overflow condition occurs, the S flag is inverted ($0 \rightarrow 1$ or $1 \rightarrow 0$). This is because the result is a negative value if it exceeds the maximum positive value and it is a positive value if it exceeds the maximum negative value.

However, when a saturated operation instruction is executed, and if the result exceeds the maximum positive value, the result is saturated with a positive value; if the result exceeds the maximum negative value, the result is saturated with a negative value. Unlike the ordinary operation, therefore, the S flag is not inverted even if an overflow occurs.

Hence, the S flag of the PSW is affected differently when the instruction is a saturate operation, as opposed to an ordinary arithmetic operation. A branch condition which is an XOR of the S and OV flags will therefore have no meaning.

BSH

Byte Swap 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

Opcode

15 0	31 16
rrrrr11111100000	wwwww01101000010

Flag

CY	1 if one or more bytes in result halfword is 0; otherwise 0.
OV	0
S	1 if the result of the operation is negative; otherwise, 0.
Z	1 if the result of the operation is 0; otherwise, 0.
SAT	–

Instruction BSH Byte Swap Half-word

Explanation Endian translation.

BSW

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	wwwww01101000000

Flag

CY	1 if one or more bytes in result word is 0; otherwise 0.
OV	0
S	1 if the result of the operation is negative; otherwise, 0.
Z	1 if the result of the operation is 0; otherwise, 0.
SAT	–

Instruction BSW Byte Swap Word

Explanation Endian translation.

CALLT

Call with Table Look Up

Instruction format CALLT imm6

Operation
CTPC ← PC + 2 (restore PC)
CTPSW ← PSW
adr ← CTBP + zero-extend (imm6 logically shift left by 1)
PC ← CTBP + zero-extend (Load-memory (adr, Half-word))

Format Format II

Opcode

15	0
0000001000iiiiii	

Flag
CY –
OV –
S –
Z –
SAT –

Instruction CALLT Call with Table Look Up

Explanation
(1) Transfers the restore PC and PSW to CTPC and CTPSW.
(2) Adds the CTBP and data of imm6, logically shifted left by 1 and zero-extended to word length, to generate a 32-bit table entry address.
(3) Then loads the halfword entry data and zero-extends to word length.
(4) Adds the data and CTBP to generate a 32-bit target address.
(5) Then jumps it to the target address generated in (4).

CLR1

Clear Bit

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

Operation (1) adr ← GR [reg1] + sign-extend (disp16)
Z flag ← Not (Load-memory-bit (adr, bit#3))
Store-memory-bit (adr, bit#3, 0)
(2) adr ← GR [reg1]
Z flag ← Not (Load-memory-bit (adr, reg2))
Store-memory-bit (adr, reg2, 0)

Format (1) Format VIII
(2) Format IX

Opcode

	15 0	31 16
(1)	10bbb111110RRRRR	ddddddddddddddddd

	15 0	31 16
(2)	rrrrr111111RRRRR	0000000011100100

Flag

CY	–
OV	–
S	–
Z	1 if bit specified by operands = 0. 0 if bit specified by operands = 1.
SAT	–

Instruction CLR1 Clear Bit

★ **Explanation** (1) Adds the 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 data, clears the bit specified by the bit number of bit 3, and writes the data to the former address.
(2) Reads the 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 the data to the former address.

Remark The Z flag of the PSW indicates whether the specified bit was a 0 or 1 before this instruction was executed. It does not indicate the contents of the specified bit after this instruction has been executed.

CMOV

Conditional Move

Instruction format

(1) CMOV cccc, reg1, reg2, reg3
(2) CMOV cccc, imm5, reg2, reg3

Operation

(1) if conditions are satisfied
then GR [reg3] ← GR [reg1]
else GR [reg3] ← GR [reg2]
(2) if conditions are satisfied
then GR [reg3] ← sign-extend (imm5)
else GR [reg3] ← GR [reg2]

Format

(1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww011001cccc0

(2)

15 0	31 16
rrrrr111111iiiii	wwwww011000cccc0

Flag

CY –
OV –
S –
Z –
SAT –

Instruction

CMOV Conditional Move

Explanation

(1) The data of general-purpose register reg1 is transferred to general-purpose register reg3 if the condition specified by condition code “cccc” is satisfied; otherwise, the data of general-purpose register reg2 is transferred. One of the codes shown in **Table 5-9 Condition Codes** should be specified as the condition code “cccc”.
(2) The data of 5-bit immediate, sign-extended to word length, is transferred to general-purpose register reg3 if the condition specified by condition code “cccc” is satisfied; otherwise, the data of general-purpose register reg2 is transferred. One of the codes shown in **Table 5-9 Condition Codes** should be specified as the condition code “cccc”.

Remark

See SETF Pages.

CMP

Compare

Instruction format (1) CMP reg1, reg2
(2) CMP imm5, reg2

Operation (1) result ← GR [reg2] – GR [reg1]
(2) result ← GR [reg2] – sign-extend (imm5)

Format (1) Format I
(2) Format II

Opcode

(1)

15	0
rrrrr001111RRRRR	

(2)

15	0
rrrrr010011iiiii	

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 overflow occurs; otherwise 0.
S	1 if the result of the operation is negative; otherwise, 0.
Z	1 if the result of the operation is 0; otherwise, 0.
SAT	–

Instruction (1) CMP Compare Register
(2) CMP Compare Immediate (5-bit)

Explanation

(1) Compares the word data of general-purpose register reg2 with the word data of general-purpose register reg1, and indicates the result by using the condition flags. To compare, the contents of general-purpose register reg1 are subtracted from the word data of general-purpose register reg2. The data of general-purpose registers reg1 and reg2 are not affected.

(2) Compares the word data of general-purpose register reg2 with 5-bit immediate data, sign-extended to word length, and indicates the result by using the condition flags. To compare, the contents of the sign-extended immediate data are subtracted from the word data of general-purpose register reg2. The data of general-purpose register reg2 is not affected.

CTRET

Return from CALLT

Instruction format CTRET

Operation
PC ← CTPC
PSW ← CTPSW

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101000100

Flag
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.

Instruction CTRET Return from CALLT

Explanation This instruction restores the restore PC and PSW from the appropriate system register and returns from a routine called by CALLT. The operations of this instruction are as follows.

(1) The restore PC and PSW are read from CTPC and CTPSW.
(2) Once the PC and PSW are restored in the return values, control is transferred to the return address.

DI

Disable Interrupt

Instruction format DI

Operation PSW.ID ← 1 (Disables maskable interrupt)

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101100000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–
ID	1

Instruction DI Disable Interrupt

Explanation Sets the ID flag of the PSW to 1 to disable the acknowledgement of maskable interrupts during execution of this instruction.

Remark Interrupts are not sampled during execution of this instruction. The ID flag actually becomes valid at the start of the next instruction. But because interrupts are not sampled during instruction execution, interrupts are immediately disabled. Non-maskable interrupts are not affected by this instruction.

DISPOSE

Function Dispose

Instruction format

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

Operation

(1) sp ← sp + zero-extend (imm5 logically shift left by 2)
GR [reg in list12] ← Load-memory (sp, Word)
sp ← sp + 4
repeat 2 steps above until all regs in list12 are loaded
(2) sp ← sp + zero-extend (imm5 logically shift left by 2)
GR [reg in list12] ← Load-memory (sp, Word)
sp ← sp + 4
repeat 2 states above until all regs in list12 are loaded
PC ← GR[reg1]

Format

Format XIII

Opcode

	15 0	31 16
(1)	0000011001iiiiiL	LLLLLLLLLLL00000

	15 0	31 16
(2)	0000011001iiiiiL	LLLLLLLLLLLRRRRR

RRRRR must not be 00000.
The bit assignment of list12 is shown below

15 0	31 28	27 24	23 21 16
---- ---- ---- ---3	2222	2222	223- ----
---- ---- ---- ---0	4567	0123	891- ----

Flag

CY –
OV –
S –
Z –
SAT –

Instruction

DISPOSE Function Dispose

Explanation

(1) Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pops the (loads data from the address specified by sp and adds 4 to sp) general-purpose registers listed in list12. Bit 0 of the address is masked by 0.
(2) Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pops (loads data from the address specified by sp and adds 4 to sp) the general-purpose registers listed in list12, and transfers control to the address specified by general-purpose register reg1. Bit 0 of the address is masked by 0.

Remark General-purpose registers in list12 are loaded in the downward direction. (r31, r30, ... r20)
The 5-bit immediate imm5 is used to restore a stack frame for auto variables and temporary data.
The lower 2 bits of the address specified by sp are always masked by 0 even if misalign access is enabled.
If an interrupt occurs while this instruction is being executed, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, execution is restarted. Also, sp will retain its original value prior to the start of execution.

DIV

Divide Word

Instruction format DIV reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01011000000

Flag
CY –
OV 1 if overflow occurs; otherwise, 0.
S 1 if the result of an operation is negative; otherwise, 0.
Z 1 if the result of an operation is 0; otherwise, 0.
SAT –

Instruction DIV Divide Word

Explanation Divides the word data of general-purpose register reg2 by the word data of general-purpose register reg1, and stores the quotient in general-purpose register reg2, and the remainder in general-purpose register reg3. If the data is divided by 0, an overflow occurs, and the quotient is undefined. The data of general-purpose register reg1 is not affected.

Remark An overflow occurs when the maximum negative value (80000000H) is divided by –1 (in which case the quotient is 80000000H) and when data is divided by 0 (in which case the quotient is undefined).
If an interrupt occurs while this instruction is being executed, division is aborted, and the interrupt is processed. Upon returning from the interrupt, the division is restarted from the beginning, with the return address being the address of this instruction. Also, general-purpose registers reg1 and reg2 will retain their original values prior to the start of execution.
If the address of reg2 is the same as the address of reg3, the remainder is stored in reg2 (=reg3).

DIVH

Divide Half-word

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

Operation (1) GR [reg2] ← GR [reg2] ÷ GR [reg1]
(2) GR [reg2] ← GR [reg2] ÷ GR [reg1]
GR [reg3] ← GR [reg2] % GR [reg1]

Format (1) Format I
(2) Format XI

Opcode

(1)

15	0
rrrrr000010RRRRR	

(2)

15	0	31	16
rrrrr111111RRRRR		wwwww01010000000	

Flag

CY	–
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction DIVH Divide Half-word

Explanation

(1) Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1, and stores the quotient in general-purpose register reg2. If the data is divided by 0, an overflow occurs, and the quotient is undefined.
The data of general-purpose register reg1 is not affected.

(2) Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1, and stores the quotient in general-purpose register reg2, and the remainder in general-purpose register reg3. If the data is divided by 0, an overflow occurs, and the quotient is undefined. The data of general-purpose register reg1 is not affected.

Remark

(1) The remainder is not stored.
An overflow occurs when the maximum negative value (80000000H) is divided by –1 (in which case the quotient is 80000000H) and when data is divided by 0 (in which case the quotient is undefined).
If an interrupt occurs while this instruction is being executed, division is aborted, and the interrupt is processed. Upon returning from the interrupt, the division is restarted from the beginning, with the return address being the address of this instruction. Also, general-purpose registers reg1 and reg2 will retain their original values prior to the start of

execution.

Do not specify r0 as the destination register reg2.

The higher 16 bits of general-purpose register reg1 are ignored when division is executed.

(2) An overflow occurs when the maximum negative value (80000000H) is divided by –1 (in which case the quotient is 80000000H) and when data is divided by 0 (in which case the quotient is undefined).

If an interrupt occurs while this instruction is being executed, division is aborted, and the interrupt is processed. Upon returning from the interrupt, the division is restarted from the beginning, with the return address being the address of this instruction. Also, general-purpose registers reg1 and reg2 will retain their original values prior to the start of execution.

The higher 16 bits of general-purpose register reg1 are ignored when division is executed.

If the address of reg2 is the same as the address of reg3, the remainder is stored in reg2 (=reg3).

DIVHU

Divide Half-word Unsigned

Instruction format DIVHU reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01010000010

Flag
CY –
OV 1 if overflow occurs; otherwise, 0.
S 1 if the result of an operation is negative; otherwise, 0.
Z 1 if the result of an operation is 0; otherwise, 0.
SAT –

Instruction DIVH Divide Half-word Unsigned

Explanation Divides the word data of general-purpose register reg2 by the lower halfword data of general-purpose register reg1, and stores the quotient in general-purpose register reg2, and the remainder in general-purpose register reg3. If the data is divided by 0, an overflow occurs, and the quotient is undefined. The data of general-purpose register reg1 is not affected.

Remark An overflow occurs when data is divided by 0 (in which case the quotient is undefined).
If an interrupt occurs while this instruction is being executed, division is aborted, and the interrupt is processed. Upon returning from the interrupt, the division is restarted from the beginning, with the return address being the address of this instruction. Also, general-purpose registers reg1 and reg2 will retain their original values prior to the start of execution.
If the address of reg2 is the same as the address of reg3, the remainder is stored in reg2 (=reg3).

DIVU

Divide Word Unsigned

Instruction format DIVU reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01011000010

Flag
CY –
OV 1 if overflow occurs; otherwise, 0.
S 1 if the result of an operation is negative; otherwise, 0.
Z 1 if the result of an operation is 0; otherwise, 0.
SAT –

Instruction DIVH Divide Word Unsigned

Explanation Divides the word data of general-purpose register reg2 by the word data of general-purpose register reg1, and stores the quotient in general-purpose register reg2, and the remainder to general-purpose register reg3. If the data is divided by 0, overflow occurs, and the quotient is undefined. The data of general-purpose register reg1 is not affected.

Remark An overflow occurs when data is divided by 0 (in which case the quotient is undefined).
If an interrupt occurs while this instruction is being executed, division is aborted, and the interrupt is processed. Upon returning from the interrupt, the division is restarted from the beginning, with the return address being the address of this instruction. Also, general-purpose registers reg1 and reg2 will retain their original values prior to the start of execution.
If the address of reg2 is the same as the address of reg3, the remainder is stored in reg2 (=reg3).

EI

Enable Interrupt

Instruction format EI

Operation PSW.ID ← 0 (enables maskable interrupt)

Format Format X

Opcode

15 0	31 16
1000011111100000	0000000101100000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–
ID	0

Instruction EI Enable Interrupt

Explanation Resets the ID flag of the PSW to 0 and enables the acknowledgement of maskable interrupts beginning at the next instruction.

Remark Interrupts are not sampled during instruction execution.

HALT

Halt

Instruction format HALT

Operation. Halts

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000100100000

Flag

CY –
OV –
S –
Z –
SAT –

Instruction HALT Halt

Explanation Stops the operating clock of the CPU and places the CPU in the HALT mode.

Remark The HALT mode is released by any of the following three events.

- RESET input
- NMI input
- Maskable interrupt request that is not masked

If an interrupt is acknowledged during the HALT mode, the address of the following instruction is stored in EIPC or FEPC.

HSW

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	wwwww01101000100

Flag

CY	1 if one or more halfwords in result word is 0; otherwise 0.
OV	0
S	1 if the result of the operation is negative; otherwise, 0.
Z	1 if the result of the operation is 0; otherwise, 0.
SAT	–

Instruction HSW Half-word Swap Word

Explanation Endian translation.

JARL

Jump and Register Link

Instruction format JARL disp22, reg2

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

Format Format V

Opcode

15 0	31 16
rrrrr11110dddddd	ddddddddddddddd0

ddddddddddddddddddddd is the higher 21 bits of disp22.

Flag
CY –
OV –
S –
Z –
SAT –

Instruction JARL Jump and Register Link

Explanation Saves the current PC value plus 4 to general-purpose register reg2, adds the current PC value and 22-bit displacement, sign-extended to word length, and transfers control to the PC. Bit 0 of the 22-bit displacement is masked by 0.

Remark The current PC value used for calculation is the address of the first byte of this instruction. If the displacement value is 0, the branch destination is this instruction itself.
This instruction is equivalent to a call subroutine instruction, and stores the restore PC address in general-purpose register reg2. The JMP instruction, which is equivalent to a subroutine-return instruction, can be used to specify the general-purpose register storing the restore PC as general-purpose register reg1.

JMP

Jump Register

Instruction format JMP [reg1]

Operation PC ← GR [reg1]

Format Format I

Opcode

15	0
00000000011RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction JMP Jump Register

Explanation Transfers control to the address specified by general-purpose register reg1. Bit 0 of the address is masked by 0.

Remark When using this instruction as the subroutine-return instruction, specify the restore PC using general-purpose register reg1. When using the JARL instruction, which is equivalent to the subroutine-call instruction, store the restore PC address in general-purpose register reg2.

JR

Jump Relative

Instruction format JR disp22

Operation PC ← PC + sign-extend (disp22)

Format Format V

Opcode

15 0	31 16
0000011110dddddd	ddddddddddddddd0

ddddddddddddddddddddd is the higher 21 bits of disp22.

Flag

CY –
OV –
S –
Z –
SAT –

Instruction JR Jump Relative

Explanation Adds the 22-bit displacement, sign-extended to word length, to the current PC value and stores the value in the PC, and then transfers control to the PC. Bit 0 of the 22-bit displacement is masked by 0.

Remark The current PC value used for the calculation is the address of the first byte of this instruction itself. Therefore, if the displacement value is 0, the jump destination is this instruction.

LD

Load

Instruction format

(1) LD.B disp16 [reg1], reg2
(2) LD.H disp16 [reg1], reg2
(3) LD.W disp16 [reg1], reg2
(4) LD.BU disp16 [reg1], reg2
(5) LD.HU disp16 [reg1], reg2

Operation

(1) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← sign-extend (Load-memory (adr, Byte))
(2) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← sign-extend (Load-memory (adr, Half-word))
(3) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← Load-memory (adr, Word)
(4) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← zero-extend (Load-memory (adr, Byte))
(5) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← zero-extend (Load-memory (adr, Half-word))

Format

Format VII

Opcode

(1)

15 0	31 16
rrrrr111000RRRRR	ddddddddddddddd

(2)

15 0	31 16
rrrrr111001RRRRR	ddddddddddddddd0

ddddddddddddddd is the higher 15 bits of disp16.

(3)

15 0	31 16
rrrrr111001RRRRR	ddddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16.

(4)

15 0	31 16
rrrrr11110bRRRRR	ddddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16, b is the bit 0 of disp 16.

(5)

15 0	31 16
rrrrr111111RRRRR	ddddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16.

Flag

CY –
OV –
S –
Z –
SAT –

Instruction

(1) LD.B Load Byte
(2) LD.H Load Half-word
(3) LD.W Load Word
(4) LD.BU Load Byte Unsigned
(5) LD.HU Load Half-word Unsigned

Explanation

(1) Adds the data of general-purpose register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and stored in general-purpose register reg2.
(2) Adds the data of general-purpose register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Halfword data is read from this 32-bit address with its bit 0 masked by 0, sign-extended to word length, and stored in general-purpose register reg2.
(3) Adds the data of general-purpose register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Word data is read from this 32-bit address with bits 0 and 1 masked by 0, and stored in general-purpose register reg2.
(4) Adds the data of general-purpose register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored in general-purpose register reg2.
Do not specify r0 as the destination register reg2.
(5) Adds the data of general-purpose register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Halfword data is read from this 32-bit address with its bit 0 masked by 0, zero-extended to word length, and stored in general-purpose register reg2.
Do not specify r0 as the destination register reg2.

Caution

The result of adding the data of general-purpose register reg1 and the 16-bit displacement sign-extended to word length is as follows.

- Lower bits are not masked and address is generated.

LDSR

Load to System Register

Instruction format LDSR reg2, regID

Operation SR [regID] ← GR [reg2]

Format Format IX

Opcode

15 0	31 16
rrrrr111111RRRRR	0000000000100000

Remark The fields used to define reg1 and reg2 are swapped in this instruction. Normally, "RRR" is used for reg1 and is the source operand while “rrr” signifies reg2 and is the destination operand. In this instruction, “RRR” is still the source operand, but is represented by reg2, while “rrr” is the special register destination, as labeled below.

rrrrr: regID specification
RRRRR: reg2 specification

Flag

CY – (Refer to **Remark** below.)
OV – (Refer to **Remark** below.)
S – (Refer to **Remark** below.)
Z – (Refer to **Remark** below.)
SAT – (Refer to **Remark** below.)

Instruction LDSR Load to System Register

Explanation Loads the word data of general-purpose register reg2 to a system register specified by the system register number (regID). The data of general-purpose register reg2 is not affected.

Remark If the system register number (regID) is equal to 5 (PSW register), the values of the corresponding bits of the PSW are set according to the contents of reg2. This only affects the flag bits, and the reserved bits remain 0. Also, interrupts are not sampled when the PSW is being written with a new value. If the ID flag is enabled with this instruction, interrupt disabling begins at the start of execution, even though the ID flag does not become valid until the beginning of the next instruction.

Caution The system register number regID is a number which identifies a system register. Accessing system registers which are reserved or write-prohibited is prohibited and will lead to undefined results.

MOV

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
rrrrr010000iiiii	

(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.

Flag
CY –
OV –
S –
Z –
SAT –

Instruction
(1) MOV Move Register
(2) MOV Move Immediate (5-bit)
(3) MOV Move Immediate (32-bit)

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

MOVEA

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	iiiiiiiiiiiiiiii

Flag

CY –
OV –
S –
Z –
SAT –

Instruction MOVEA Move Effective Address

Explanation Adds the 16-bit immediate data, sign-extended to word length, to the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected. The flags are not affected by the addition.
Do not specify r0 as the destination register reg2.

Remark This instruction calculates a 32-bit address and stores the result without affecting the PSW flags.

MOVHI

Move High Half-word

Instruction format MOVHI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110010RRRRR	iiiiiiiiiiiiiiii

Flag

CY –
OV –
S –
Z –
SAT –

Instruction MOVHI Move High half-word

Explanation Adds a word value, whose higher 16 bits are specified by the 16-bit immediate data and lower 16 bits are 0, to the word data of general-purpose register reg1 and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected. The flags are not affected by the addition.
Do not specify r0 as the destination register reg2.

Remark This instruction is used to generate the higher 16 bits of a 32-bit address.

MUL

Multiply Word

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

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

Format (1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww01000100000

(2)

15 0	31 16
rrrrr111111iiiii	wwwww01001IIII00

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

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction (1) MUL Multiply Word by Register
(2) MUL Multiply Word by Immediate (9-bit)

Explanation (1) Multiplies the word data of general-purpose register reg2 by the word data of general-purpose register reg1, and stores the result in general-purpose register reg2 and reg3 as double word data. The data of general-purpose register reg1 is not affected.
(2) Multiplies the word data of general-purpose register reg2 by a 9-bit immediate data, sign-extended to word length, and stores the result in general-purpose registers reg2 and reg3.

Remark The higher 32 bits of the result are stored in general-purpose register reg3.
If the address of reg2 is the same as the address of reg3, the higher 32 bits of the result are stored in reg2 (=reg3)

MULH

Multiply Half-word

Instruction format
(1) MULH reg1, reg2
(2) MULH imm5, reg2

Operation
(1) GR [reg2] (32) ← GR [reg2] (16) × GR [reg1] (16)
(2) GR [reg2] ← GR [reg2] × sign-extend (imm5)

Format
(1) Format I
(2) Format II

Opcode

(1)

15	0
rrrrr000111RRRRR	

(2)

15	0
rrrrr010111iiiii	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction
(1) MULH Multiply Half-word by Register
(2) MULH Multiply Half-word by Immediate (5-bit)

Explanation
(1) Multiplies the lower halfword data of general-purpose register reg2 by the halfword data of general-purpose register reg1, and stores the result in general-purpose register reg2 as word data. The data of general-purpose register reg1 is not affected.
Do not specify r0 as the destination register reg2.
(2) Multiplies the lower halfword data of general-purpose register reg2 by a 5-bit immediate data, sign-extended to halfword length, and stores the result in general-purpose register reg2.
Do not specify r0 as the destination register reg2.

Remark The higher 16 bits of general-purpose registers reg1 and reg2 are ignored in this operation.

MULHI

Multiply Half-word Immediate

Instruction format MULHI imm16, reg1, reg2

Operation GR [reg2] ← GR [reg1] × imm16

Format Format VI

Opcode

15 0	31 16
rrrrr110111RRRRR	iiiiiiiiiiiiiiii

Flag

CY –
OV –
S –
Z –
SAT –

Instruction MULHI Multiply Half-word by immediate (16-bit)

Explanation Multiplies the lower halfword data of general-purpose register reg1 by the 16-bit immediate data, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.
Do not specify r0 as the destination register reg2.

Remark The higher 16 bits of general-purpose register reg1 are ignored in this operation.

MULU

Multiply Word Unsigned

Instruction format

(1) MULU reg1, reg2, reg3
(2) MULU imm9, reg2, reg3

Operation

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

Format

(1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww01000100010

(2)

15 0	31 16
rrrrr111111iiiii	wwwww01001IIII10

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

Flag

CY –
OV –
S –
Z –
SAT –

Instruction

(1) MULU Multiply Word by Register
(2) MULU Multiply Word by Immediate (9-bit)

Explanation

(1) Multiplies the word data of general-purpose register reg2 by the word data of general-purpose register reg1, and stores the result in general-purpose registers reg2 and reg3 as double word data. The data of general-purpose register reg1 is not affected.
(2) Multiplies the word data of general-purpose register reg2 by a 9-bit immediate data, zero-extended to word length, and stores the result in general-purpose registers reg2 and reg3.

Remark

The higher 32 bits of the result are stored in general-purpose register reg3.
If the address of reg2 is the same as the address of reg3, the higher 32 bits of the result are stored in reg2 (=reg3).

NOP

No Operation

Instruction format NOP

Operation Executes nothing and consumes at least one clock.

Format Format I

Opcode

15	0
0000000000000000	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction NOP No Operation

Explanation Executes nothing and consumes at least one clock cycle.

Remark The contents of the PC are incremented by two. The opcode is the same as that of MOV r0, r0.

NOT

Not

Instruction format NOT reg1, reg2

Operation GR [reg2] ← NOT (GR [reg1])

Format Format I

Opcode

15	0
rrrrr000001RRRRR	

Flag

CY –
OV 0
S 1 if the result of an operation is negative; otherwise, 0.
Z 1 if the result of an operation is 0; otherwise, 0.
SAT –

Instruction NOT Not

Explanation Logically negates (takes the 1's complement of) the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

NOT1

Not Bit

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

Operation (1) adr ← GR [reg1] + sign-extend (disp16)
Z flag ← Not (Load-memory-bit (adr, bit#3))
Store-memory-bit (adr, bit#3, Z flag)
(2) adr ← GR [reg1]
Z flag ← Not (Load-memory-bit (adr, reg2))
Store-memory-bit (adr, reg2, Z flag)

Format (1) Format VIII
(2) Format IX

Opcode

(1)

15 0	31 16
01bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100010

Flag
CY –
OV –
S –
Z 1 if bit specified by operands = 0.
0 if bit specified by operands = 1.
SAT –

Instruction NOT1 Not Bit

★ **Explanation** (1) Adds the 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 field “bbb”, and writes the data to the previous address.
(2) Reads the 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 the lower 3 bits of reg2, and writes the data to the previous address.

Remark The Z flag of the PSW indicates whether the specified bit was 0 or 1 before this instruction was executed, and does not indicate the contents of the specified bit after this instruction has been executed.

OR

Or

Instruction format OR reg1, reg2

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

Format Format I

Opcode

15	0
rrrrr001000RRRRR	

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction OR Or

Explanation ORs the word data of general-purpose register reg2 with the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

ORI

Or Immediate

Instruction format ORI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110100RRRRR	iiiiiiiiiiiiiiii

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction OR Or immediate (16-bit)

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

PREPARE

Function Prepare

Instruction format (1) PREPARE list12, imm5

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

Note sp / imm is specified by sub-opcode bits 20 and 19.

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	Optional(47-32 or 63-32)
(1)	00000111110iiiiiL	LLLLLLLLLLL00001	
(2)	00000111110iiiiiL	LLLLLLLLLLLff011	imm16 / imm32

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

ff = 00: Load sp to ep
01: Load 16-bit immediate data (bit 47 to 32), sign-extended, to ep
10: Load 16-bit immediate data (bit 47 to 32), logically shifted left by 16, to ep
11: Load 32-bit immediate data (bit 63 to 32) to ep

Bit assignment of list12 is below

15 0	31 28 27 24 23 21 16
---- ---- ---- ---3	2222 2222 223- ----
---- ---- ---- ---0	4567 0123 891- ----

Flag CY –
OV –
S –
Z –
SAT –

Instruction PREPARE Function Prepare

Explanation (1) Pushes (subtracts 4 from sp and stores the data in that address) the general-purpose registers listed in list12. Then subtracts the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, from sp.

(2) Pushes (subtracts 4 from sp and stores the data in that address) the general-purpose registers listed in list12. Then subtracts the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, from sp.
Next, load the data specified by 3rd operand to ep.

Remark The general-purpose registers in list12 are stored in the upward direction. (r20, r21, ... r31)
The 5-bit immediate imm5 is used to make a stack frame for auto variables and temporary data.
The lower 2 bits of the address specified by sp are always masked by 0 even if misalign access is enabled.
If an interrupt occurs while this instruction is being executed, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, execution is restarted. Also, sp and ep will retain their original values prior to the start of execution.

RETI

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

Flag

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.

Instruction RETI Return from Trap or Interrupt

Explanation This instruction reads the restore PC and PSW from the appropriate system register, and operation returns from an exception or interrupt routine. The operations of this instruction are as follows.

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

(2) Once the restore PC and PSW values are set to the PC and PSW, the operation returns to the address immediately before the trap or interrupt occurred.

Caution When returning from an NMI or exception routine using the RETI instruction, the PSW.NP and PSW.EP flags must be set accordingly to restore the PC and PSW:

- When returning from non-maskable interrupt routine using the RETI instruction:
  PSW.NP = 1 and PSW.EP = 0
- When returning from an exception routine using the RETI instruction:
  PSW.EP = 1

Use the LDSR instruction for setting the flags.

Interrupts are not acknowledged in the latter half of the ID stage during LDSR execution because of the operation of the interrupt controller.

SAR

Shift Arithmetic Right

Instruction format (1) SAR reg1, reg2
(2) SAR imm5, reg2

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

Format (1) Format IX
(2) Format II

Opcode

	15	0	31	16
(1)	rrrrr111111RRRRR		0000000010100000	

	15	0
(2)	rrrrr010101iiiii	

Flag

CY	1 if the bit shifted out last is 1; otherwise, 0. However, if the number of shifts is 0, the result is 0.
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction (1) SAR Shift Arithmetic Right by Register
(2) SAR Shift Arithmetic Right by Immediate (5-bit)

Explanation (1) Arithmetically shifts the word data of general-purpose register reg2 to the right by 'n' positions, where 'n' is a value from 0 to +31, specified by the lower 5 bits of general-purpose register reg1 (after the shift, the MSB prior to shift execution is copied and set as the new MSB value), and then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the same value prior to instruction execution. The data of general-purpose register reg1 is not affected.

(2) Arithmetically shifts the word data of general-purpose register reg2 to the right by 'n' positions, where 'n' is a value from 0 to +31, 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), and then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the same value prior to instruction execution.

SASF

Shift and Set Flag Condition

Instruction format SASF cccc, reg2

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

Format Format IX

Opcode

15 0	31 16
rrrrr1111110cccc	0000001000000000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction SASF Shift And Set Flag Condition

Explanation General-purpose register reg2 is logically shifted left by 1, and its LSB is set to 1 if the condition specified by condition code “cccc” is satisfied; otherwise, the LSB is set to 0. One of the codes shown in **Table 5-9 Condition Codes** should be specified as the condition code “cccc”.

Remark See SETF Pages.

SATADD

Saturated Add

Instruction format
(1) SATADD reg1, reg2
(2) SATADD imm5, reg2

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

Format
(1) Format I
(2) Format II

Opcode

(1)

15	0
rrrrr000110RRRRR	

(2)

15	0
rrrrr010001iiiii	

Flag
CY 1 if a carry occurs from MSB; otherwise, 0.
OV 1 if overflow occurs; otherwise, 0.
S 1 if the result of the saturated operation is negative; otherwise, 0.
Z 1 if the result of the saturated operation is 0; otherwise, 0.
SAT 1 if OV = 1; otherwise, not affected.

Instruction
(1) SATADD Saturated add register
(2) SATADD Saturated add Immediate (5-bit)

Explanation
(1) Adds the word data of general-purpose register reg1 to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, if the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored in reg2. The SAT flag is set to 1. The data of general-purpose register reg1 is not affected.
Do not specify r0 as the destination register reg2.
(2) Adds a 5-bit immediate data, sign-extended to word length, to the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. However, if the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored in reg2. The SAT flag is set to 1.
Do not specify r0 as the destination register reg2.

Remark
The SAT flag is a cumulative flag. Once the result of the saturated operation instruction has been saturated, this flag is set to 1 and is not reset to 0 even if the result of the subsequent operation is not saturated.
Even if the SAT flag is set to 1, the saturated operation instruction is executed normally.

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

SATSUB

Saturated Subtract

Instruction format SATSUB reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr000101RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of the saturated operation is negative; otherwise, 0.
Z	1 if the result of the saturated operation is 0; otherwise, 0.
SAT	1 if OV = 1; otherwise, not affected.

Instruction SATSUB Saturated Subtract

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

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not reset to 0 even if the result of the subsequent operations is not saturated.
Even if the SAT flag is set to 1, the saturated operation instruction is executed normally.

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

SATSUBI

Saturated Subtract Immediate

Instruction format SATSUBI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110011RRRRR	iiiiiiiiiiiiiiii

Flag

CY 1 if a borrow to MSB occurs; otherwise, 0.
OV 1 if overflow occurs; otherwise, 0.
S 1 if the result of the saturated operation is negative; otherwise, 0.
Z 1 if the result of the saturated operation is 0; otherwise, 0.
SAT 1 if OV = 1; otherwise, not affected.

Instruction SATSUBI Saturated Subtract Immediate

Explanation Subtracts the 16-bit immediate data, sign-extended to word length, from the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. However, if the result exceeds the maximum positive value 7FFFFFFFH, 7FFFFFFFH is stored in reg2; if the result exceeds the maximum negative value 80000000H, 80000000H is stored in reg2. The SAT flag is set to 1. The data of general-purpose register reg1 is not affected.
Do not specify r0 as the destination register reg2.

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not reset to 0 even if the result of the subsequent operations is not saturated.
Even if the SAT flag is set to 1, the saturated operation instruction is executed normally.

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

SATSUBR

Saturated Subtract Reverse

Instruction format SATSUBR reg1, reg2

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

Format Format I

Opcode

15	0
rrrrr000100RRRRR	

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of the saturated operation is negative; otherwise, 0.
Z	1 if the result of the saturated operation is 0; otherwise, 0.
SAT	1 if OV = 1; otherwise, not affected.

Instruction SATSUBR Saturated Subtract Reverse

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

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not reset to 0 even if the result of the subsequent operations is not saturated.
Even if the SAT flag is set to 1, the saturated operation instruction is executed normally.

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

SETF

Set Flag Condition

Instruction format SETF cccc, reg2

Operation
if conditions are satisfied
then GR [reg2] ← 00000001H
else GR [reg2] ← 00000000H

Format Format IX

Opcode

15 0	31 16
rrrrr1111110cccc	0000000000000000

Flag
CY –
OV –
S –
Z –
SAT –

Instruction SETF Set Flag Condition

Explanation General-purpose register reg2 is set to 1 if the condition specified by condition code “cccc” is satisfied; otherwise, 0 is stored in the register. One of the codes shown in **Table 5-9 Condition Codes** should be specified as the condition code “cccc”.

Remark Here are some examples of using this instruction.

(1) Translation of two or more 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 a “conditional branch” by checking the result of evaluation equivalent to a_n. Since a pipeline processor takes more time to execute “condition judgment” + “branch” than to execute an ordinary operation, the result of evaluating each condition clause if (a_n) is stored in register Ra. By performing a logical operation on Ra_n after all the condition clauses have been evaluated, the delay due to the pipeline can be prevented.

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

Table 5-9. Condition Codes

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

SET1

Set Bit

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

Operation (1) adr ← GR [reg1] + sign-extend (disp16)
Z flag ← Not (Load-memory-bit (adr, bit#3))
Store-memory-bit (adr, bit#3, 1)
(2) adr ← GR [reg1]
Z flag ← Not (Load-memory-bit (adr, reg2))
Store-memory-bit (adr, reg2, 1)

Format (1) Format VIII
(2) Format IX

Opcode

(1)

15 – 0	31 – 16
00bbb111110RRRRR	dddddddddddddddd

(2)

15 – 0	31 – 16
rrrrr111111RRRRR	0000000011100000

Flag

CY	–
OV	–
S	–
Z	1 if bit specified by operands = 0. 0 if bit specified by operands = 1.
SAT	–

Instruction SET1 Set Bit

★ **Explanation**

(1) Adds the 16-bit displacement, sign-extended to word length, to the 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 3-bit field “bbb”, and writes the data to the previous address. Bits other than the specified bit are not affected.

(2) Reads the 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 the lower 3 bits of reg2, and writes the data to the previous address. Bits other than the specified bit are not affected.

Remark The Z flag of the PSW indicates whether the specified bit was 0 or 1 before this instruction was executed, and does not indicate the content of the specified bit after this instruction has been executed.

SHL

Shift Logical Left

Instruction format (1) SHL reg1, reg2
(2) SHL imm5, reg2

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

Format (1) Format IX
(2) Format II

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	0000000011000000

(2)

15 0
rrrrr010110iiiii

Flag

CY	1 if the bit shifted out last is 1; otherwise, 0. However, if the number of shifts is 0, the result is 0.
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction (1) SHL Shift Logical Left by Register
(2) SHL Shift Logical Left by Immediate (5-bit)

Explanation

(1) Logically shifts the word data of general-purpose register reg2 to the left by 'n' positions, where 'n' is a value from 0 to +31, specified by the lower 5 bits of general-purpose register reg1 (0 is shifted to the LSB side), and then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the same value prior to instruction execution. The data of general-purpose register reg1 is not affected.

(2) Logically shifts the word data of general-purpose register reg2 to the left by 'n' positions, where 'n' is a value from 0 to +31, specified by the 5-bit immediate data, zero-extended to word length (0 is shifted to the LSB side), and then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the value prior to instruction execution.

SHR

Shift Logical Right

Instruction format
(1) SHR reg1, reg2
(2) SHR imm5, reg2

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

Format
(1) Format IX
(2) Format II

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	0000000010000000

(2)

15 0
rrrrr010100iiiii

Flag

CY	1 if the bit shifted out last is 1; otherwise, 0. However, if the number of shifts is 0, the result is 0.
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction
(1) SHR Shift Logical Right by Register
(2) SHR Shift Logical Right by Immediate (5-bit)

Explanation
(1) Logically shifts the word data of general-purpose register reg2 to the right by 'n' positions where 'n' is a value from 0 to +31, specified by the lower 5 bits of general-purpose register reg1 (0 is shifted to the MSB side). This instruction then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the same value prior to instruction execution. The data of general-purpose register reg1 is not affected.
(2) Logically shifts the word data of general-purpose register reg2 to the right by 'n' positions, where 'n' is a value from 0 to +31, specified by the 5-bit immediate data, zero-extended to word length (0 is shifted to the MSB side). This instruction then writes the result in general-purpose register reg2. If the number of shifts is 0, general-purpose register reg2 retains the same value prior to instruction execution.

SLD

Short Load

Instruction format

(1) SLD.B disp7 [ep], reg2
(2) SLD.H disp8 [ep], reg2
(3) SLD.W disp8 [ep], reg2
(4) SLD.BU disp4 [ep], reg2
(5) SLD.HU disp5 [ep], reg2

Operation

(1) adr ← ep + zero-extend (disp7)
GR [reg2] ← sign-extend (Load-memory (adr, Byte))
(2) adr ← ep + zero-extend (disp8)
GR [reg2] ← sign-extend (Load-memory (adr, Half-word))
(3) adr ← ep + zero-extend (disp8)
GR [reg2] ← Load-memory (adr, Word)
(4) adr ← ep + zero-extend (disp4)
GR [reg2] ← zero-extend (Load-memory (adr, Byte))
(5) adr ← ep + zero-extend (disp5)
GR [reg2] ← zero-extend (Load-memory (adr, Half-word))

Format

Format IV

Opcode

(1)

15															0
r	r	r	r	r	0	1	1	0	d	d	d	d	d	d	d

(2)

15															0
r	r	r	r	r	1	0	0	0	d	d	d	d	d	d	d

ddddddd is the higher 7 bits of disp8.

(3)

15															0
r	r	r	r	r	1	0	1	0	d	d	d	d	d	d	0

dddddd is the higher 6 bits of disp8.

(4)

15															0
r	r	r	r	r	0	0	0	0	1	1	0	d	d	d	d

rrrrr must not be 00000.

(5)

15															0
r	r	r	r	r	0	0	0	0	1	1	1	d	d	d	d

dddd is the higher 4 bits of disp5, rrrrr must not be 00000.

Flag

CY –
OV –
S –
Z –
SAT –

Instruction

(1) SLD.B Short format Load Byte
(2) SLD.H Short format Load Half-word
(3) SLD.W Short format Load Word
(4) SLD.BU Short format Load Byte Unsigned
(5) SLD.HU Short format Load Half-word Unsigned

Explanation

(1) 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.
(2) Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Halfword data is read from this 32-bit address with bit 0 masked by 0, sign-extended to word length, and stored in reg2.
(3) Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Word data is read from this 32-bit address with bits 0 and 1 masked by 0, and stored in reg2.
(4) 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.
Do not specify r0 as the destination register reg2.
(5) Adds the 5-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Halfword data is read from this 32-bit address with bit 0 masked by 0, zero-extended to word length, and stored in reg2.
Do not specify r0 as the destination register reg2.

Caution

(1) The result of adding the element pointer and the 8-bit displacement zero-extended to word length can be of two types depending on the type of data to be accessed (halfword, word) and the misaligned mode setting.

- Lower bits are masked by 0 and address is generated (when misalign access is disabled)
- Lower bits are not masked and address is generated (when misalign access is enabled)

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

★ (2) If an interrupt to an SLD instruction that reads from the external memory space is generated, the read value may be written to a register other than that specified by the SLD instruction. To prevent this, change all the SLD instructions that access the external memory to LD instructions.

SST

Store

Instruction format (1) SST.B reg2, disp7 [ep]
(2) SST.H reg2, disp8 [ep]
(3) SST.W reg2, disp8 [ep]

Operation (1) adr ← ep + zero-extend (disp7)
Store-memory (adr, GR [reg2], Byte)
(2) adr ← ep + zero-extend (disp8)
Store-memory (adr, GR [reg2], Half-word)
(3) adr ← ep + zero-extend (disp8)
Store-memory (adr, GR [reg2], Word)

Format Format IV

Opcode

(1)

15															0
r	r	r	r	r	0	1	1	1	d	d	d	d	d	d	d

(2)

15															0
r	r	r	r	r	1	0	0	1	d	d	d	d	d	d	d

ddddddd is the higher 7 bits of disp8.

(3)

15															0
r	r	r	r	r	1	0	1	0	d	d	d	d	d	d	1

dddddd is the higher 6 bits of disp8.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction (1) SST.B Short format Store Byte
(2) SST.H Short format Store Half-word
(3) SST.W Short format Store Word

Explanation

(1) Adds the 7-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the data of the lowest byte of reg2 in the generated address.

(2) Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the lower halfword data of reg2 in the generated 32-bit address with bit 0 masked by 0.

(3) Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the word data of reg2 in the generated 32-bit address with bits 0 and 1 masked by 0.

Cautions

(1) The result of adding the element pointer and the 8-bit displacement zero-extended to word length can be of two types depending on the type of data to be accessed (halfword, word) and the misaligned mode setting.

- Lower bits are masked by 0 and address is generated (when misalign access is disabled)
- Lower bits are not masked and address is generated (when misalign access is enabled)

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

★ (2) Branch instructions may not be correctly executed in the following instruction sequence.

Instruction 1	sst/st instruction (access to the external memory)
Instruction 2	Any instruction string other than sst/st instruction (0 or more)
Instruction 3	sst instruction
Instruction 4	bcond (bc, be, bge, bgt, bh, bl, ble, blt, bn, bnc, bne, bnh, bnl, bnv, bnz, bp, br, bsa, bv, bz) instruction

Perform either of the following to avoid the above.

- Replace the sst instruction immediately before the bcond instruction with the st instruction
- Insert a nop instruction between the bcond instruction and the sst instruction immediately before

ST

Store

Instruction format (1) ST.B reg2, disp16 [reg1]
(2) ST.H reg2, disp16 [reg1]
(3) ST.W reg2, disp16 [reg1]

Operation (1) adr ← GR [reg1] + sign-extend (disp16)
Store-memory (adr, GR [reg2], Byte)
(2) adr ← GR [reg1] + sign-extend (disp16)
Store-memory (adr, GR [reg2], Half-word)
(3) adr ← GR [reg1] + sign-extend (disp16)
Store-memory (adr, GR [reg2], Word)

Format Format VII

Opcode

(1)

15 0	31 16
rrrrr111010RRRRR	ddddddddddddddddd

(2)

15 0	31 16
rrrrr111011RRRRR	ddddddddddddddd0

ddddddddddddddd is the higher 15 bits of disp16.

(3)

15 0	31 16
rrrrr111011RRRRR	ddddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16.

Flag CY –
OV –
S –
Z –
SAT –

Instruction (1) ST.B Store Byte
(2) ST.H Store Half-word
(3) ST.W Store Word

Explanation

(1) Adds the 16-bit displacement, sign-extended to word length, to the data of general-purpose register reg1 to generate a 32-bit address, and stores the lowest byte data of general-purpose register reg2 in the generated address.

(2) Adds the 16-bit displacement, sign-extended to word length, to the data of general-purpose register reg1 to generate a 32-bit address, and stores the lower halfword data of general-purpose register reg2 in the generated 32-bit address with bit 0 masked by 0. Therefore, stored data is automatically aligned on a halfword boundary.

(3) Adds the 16-bit displacement, sign-extended to word length, to the data of general-purpose register reg1 in generate a 32-bit address, and stores the word data of general-purpose register reg2 in the generated 32-bit address with bits 0 and 1 masked by 0. Therefore, stored data is automatically aligned on a word boundary.

Caution

The result of adding the data of general-purpose register reg1 and the 16-bit displacement sign-extended to word length can be of two types depending on the type of data to be accessed (halfword, word), and the misalign mode setting.

- Lower bits are masked by 0 and address is generated (when misalign access is disabled)
- Lower bits are not masked and address is generated (when misalign access is enabled)

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

STSR

Store Contents of System Register

Instruction format STSR regID, reg2

Operation GR [reg2] ← SR [regID]

Format Format IX

Opcode

15 0	31 16
rrrrr111111RRRRR	0000000001000000

Flag

CY –
OV –
S –
Z –
SAT –

Instruction STSR Store Contents of System Register

Explanation Stores the contents of a system register specified by a system register number (regID) in general-purpose register reg2. The contents of the system register are not affected.

Remark The system register number regID is a number which identifies a system register. Accessing a system register which is reserved is prohibited and will lead to undefined results.

SUB

Subtract

Instruction format SUB reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr001101RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction SUB Subtract

Explanation Subtracts the word data of general-purpose register reg1 from the word data of general-purpose register reg2, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

SUBR

Subtract Reverse

Instruction format SUBR reg1, reg2

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

Format Format I

Opcode

15	0
rrrrr001100RRRRR	

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if overflow occurs; otherwise, 0.
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction SUBR Subtract Reverse

Explanation Subtracts the word data of general-purpose register reg2 from the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

SWITCH

Jump with Table Look Up

Instruction format SWITCH reg1

Operation
adr ← (PC + 2) + (GR[reg1] logically shift left by 1)
PC ← (PC + 2) + (sign-extend (Load-memory (adr, Half-word))) logically shift left by 1

Format Format I

Opcode

15	0
00000000010RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction Switch Jump with Table Look Up

Explanation

<1> Adds the table entry address (address following the SWITCH instruction) and data of general-purpose register reg1 logically shifted left by 1, and generates 32-bit table entry address.

<2> Loads halfword data pointed by address generated in <1>.

<3> Sign-extends the loaded halfword data to word length, and adds the table entry address after logically shifts it left by 1 bit (next address following SWITCH instruction) to generate a 32-bit target address.

<4> Then jumps to the target address generated in <3>.

SXB

Sign Extend Byte

Instruction format SXB reg1

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

Format Format I

Opcode

15	0
00000000101RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction SXB Sign Extend Byte

Explanation Sign-extends the lowest byte of general-purpose register reg1 to word length.

SXH

Sign Extend Half-word

Instruction format SXH reg1

Operation GR [reg1] ← sign-extend (GR [reg1] (15:0))

Format Format I

Opcode

15	0
00000000111RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction SXH Sign Extend Half-word

Explanation Sign-extends the lower halfword of general-purpose register reg1 to word length.

TRAP

Software Trap

Instruction format TRAP vector

Operation

```
EIPC      ← PC + 4 (restore PC)
EIPSW     ← PSW
ECR.EICC  ← interrupt code
PSW.EP    ← 1
PSW.ID    ← 1
PC        ← 00000040H (vector = 00H to 0FH)
            00000050H (vector = 10H to 1FH)
```

Format Format X

Opcode

15 0	31 16
0000011111iiiii	0000000100000000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction TRAP Trap

Explanation Saves the restore PC and PSW to EIPC and EIPSW, respectively; sets the exception code (EICC of ECR) and the flags of the PSW (EP and ID flags); jumps to the address of the trap handler corresponding to the trap vector specified by vector number (0-31), and starts exception processing. The condition flags are not affected.
The restore PC is the address of the instruction following the TRAP instruction.

TST

Test

Instruction format TST reg1, reg2

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

Format Format I

Opcode

15	0
rrrrr001011RRRRR	

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction TST Test

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

TST1

Test Bit

Instruction format

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

Operation

(1) adr ← GR [reg1] + sign-extend (disp16)
Z flag ← Not (Load-memory-bit (adr,bit#3))
(2) adr ← GR [reg1]
Z flag ← Not (Load-memory-bit (adr,reg2))

Format

(1) Format VIII
(2) Format IX

Opcode

(1)

15 – 0	31 – 16
11bbb111110RRRRR	ddddddddddddddddd

(2)

15 – 0	31 – 16
rrrrr111111RRRRR	0000000011100110

Flag

CY –
OV –
S –
Z 1 if bit specified by operands = 0.
0 if bit specified by operands = 1.
SAT –

Instruction

TST1 Test Bit

Explanation

(1) Adds the data of general-purpose register reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Performs the test on the bit specified by the 3-bit field “bbb”, at the byte data location referenced by the generated address. If the specified bit is 0, the Z flag is set to 1; if the bit is 1, the Z flag is reset to 0. The byte data, including the specified bit, is not affected.
(2) Reads the data of general-purpose register reg1 to generate a 32-bit address. Performs a test on 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, the Z flag is set to 1; if the bit is 1, the Z flag is reset to 0. The byte data, including the specified bit, is not affected.

XOR

Exclusive Or

Instruction format XOR reg1, reg2

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

Format Format I

Opcode

15	0
rrrrr001001RRRRR	

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction XOR Exclusive Or

Explanation Exclusively ORs the word data of general-purpose register reg2 with the word data of general-purpose register reg1, and stores the result in general-purpose register reg2. The data of general-purpose register reg1 is not affected.

XORI

Exclusive Or Immediate

Instruction format XORI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110101RRRRR	iiiiiiiiiiiiiiii

Flag

CY	–
OV	0
S	1 if the result of an operation is negative; otherwise, 0.
Z	1 if the result of an operation is 0; otherwise, 0.
SAT	–

Instruction XORI Exclusive Or Immediate (16-bit)

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

ZXB

Zero Extend Byte

Instruction format ZXB reg1

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

Format Format I

Opcode

15	0
00000000100RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction ZXB Sign Extend Byte

Explanation Zero-extends the lowest byte of general-purpose register reg1 to word length.

ZXH

Zero Extend Half-word

Instruction format ZXH reg1

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

Format Format I

Opcode

15	0
00000000110RRRRR	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction ZXH Zero Extend Half-word

Explanation Zero-extends the lower halfword of general-purpose register reg1 to word length.

5.4 Number of Instruction Execution Clock Cycles

The number of instruction execution clock cycles differs depending on the combination of instructions. For details, refer to **CHAPTER 8 PIPELINE**.

Table 5-10 shows a list of the number of instruction execution clock cycles.

Table 5-10. List of Number of Instruction Execution Clock Cycles (1/3)

Instruction	Mnemonic	Operand	Bytes	Execution Clocks
				i – r – l
Load	SLD.B	disp7 [ep], r	2	1 – 1 – n[Note 1]
	SLD.H	disp8 [ep], r	2	1 – 1 – n[Note 1]
	SLD.W	disp8 [ep], r	2	1 – 1 – n[Note 1]
	SLD.BU	disp4 [ep], r	2	1 – 1 – n[Note1]
	SLD.HU	disp5 [ep], r	2	1 – 1 – n[Note 1]
	LD.B	disp16 [R], r	4	1 – 1 – n[Note 2]
	LD.H	disp16 [R], r	4	1 – 1 – n[Note 2]
	LD.W	disp16 [R], r	4	1 – 1 – n[Note 2]
	LD.BU	disp16 [R], r	4	1 – 1 – n[Note 2]
	LD.HU	disp16 [R], r	4	1 – 1 – n[Note 2]
Store	SST.B	r, disp7 [ep]	2	1 – 1 – 1
	SST.H	r, disp8 [ep]	2	1 – 1 – 1
	SST.W	r, disp8 [ep]	2	1 – 1 – 1
	ST.B	r, disp16 [R]	4	1 – 1 – 1
	ST.H	r, disp16 [R]	4	1 – 1 – 1
	ST.W	r, disp16 [R]	4	1 – 1 – 1
Arithmetic operation	MOV	R, r	2	1 – 1 – 1
	MOV	imm5, r	2	1 – 1 – 1
	MOV	imm32, r	6	2 – 2 – 2
	MOVEA	imm16, R, r	4	1 – 1 – 1
	MOVHI	imm16, R, r	4	1 – 1 – 1
	DIVH	R, r	2	35 – 35 – 35
	DIVH	R, r, w	4	35 – 35 – 35
	DIVHU	R, r, w	4	34 – 34 – 34
	DIV	R, r, w	4	35 – 35 – 35
	DIVU	R, r, w	4	34 – 34 – 34
	MULH	R, r	2	1 – 1 – 2
	MULH	imm5, r	2	1 – 1 – 2
	MULHI	imm16, R, r	4	1 – 1 – 2
	MUL	R, r, w	4	1 – 2[Note 3] – 2
	MUL	imm9, r, w	4	1 – 2[Note 3] – 2

Table 5-10. List of Number of Instruction Execution Clock Cycles (2/3)

Instruction	Mnemonic	Operand	Bytes	Execution Clocks
				i – r – l
Arithmetic operation (continued)	MULU	R, r, w	4	1 – 2[Note 3] – 2
	MULU	imm9, r, w	4	1 – 2[Note 3] – 2
	ADD	R, r	2	1 – 1 – 1
	ADD	imm5, r	2	1 – 1 – 1
	ADDI	imm16, R, r	4	1 – 1 – 1
	CMP	R, r	2	1 – 1 – 1
	CMP	imm5, r	2	1 – 1 – 1
	SUBR	R, r	2	1 – 1 – 1
	SUB	R, r	2	1 – 1 – 1
	CMOV	cccc, R, r, w	4	1 – 1 – 1
	CMOV	cccc, imm5, r, w	4	1 – 1 – 1
	SASF	cccc, r	4	1 – 1 – 1
	SETF	cccc, r	4	1 – 1 – 1
Saturated operation	SATSUBR	R, r	2	1 – 1 – 1
	SATSUB	R, r	2	1 – 1 – 1
	SATADD	R, r	2	1 – 1 – 1
	SATADD	imm5, r	2	1 – 1 – 1
	SATSUBI	imm16, R, r	4	1 – 1 – 1
Logical operation	NOT	R, r	2	1 – 1 – 1
	OR	R, r	2	1 – 1 – 1
	XOR	R, r	2	1 – 1 – 1
	AND	R, r	2	1 – 1 – 1
	TST	R, r	2	1 – 1 – 1
	SHR	imm5, r	2	1 – 1 – 1
	SAR	imm5, r	2	1 – 1 – 1
	SHL	imm5, r	2	1 – 1 – 1
	ORI	imm16, R, r	4	1 – 1 – 1
	XORI	imm16, R, r	4	1 – 1 – 1
	ANDI	imm16, R, r	4	1 – 1 – 1
	SHR	R, r	4	1 – 1 – 1
	SAR	R, r	4	1 – 1 – 1
	SHL	R, r	4	1 – 1 – 1
	ZXB	R	2	1 – 1 – 1
	ZXH	R	2	1 – 1 – 1
	SXB	R	2	1 – 1 – 1
	SXH	R	2	1 – 1 – 1

Table 5-10. List of Number of Instruction Execution Clock Cycles (3/3)

Instructions	Mnemonic	Operand		Bytes	Execution Clocks
					i – r – l
Logical operation	BSH	r, w		4	1 – 1 – 1
	BSW	r, w		4	1 – 1 – 1
	HSW	r, w		4	1 – 1 – 1
Branch (Continued)	JMP	[R]		2	3 – 3 – 3
	JR	disp22		4	2 – 2 – 2
	JARL	disp22, r		4	2 – 2 – 2
	Bcond	disp9	When condition is satisfied	2	$2^{Note\ 4}$ – $2^{Note\ 4}$ – $2^{Note\ 4}$
			When condition is not satisfied	2	1 – 1 – 1
Bit manipulation	SET1	bit#3, disp16 [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	SET1	r, [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	CLR1	bit#3, disp16 [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	CLR1	r, [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	NOT1	bit#3, disp16 [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	NOT1	r, [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	TST1	bit#3, disp16 [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
	TST1	r, [R]		4	$3^{Note\ 5}$ – $3^{Note\ 5}$ – $3^{Note\ 5}$
Special	LDSR	R, SR		4	1 – 1 – 1
	STSR	SR, r		4	1 – 1 – 1
	SWITCH	R		2	5 – 5 – 5
	PREPARE	list12, imm5		4	N+1$^{Note\ 6}$ – N+1^{Note6} – N+1$^{Note\ 6}$
	PREPARE	list12, imm5, sp		4	N+2$^{Note\ 6}$ – N+2$^{Note\ 6}$ – N+2$^{Note\ 6}$
	PREPARE	list12, imm5, imm16		6	N+2$^{Note\ 6}$ – N+2$^{Note\ 6}$ – N+2$^{Note\ 6}$
	PREPARE	list12, imm5, imm32		8	N+3$^{Note\ 6}$ – N+3$^{Note\ 6}$ – N+3$^{Note\ 6}$
	DISPOSE	imm5, list12		4	N+1$^{Note\ 6}$ – N+1$^{Note\ 6}$ – N+1$^{Note\ 6}$
	DISPOSE	imm5, list12, [R]		4	N+3$^{Note\ 6}$ – N+3$^{Note\ 6}$ – N+3$^{Note\ 6}$
	CALLT	imm6		2	4 – 4 – 4
	CTRET	–		4	3 – 3 – 3
	TRAP	vector		4	3 – 3 – 3
	RETI	–		4	3 – 3 – 3
	HALT	–		4	1 – 1 – 1
	EI	–		4	1 – 1 – 1
	DI	–		4	1 – 1 – 1
	NOP	–		2	1 – 1 – 1
	Undefined instruction code trap			4	3 – 3 – 3

Notes
1. Depends on the number of wait states (1 if no wait states).
2. Depends on the number of wait states (2 if no wait states).
3. 1 if r = w (lower 32 bits of results are not written to register) or w = r0 (higher 32 bits of results are not written to register).
4. 1 if last instruction involves PSW write access.
5. In case of no wait states (3 + number of read access wait states).
6. N is the total number of cycles to load registers in list12.
   (Depends on the number of wait states, N is the number of registers in list12 if no wait states)

Remarks 1. Operand conventions

Symbol	Meaning
R: reg1	General-purpose register (used as source register)
r: reg2	General-purpose register (mainly used as destination register)
w: reg3	General-purpose register (mainly used as remainder or higher 32 bits of multiply results)
SR: System Register	System register
imm×: immediate	×-bit immediate
disp×: displacement	×-bit displacement
bit#3: bit number	3-bit data for bit number specification
ep: Element Pointer	Element pointer
B: Byte	Byte (8 bits)
H: Half-word	Half-word (16 bits)
W: Word	Word (32 bits)
cccc: conditions	4-bit data condition code specification
vector	5-bit data for trap vector (00H to 1FH) specification
list×	List of registers (× is a maximum number of registers)

2. Execution clock conventions

Symbol	Meaning
i: issue	When other instruction is executed immediately after executing an instruction
r: repeat	When the same instruction is repeatedly executed immediately after the instruction has been executed
l: 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 and non-maskable interrupts. In contrast, an exception is an event whose occurrence is dependent on the program execution.

The V850 Series can process various interrupt requests from the on-chip peripheral hardware and external sources. In addition, exception processing can be started by an instruction (TRAP instruction) and by occurrence of an exception event (exception trap).

The interrupts and exceptions supported in the V850 Series are described below. When an interrupt or exception is deleted, control is transferred to a handler whose address is determined by the source of the interrupt or exception. The source of the event is specified by the exception code that is stored in the exception cause register (ECR). Each handler analyzes the exception cause register (ECR) and performs appropriate interrupt servicing or exception processing. The restore PC and PSW are written to the status saving registers (EIPC, EIPSW/FEPC, FEPSW).

To restore execution from interrupt or exception processing, use the RETI instruction.

Read the restore PC and PSW from the status saving register, and transfer control to the restore PC.

- Types of interrupt/exception processing

  The V850 Series handles the following four types of interrupts/exceptions.

  - Non-maskable interrupts
  - Maskable interrupts
  - Software exceptions
  - Exception traps

Table 6-1. Interrupt/Exception Codes

Interrupt/Exception Source		Classification	Exception Code	Handler	Restore PC
Name	Trigger			Address	
NMI	NMI input	Interrupt	0010H	00000010H	next PC Note 3
Maskable interrupt	**Note 2**	Interrupt	**Note 2**	**Note 1**	next PC Note 3
TRAP0n (n = 0 to FH)	TRAP instruction	Exception	004nH	00000040H	next PC
TRAP1n (n = 0 to FH)	TRAP instruction	Exception	005nH	00000050H	next PC
ILGOP	Illegal opcode	Exception	0060H	00000060H	next PC Note 4

Notes 1. The higher 16 bits of the handler address are 0000H and the lower 16 bits of the handler address are the same as the exception code.

2. Differs depending on the type of interrupt.

3. If an interrupt is acknowledged during execution of a DIV/DIVH/DIVU (divide) instruction, the restore PC becomes the PC value for the currently executed instruction (DIV/DIVH/DIVU).

4. The execution address of the illegal instruction is obtained by “restore PC-4” when an illegal opcode exception occurs.

The restore PC is the PC saved to EIPC or FEPC when interrupt/exception processing is started. “next PC” is the PC that starts processing after interrupt/exception processing.

The processing of maskable interrupts is controlled by the user through the interrupt controller (INTC). The INTC is different for each device in the V850 Series due to variations in on-chip peripherals, interrupt/exception sources and exception codes.

6.1 Interrupt Servicing

6.1.1 Maskable interrupt

A maskable interrupt can be masked by the interrupt control register.

The interrupt controller (INTC) issues an interrupt request to the CPU, based on the received interrupt with the highest priority.

If a maskable interrupt occurs due to INT input, the processor performs the following steps, and transfers control to the handler routine.

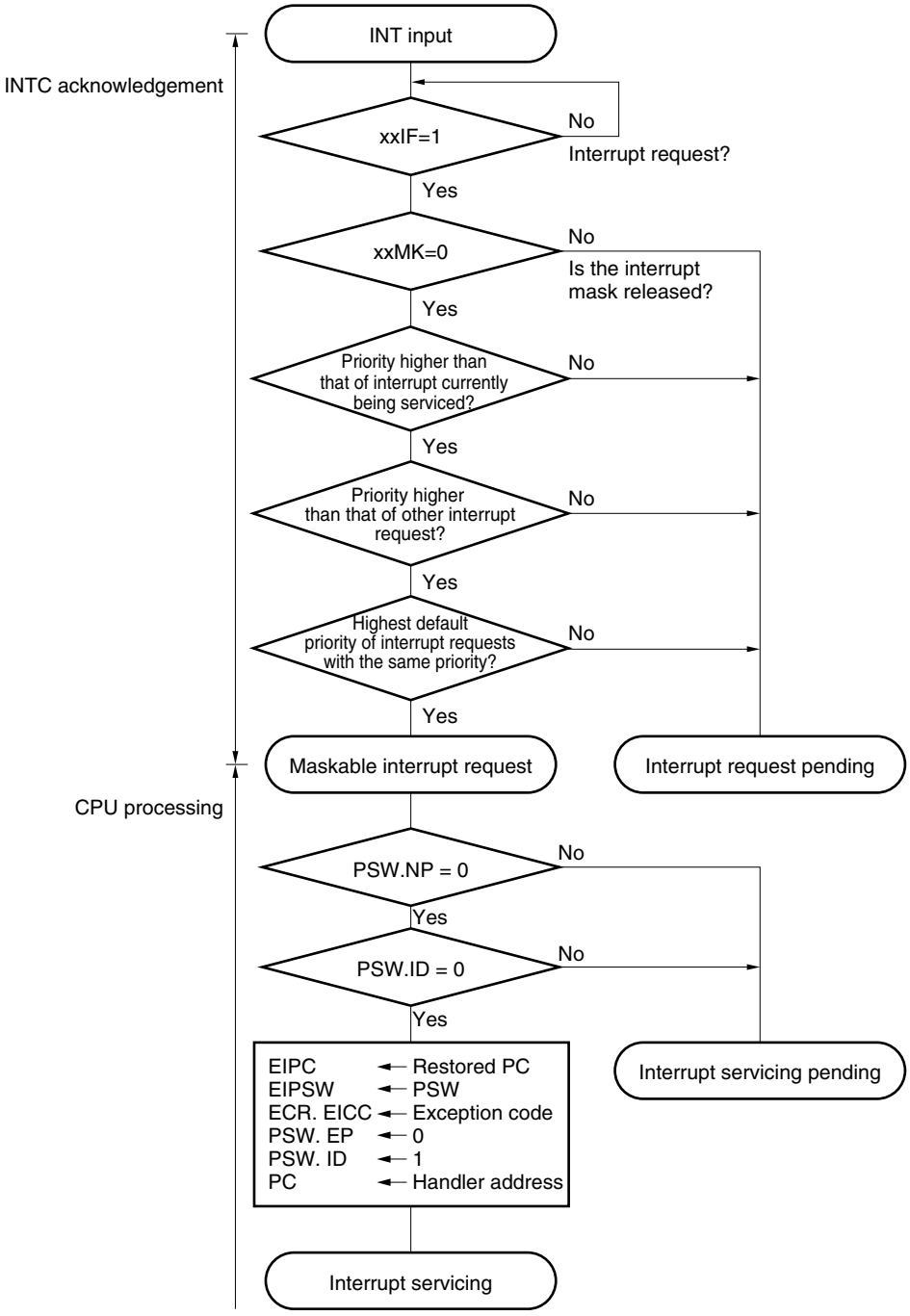
(1) Saves restore PC to EIPC.
(2) Saves current PSW to EIPSW.
(3) Writes exception code to lower halfword of ECR (EICC).
(4) Sets ID bit of PSW and clears EP bit.
(5) Sets handler address for each interrupt to PC and transfers control.

Interrupts are held pending in the interrupt controller (INTC) when one of the following two conditions occurs: when the interrupt input (INT) is masked by its interrupt controller, or when an interrupt service routine is currently being executed (when the NP bit of the PSW is 1 or when the ID bit of the PSW is 1). Interrupts are enabled by clearing the mask condition or by resetting the NP and ID bits of the PSW to 0 with the LDSR instruction, which will enable servicing of a new or already pending interrupt.

EIPC and EIPSW are used as the status saving registers. These registers must be saved by program to enable nesting of interrupts because there is only one set of EIPC and EIPSW provided. Bits 31 to 24 of EIPC and bits 31 to 8 of EIPSW are fixed to 0.

Figure 6-1 illustrates how a maskable interrupt is serviced.

Figure 6-1. Maskable Interrupt Servicing Format



6.1.2 Non-maskable interrupt

A non-maskable interrupt cannot be disabled by an instruction and can therefore always be acknowledged. Non-maskable interrupts of the V850 Series are generated by NMI input.

When the non-maskable interrupt is generated by NMI input, the processor performs the following steps, and transfers control to the handler routine.

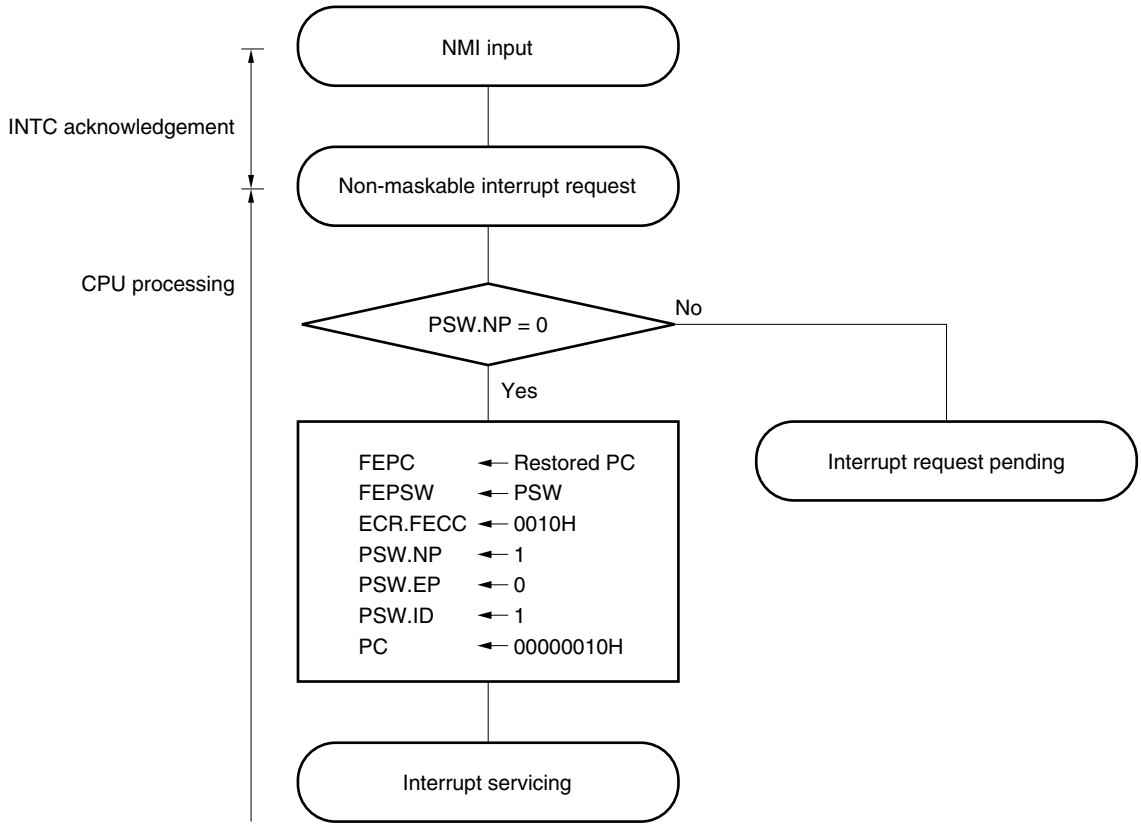
(1) Saves restore PC to FEPC.
(2) Saves current PSW to FEPSW.
(3) Writes exception code (0010H) to higher halfword of ECR (FECC).
(4) Sets NP and ID bits of PSW and clears EP bit.
(5) Sets handler address (00000010H) for the non-maskable interrupt to PC and transfers control.

Non-maskable interrupts are held pending in the interrupt controller INTC when another non-maskable interrupt is currently being executed (when the NP bit of the PSW is 1). Non-maskable interrupts are enabled by resetting the NP bit of the PSW to 0 with the RETI and LDSR instructions, which will enable servicing of a new or already pending interrupt.

FEPC and FEPSW are used as the status saving registers.

Figure 6-2 illustrates how a non-maskable interrupt is serviced.

Figure 6-2. Non-Maskable Interrupt Servicing Format



6.2 Exception Processing

6.2.1 Software exception

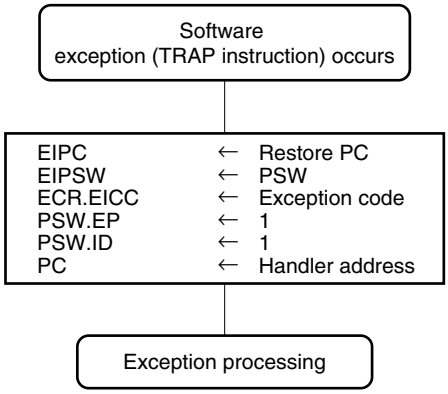
A software exception is generated when the CPU executes the TRAP instruction and is always acknowledged.

If a software exception occurs, the CPU performs the following steps, and transfers control to the handler routine.

(1) Saves restore PC to EIPC.
(2) Saves current PSW to EIPSW.
(3) Writes exception code to lower 16 bits (EICC) of ECR (interrupt cause).
(4) Sets EP and ID bits of PSW.
(5) Sets handler address (00000040H or 00000050H) for software exception to PC and transfers control.

Figure 6-3 illustrates how the software exception is processed.

Figure 6-3. Software Exception Processing Format



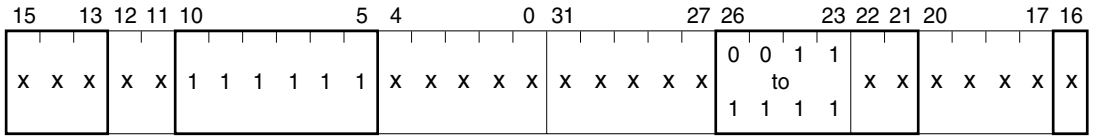
Handler address: 00000040H (vector = 0nH)
00000050H (vector = 1nH)

6.2.2 Exception trap

An exception trap is an interrupt requested when an instruction is illegally executed. The illegal opcode instruction code trap (ILGOP: ILlegal OPcode trap) is the exception trap in the V850 Series.

An illegal opcode instruction has an instruction code with an opcode (bits 10 through 5) of 111111B and sub-opcodes of 0111B through 1111B (bits 26 through 23) and 0B (bit16). When this kind of an illegal opcode instruction is executed, an illegal opcode instruction code trap occurs.

Figure 6-4. Illegal Instruction Code



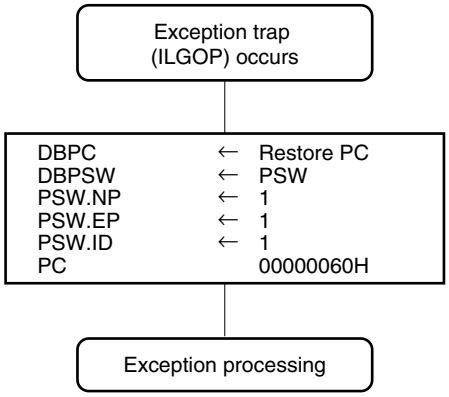
Remark ×: Don't care
□: Opcode/sub-opcode

If an exception trap occurs, the CPU performs the following steps, and transfers control to the handler routine.

(1) Saves restore PC to DBPC.
(2) Saves current PSW to DBPSW.
(3) Sets NP, EP, and ID bits of PSW.
(4) Sets handler address (00000060H) for exception trap to PC and transfers control.

Figure 6-5 illustrates how the exception trap is processed.

Figure 6-5. Exception Trap Processing Format



The execution address of the illegal instruction is obtained by "restore PC - 4" when an exception trap occurs.

Caution In addition to the defined opcodes and illegal opcodes, there is a range of codes not recognized by this processor. If an instruction corresponding to these codes is executed, normal operation is undetermined.

6.3 Restoring from Interrupt/Exception

All restoration from interrupt servicing/exception processing is executed by the RETI instruction.

With the RETI instruction, the processor performs the following steps, and transfers control to the address of the restore PC.

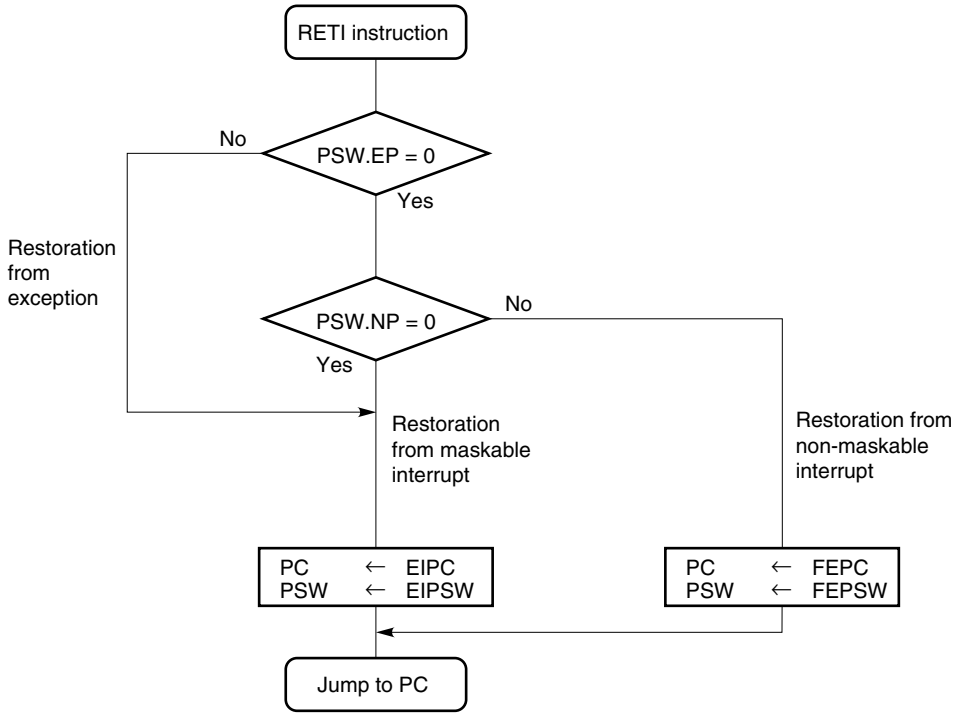
(1) If the EP bit of the PSW is 0 and the NP bit of the PSW is 1, the restore PC and PSW are read from FEPC and FEPSW. Otherwise, the restore PC and PSW are read from EIPC and EIPSW.
(2) Control is transferred to the address of the restored PC and PSW.

When execution has returned from exception processing or non-maskable interrupt servicing, the NP and EP bits of the PSW must be set to the following values by using the LDSR instruction immediately before the RETI instruction, in order to restore the PC and PSW normally:

To restore from non-maskable interrupt..... NP bit of PSW = 1, EP bit = 0
To restore from maskable interrupt processing..... NP bit of PSW = 0, EP bit = 0
To restore from exception processing ..... EP bit of PSW = 1

Figure 6-6 illustrates how restoration from an interrupt/exception is performed.

Figure 6-6. Restoration from Interrupt/Exception



CHAPTER 7 RESET

When a low-level signal is input to the $\overline{\text{RESET}}$ pin, the system is reset, and all on-chip hardware is initialized.

7.1 Initialization

When a low-level signal is input to the $\overline{\text{RESET}}$ pin, the system is reset, and each hardware register is set in the status shown in Table 7-1. When the $\overline{\text{RESET}}$ signal goes high, program execution begins. If necessary, initialize the contents of each register by program control.

Table 7-1. Register Status After Reset

Hardware (Symbol)		Status After Reset
Program counter	PC	00000000H
Interrupt status saving registers	EIPC	Undefined
	EIPSW	Undefined
NMI status saving registers	FEPC	Undefined
	FEPSW	Undefined
Exception cause registers (ECR)	FECC	0000H
	EICC	0000H
Program status word	PSW	00000020H
CALLT caller status saving registers	CTPC	Undefined
	CTPSW	Undefined
ILGOP caller status saving registers	DBPC	Undefined
	DBPSW	Undefined
CALLT base pointer	CTBP	Undefined
General-purpose registers	r0	Fixed to 00000000H
	r1 to r31	Undefined

7.2 Starting Up

All devices in the V850 Series begin program execution from address 00000000H after reset. No interrupt requests are acknowledged immediately after reset. To enable interrupts, clear the ID bit of the program status word (PSW) to 0.

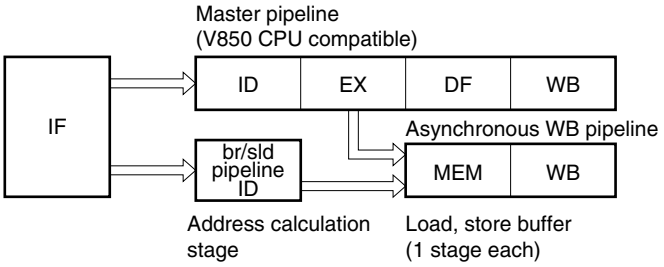
CHAPTER 8 PIPELINE

The V850 Series is based on RISC architecture and executes almost all instructions in one clock cycle under the control of a 5-stage pipeline.

The V850E/MS1 includes the V850E CPU core. The V850E CPU core, by optimizing the pipeline, improves the CPI (Cycles Per Instruction) rate over the previous V850 CPU core.

The pipeline configuration of the V850E CPU core is shown in Figure 8-1.

Figure 8-1. Pipeline Configuration



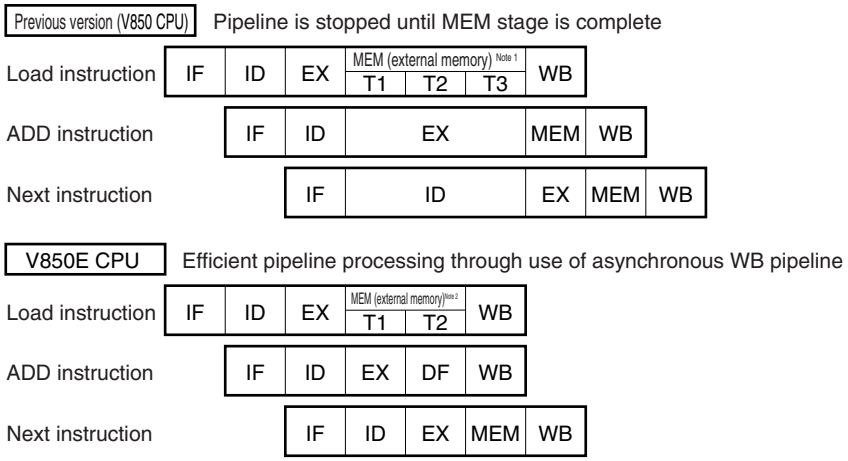
IF (instruction fetch):	Instruction is fetched and fetch pointer is incremented.
ID (instruction decode):	Instruction is decoded, immediate data is generated, and register is read.
EX (execution of ALU, multiplier, and barrel shifter):	The decoded instruction is executed.
MEM (memory access):	The memory is accessed at a specified address.
WB (write back):	Result of execution is written to register.
DF (data fetch):	Execution data is transferred to the WB stage.

8.1 Features

(1) Non-blocking load/store

As the pipeline does not stop during external memory access, efficient processing is possible. For example, Figure 8-2 shows a comparison of pipeline operations between the V850 CPU and the V850E CPU when the ADD instruction is executed after the execution of a load instruction for external memory.

Figure 8-2. Non-Blocking Load/Store



Notes 1. The basic bus cycle for the external memory of the V850 is 3 clocks.

2. The basic bus cycle for the external memory of the V850E is 2 clocks.

- **V850 CPU**

  The EX stage of the ADD instruction is usually executed in 1 clock. However, a wait time is generated in the EX stage of the ADD instruction during execution of the MEM stage of the previous load instruction. This is because the same stage of the 5 stages on the pipeline cannot be executed in the same internal clock interval. This also causes a wait time to be generated in the ID stage of the next instruction after the ADD instruction.

- **V850E CPU**

  An asynchronous WB pipeline for the instructions that are necessary for the MEM stage is provided in addition to the master pipeline. The MEM stage of the load instruction is therefore processed on this asynchronous WB pipeline. Because the ADD instruction is processed on the master pipeline, a wait time is not generated, making it possible to execute instructions efficiently.

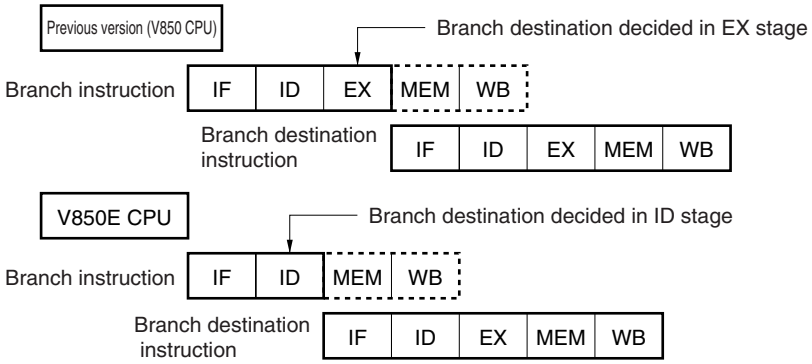
(2) 2-clock branch

When executing a branch instruction, the branch destination is decided in the ID stage.

In the case of the conventional V850 CPU, the branch destination of when the branch instruction is executed was decided after execution of the EX stage, but in the case of the V850E CPU, due to the addition of an address calculation stage for branch/short load instructions, the branch destination is decided in the ID stage. Therefore, it is possible to fetch the branch destination instruction 1 clock faster than in the conventional V850 CPU.

Figure 8-3 shows a comparison between the V850 CPU and the V850E CPU for pipeline operations with branch instructions.

Figure 8-3. Pipeline Operations with Branch Instructions

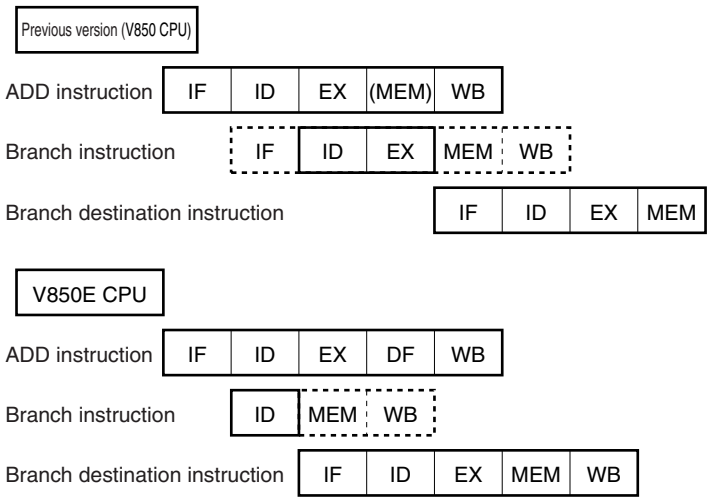


(3) Efficient pipeline processing

Because the V850E CPU has an ID stage for branch/short-load instructions in addition to the ID stage on the master pipeline, it is possible to perform efficient pipeline processing.

Figure 8-4 shows an example of a pipeline operation where the next branch instruction was fetched in the IF stage of the ADD instruction. The products of the V850 Series are 32-bit single-chip microcontrollers and the instruction fetch for the on-chip memory is performed in 32-bit (4-byte) units. Both ADD instructions and branch instructions use a 2-byte length instruction code.

Figure 8-4. Parallel Execution of Branch Instructions



- **V850 CPU**

  Although the instruction codes up to the next branch instruction are fetched in the IF stage of the ADD instruction, the ID stage of the ADD instruction and the ID stage of the branch instruction cannot operate together within the same internal clock. Therefore, it takes 5 clocks from the branch instruction fetch to the branch destination instruction fetch.

- **V850E CPU**

  Because V850E CPU has an ID stage for branch/short load instructions in addition to the ID stage on the master pipeline, the parallel execution of the ID stage of the ADD instruction and the ID stage of the branch instruction within the same internal clock is possible. Therefore, it takes only 3 clocks from the branch instruction fetch to the branch destination instruction.

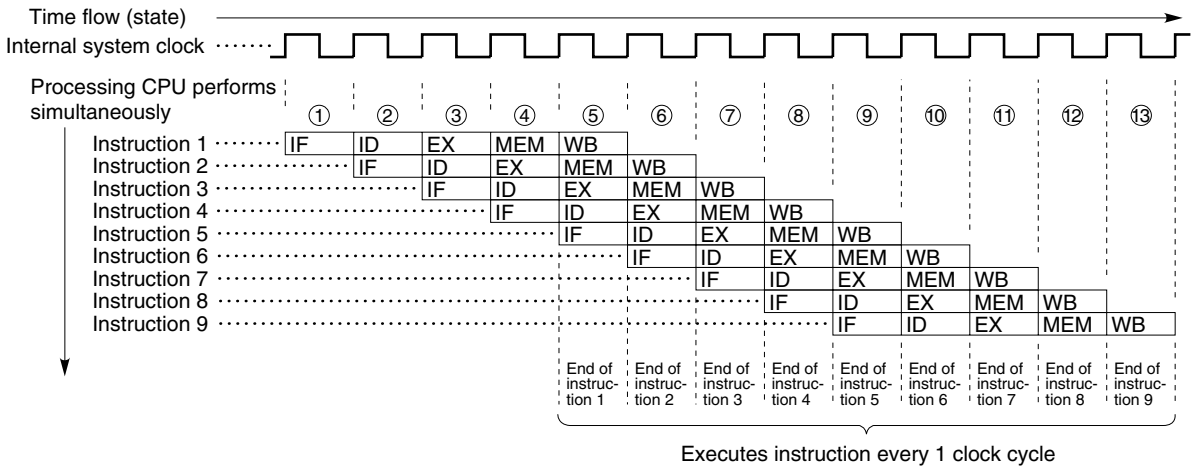
8.2 Outline of Operation

The instruction execution sequence of the V850 Series usually consists of five stages including fetch and write-back stages.

The execution time of each stage differs depending on the type of the instruction and the type of the memory to be accessed.

As an example of pipeline operation, Figure 8-5 shows the processing of the CPU when nine standard instructions are executed in succession.

Figure 8-5. Example of Executing Nine Standard Instructions



1 through 13 in the figure above indicate the states of the CPU. In each state, write-back of instruction n, memory access of instruction n+1, execution of instruction n+2, decoding of instruction n+3, and fetching of instruction n+4 are simultaneously performed. It takes five clock cycles to process a standard instruction, including fetching and writeback. Because five instructions can be processed at the same time, however, a standard instruction can be executed in 1 clock on average.

8.3 Pipeline Flow During Execution of Instructions

This section explains the pipeline flow during the execution of instructions.
During instruction fetch (IF stage) and memory access (MEM stage), the internal ROM/flash memory and the internal RAM are accessed, respectively. In this case, the IF and MEM stages are processed in 1 clock. In all other cases, the required time for access consists of the fixed access time, with the addition in some cases of the path wait time. Access times are shown in Figure 8-2 below.

Table 8-1. Access Times (in Clocks)

Resource (Bus Width) / Stage	Internal ROM/Flash Memory (32 Bits)	Internal RAM (32 Bits)	Internal Peripheral I/O (8/16 Bits)	External Memory[Note] (8/16 Bits)
Instruction fetch	1	1 or 2	Not possible	2 + n
Memory access (MEM)	3	1	3 + n	2 + n

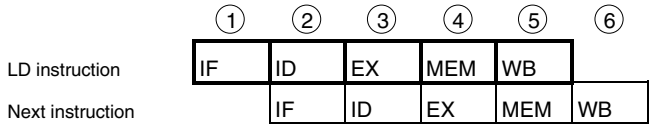
Note When the external memory type is set to SRAM, I/O.

Remark n: Wait number

8.3.1 Load instructions

(1) LD

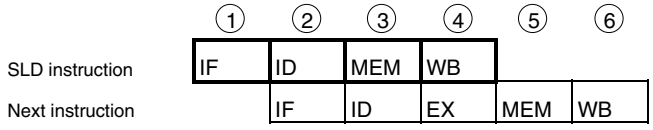
[Pipeline]



[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. If an instruction using the execution result is placed immediately after the LD instruction, data wait time occurs.

(2) SLD

[Pipeline]

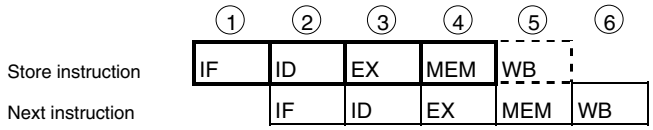


[Description] The pipeline consists of 4 stages, IF, ID, MEM and WB.

8.3.2 Store instructions

[Instructions] ST, SST

[Pipeline]



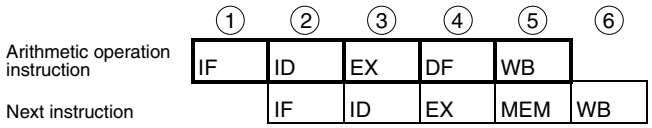
[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM and WB. However, no operation is performed in the WB stage, because no data is written to registers.

8.3.3 Arithmetic operation instructions (excluding multiply and divide instructions)

(1) Generic arithmetic operation instructions

[Instructions] MOV, MOVEA, MOVHI, ADD, ADDI, CMP, SUB, SUBR, SETF, SASF, CMOV, ZXB, ZXH, SXB, SXH, BSH, BSW, HSW

[Pipeline]

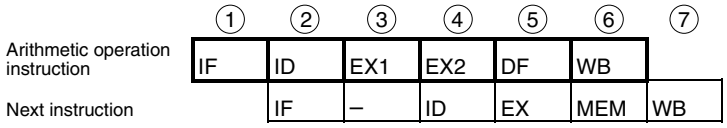


[Description] The pipeline consists of 5 stages, IF, ID, EX, DF and WB.

(2) Move word instruction

[Instructions] MOV imm32

[Pipeline]



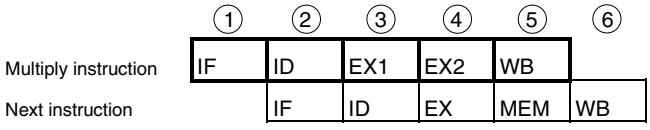
–: Idle inserted for wait

[Description] The pipeline consists of 6 stages, IF, ID, EX1, EX2, DF and WB.

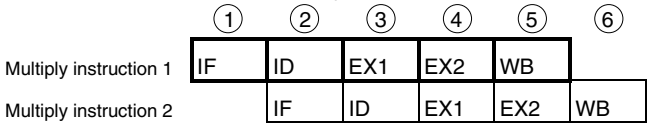
8.3.4 Multiply instructions

[Instructions] MULH, MULHI, MUL, MULU

[Pipeline] (a) When next instruction is not multiply instruction



(b) When next instruction is multiply instruction

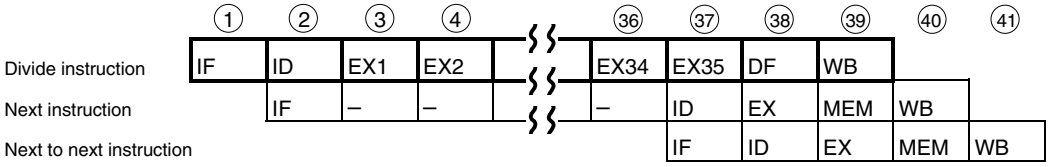


[Description] The pipeline consists of 5 stages, IF, ID, EX1, EX2, and WB. The EX stage requires 2 clocks, but the EX1 and EX2 stages can operate independently. Therefore, the number of clocks for instruction execution is always 1, even if several multiply instructions are executed in a row. However, if an instruction using the execution result is placed immediately after a multiply instruction, data wait time occurs.

8.3.5 Divide instructions

[Instructions] DIVH, DIV, DIVU, DIVHU

[Pipeline]



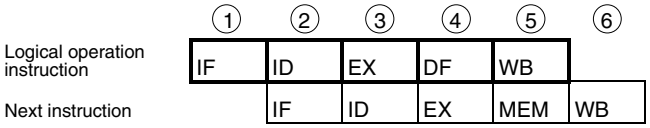
–: Idle inserted for wait

[Description] When a DIVU or DIVHU instruction is executed, the pipeline consists of 38 stages of IF, ID, EX1 to EX34, DF, and WB.
When a DIVH or DIV instruction is executed, the pipeline consists of 39 stages of IF, ID, EX1 to EX35, DF, and WB.

8.3.6 Logical operation instructions

[Instructions] NOT, OR, ORI, XOR, XORI, AND, ANDI, TST, SHR, SAR, SHL

[Pipeline]

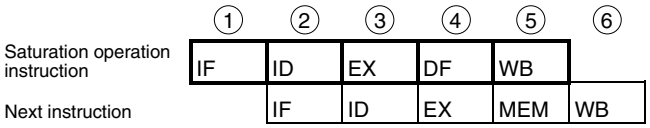


[Description] The pipeline consists of 5 stages, IF, ID, EX, DF, and WB.

8.3.7 Saturation operation instructions

[Instructions] SATADD, SATSUB, SATSUBI, SATSUBR

[Pipeline]



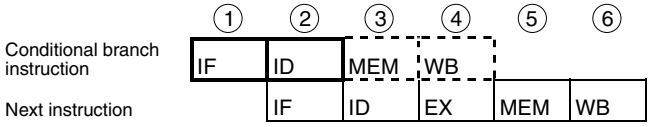
[Description] The pipeline consists of 5 stages, IF, ID, EX, DF, and WB.

8.3.8 Branch instructions

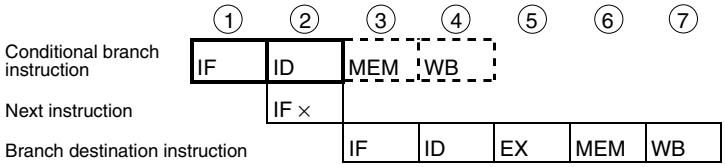
(1) Conditional branch instructions

[Instructions] Bcond instructions (BGT, BGE, BLT, BLE, BH, BNL, BL, BNH, BE, BNE, BV, BNV, BN, BP, BC, BNC, BZ, BNZ, BSA): Except BR instruction

[Pipeline] (a) When the condition is not realized



(b) When the condition is realized



IF ×: Instruction fetch that is not executed

[Description] The pipeline consists of 4 stages, IF, ID, MEM, and WB. However, no operation is performed in the MEM and WB stages, because memory is not accessed and no data is written to registers.

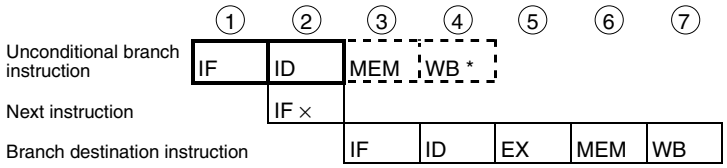
(a) When the condition is not realized
The number of execution clocks for the branch instruction is 1.

(b) When the condition is realized
The number of execution clocks for the branch instruction is 2. IF stage of the next instruction of the branch instruction is not executed. If an instruction overwriting the contents of PSW occurs immediately before a branch instruction execution, condition wait time occurs.

(2) Unconditional branch instructions

[Instructions] JR, JARL, BR

[Pipeline]



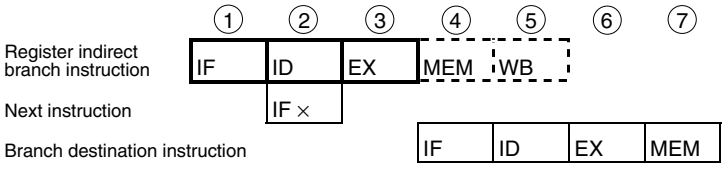
IF ×: Instruction fetch that is not executed
WB *: No operation is performed in the case of the JR instruction, and BR instruction but in the case of the JARL instruction, data is written to the restore PC.

[Description] The pipeline consists of 4 stages, IF, ID, MEM, and WB. However, no operation is performed in the MEM and WB stages, because memory is not accessed and no data is written to registers. However, in the case of the JARL instruction, data is written to the restore PC in the WB stage. Also, the IF stage of the next instruction of the branch instruction is not executed.

(3) Register indirect branch instructions

[Instructions] JMP, CTRET

[Pipeline]



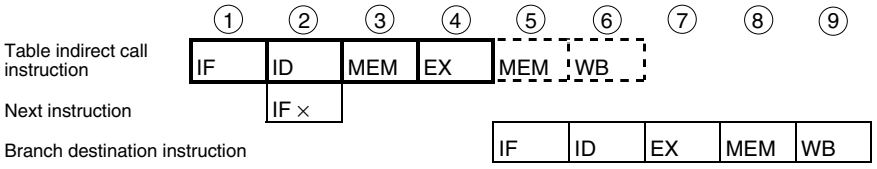
IF ×: Instruction fetch that is not executed

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the MEM and WB stages, because memory is not accessed and no data is written to registers.

(4) Table indirect call instructions

[Instructions] CALLT

[Pipeline]



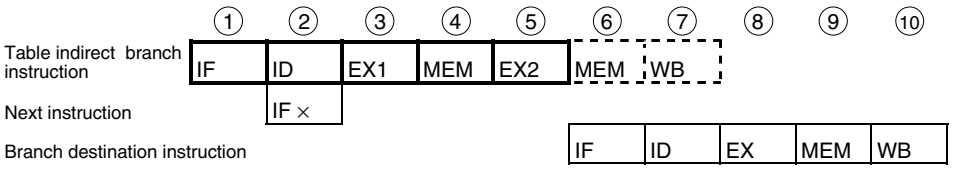
IF ×: Instruction fetch that is not executed

[Description] The pipeline consists of 6 stages, IF, ID, MEM, EX, MEM, and WB. However, no operation is performed in the second MEM and WB stages, because there is no second memory access and no data is written to registers.

(5) Table indirect branch instructions

[Instructions] SWITCH

[Pipeline]



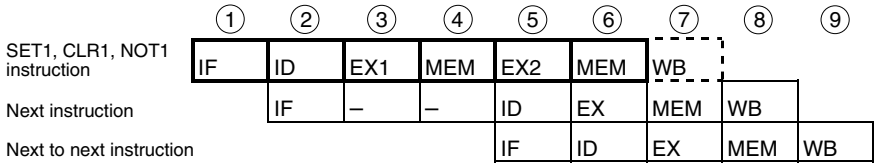
IF ×: Instruction fetch that is not executed

[Description] The pipeline consists of 7 stages, IF, ID, EX1, MEM, EX2, MEM, and WB. However, no operation is performed in the second MEM and WB stages, because there is no second memory access and no data is written to registers.

8.3.9 Bit manipulation instructions

(1) SET1, CLR1, NOT1

[Pipeline]

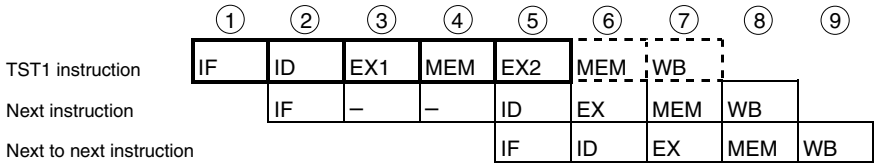


–: Idle inserted for wait

[Description] The pipeline consists of 7 stages, IF, ID, EX1, MEM, EX2, MEM, and WB. However, no operation is performed in the WB stage, because no data is written to registers.
In the case of these instructions, the memory access is read modify write, and the EX and MEM stages require 2 and 2 clocks, respectively.

(2) TST1

[Pipeline]



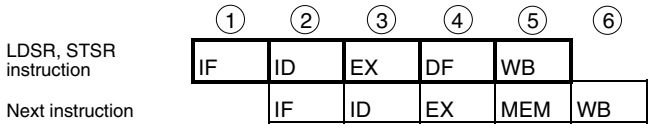
–: Idle inserted for wait

[Description] The pipeline consists of 7 stages, IF, ID, EX1, MEM, EX2, MEM, and WB. However, no operation is performed in the second MEM and WB stages, because there is no second memory access nor data write to registers.
In the case of this instruction, the memory access is read modify write, and the EX and MEM stages require 2 and 2 clocks, respectively.

8.3.10 Special instructions

(1) LDSR, STSR

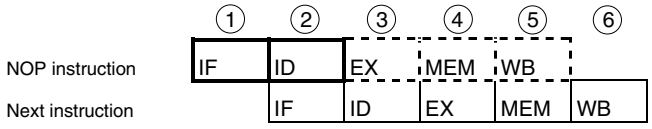
[Pipeline]



[Description] The pipeline consists of 5 stages, IF, ID, EX, DF, and WB. If the STSR instruction using the EIPC and FEPC system registers is placed immediately after the LDSR instruction setting these registers, data wait time occurs.

(2) NOP

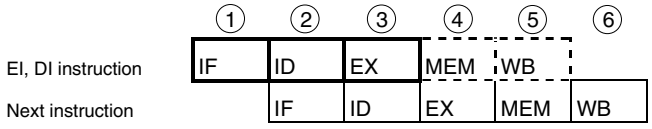
[Pipeline]



[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the EX, MEM and WB stages, because no operation and no memory access is executed, and no data is written to registers.

(3) EI, DI

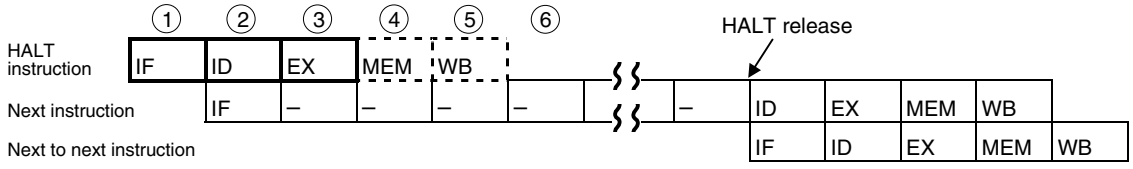
[Pipeline]



[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the MEM and WB stages, because memory is not accessed and data is not written to registers.

(4) HALT

[Pipeline]

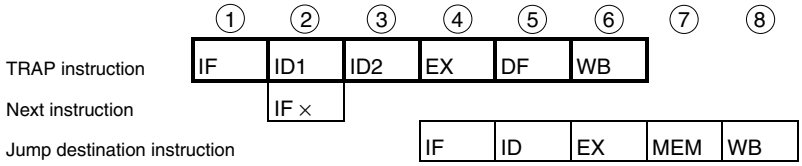


–: Idle inserted for wait

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM and WB. No operation is performed in the MEM and WB stages, because memory is not accessed and no data is written to registers. Also, for the next instruction, the ID stage is delayed until the HALT state is released.

(5) TRAP

[Pipeline]

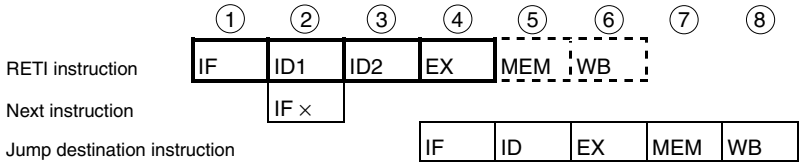


IF ×: Instruction fetch that is not executed
ID1: TRAP code detect
ID2: address generate

[Description] The pipeline consists of 6 stages, IF, ID1, ID2, EX, DF, and WB. The ID stage requires 2 clocks. Also, the IF stage of the next instruction is not executed.

(6) RETI

[Pipeline]



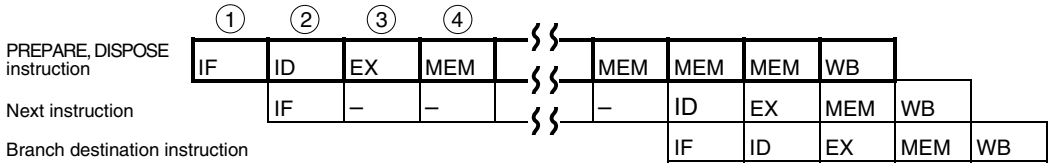
IF ×: Instruction fetch that is not executed
ID1: register select
ID2: read EIPC/FEPC

[Description] The pipeline consists of 6 stages, IF, ID1, ID2, EX, MEM, and WB. However, no operation is performed in the MEM and WB stages, because memory is not accessed and no data is written to registers. The ID stage requires 2 clocks. Also, the IF stage of the next instruction is not executed.

(7) PREPARE / DISPOSE

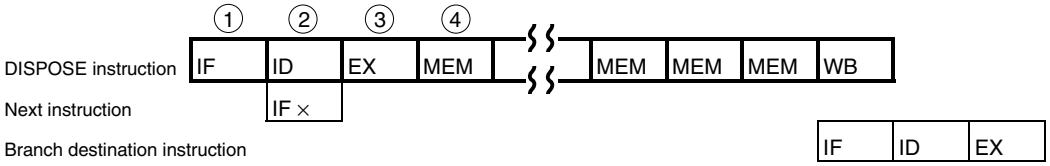
[Instructions] PREPARE, DISPOSE

[Pipeline] (a) PREPARE or DISPOSE without JMP



–: Idle inserted for wait

(b) DISPOSE with JMP



IF ×: Instruction fetch that is not executed

–: Idle inserted for wait

[Description] The pipeline consists of n (Number of register lists) + 4 stages, IF, ID, EX, n + 1 times MEM, and WB. The MEM stage requires n clocks.

8.4 Pipeline Disorder

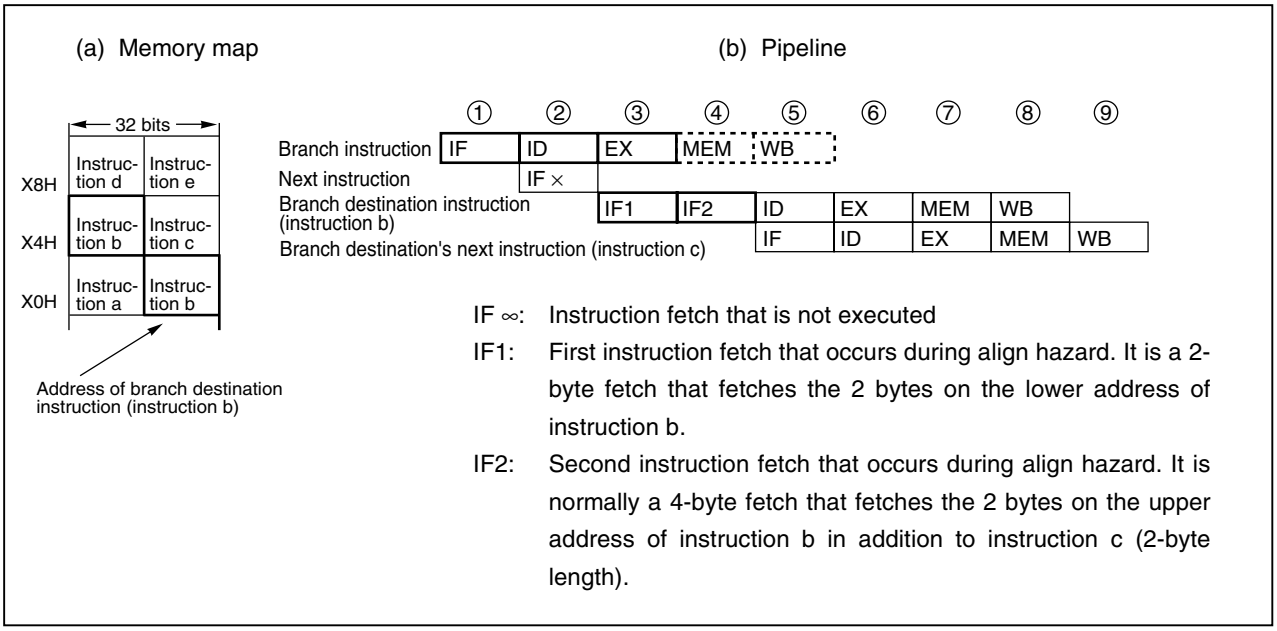
The pipeline consists of 5 stages from IF (Instruction Fetch) to WB (Write Back). Each stage basically requires 1 clock for processing, but the pipeline may become disordered, causing the number of execution clocks to increase. This section describes the main causes of pipeline disorder.

8.4.1 Alignment hazard

If the branch destination instruction address is not word aligned (A1=1, A0=0) and is 4 bytes in length, it is necessary to repeat IF twice in order to align instructions in word units. This is called an align hazard.

Look at this example: The instructions a to e are placed from address X0H, instruction b consists of 4 bytes, and the other instructions each consist of 2 bytes. In this case, instruction b is placed at X2H (A1=1, A0=0), and is not word aligned (A1=0, A0=0). Therefore, when this instruction b becomes the branch destination instruction, an align hazard occurs. When an align hazard occurs, the number of execution clocks of the branch instruction becomes 4.

Figure 8-6. Align Hazard Example



IF ∞: Instruction fetch that is not executed

IF1: First instruction fetch that occurs during align hazard. It is a 2-byte fetch that fetches the 2 bytes on the lower address of instruction b.

IF2: Second instruction fetch that occurs during align hazard. It is normally a 4-byte fetch that fetches the 2 bytes on the upper address of instruction b in addition to instruction c (2-byte length).

Align hazards can be prevented through the following handling in order to obtain faster instruction execution.

- Use 2-byte branch destination instruction.
- Use 4-byte instructions placed at word boundaries (A1=0, A0=0) for branch destination instructions.

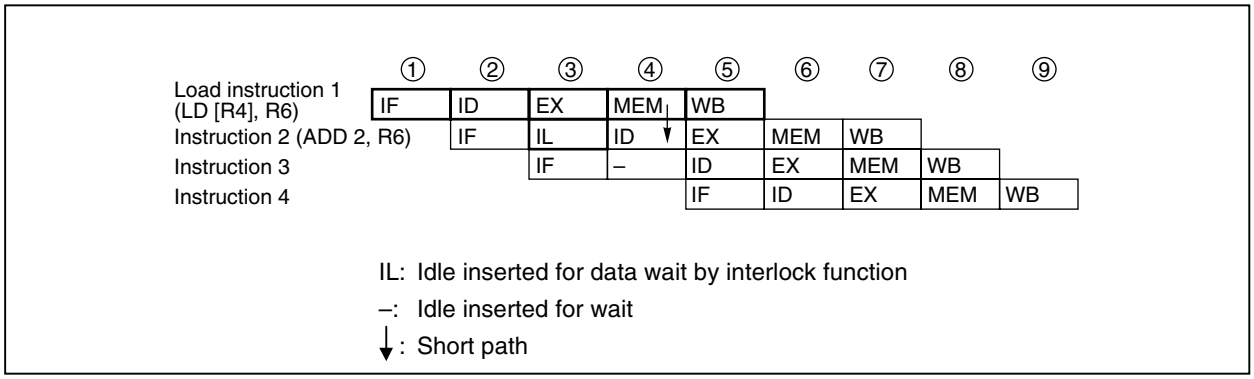
8.4.2 Referencing execution result of load instruction

For load instructions (LD, SLD), data read in the MEM stage is saved during the WB stage. Therefore, if the contents of the same register are used by the instruction immediately after the load instruction, it is necessary to delay the use of the register by this later instruction until the load instruction has finished using that register. This is called a hazard. The V850 Series has an interlock function that causes the CPU to automatically handle this hazard by delaying the ID stage of the next instruction.

The V850 Series also has a short path that allows the data read during the MEM stage to be used in the ID stage of the next instruction. This short path allows data to be read with the load instruction during the MEM stage and the use of this data in the ID stage of the next instruction with the same timing.

As a result of the above, when using the execution result in the instruction following immediately after, the number of execution clocks of the load instruction is 2.

Figure 8-7. Example of Execution Result of Load Instruction



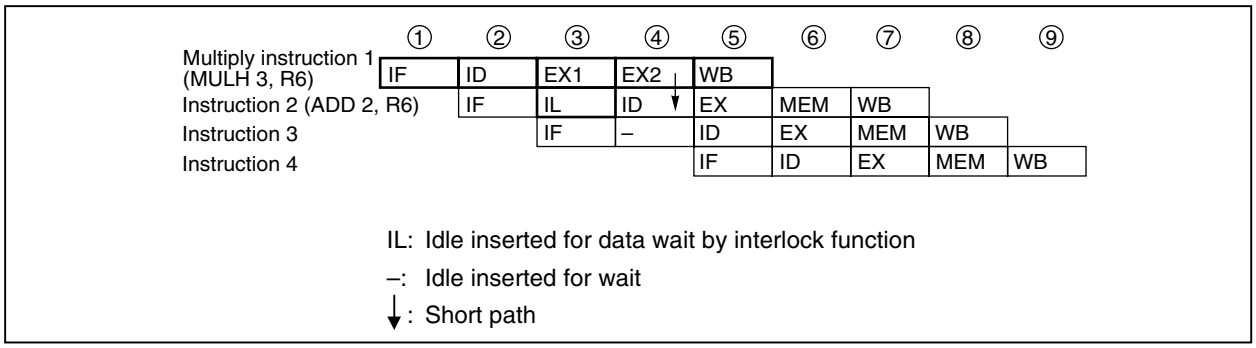
As described in Figure 8-7, when an instruction placed immediately after a load instruction uses its execution result, a data wait time occurs due to the interlock function, and the execution speed is lowered. This drop in execution speed can be avoided by placing instructions that use the execution result of a load instruction at least 2 instructions after the load instruction.

8.4.3 Referencing execution result of multiply instruction

For multiply instructions (MULH, MULHI), the operation result is saved to the register in the WB stage. Therefore, if the contents of the same register are used by the instruction immediately after the multiply instruction, it is necessary to delay the use of the register by this later instruction until the multiply instruction has ended using that register (occurrence of hazard).

The V850 Series interlock function delays the ID stage of the instruction following immediately after. A short path is also provided that allows the EX2 stage of the multiply instruction and the multiply instruction's operation result to be used in the ID stage of the instruction following immediately after with the same timing.

Figure 8-8. Example of Execution Result of Multiply Instruction



As described in Figure 8-8, when an instruction placed immediately after a multiply instruction uses its execution result, a data wait time occurs due to the interlock function, and the execution speed is lowered. This drop in execution speed can be avoided by placing instructions that use the execution result of a multiply instruction at least 2 instructions after the multiply instruction.

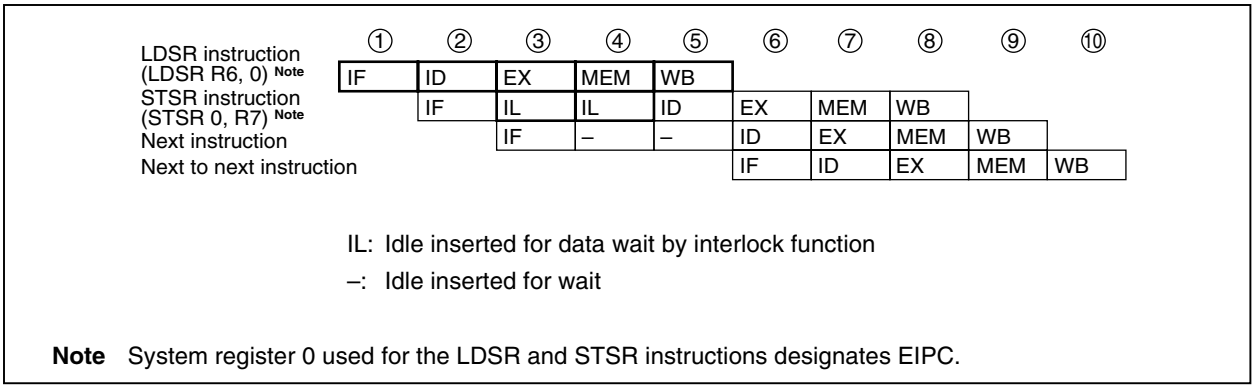
8.4.4 Referencing execution result of LDSR instruction for EIPC and FEPC

When using the LDSR instruction to set the data of the EIPC and FEPC system registers, and immediately after referencing the same system registers with the STSR instruction, the use of the system registers for the STSR instruction is delayed until the setting of the system registers with the LDSR instruction is completed (occurrence of hazard).

The V850 Series interlock function delays the ID stage of the STSR instruction immediately after.

As a result of the above, when using the execution result of the LDSR instruction for EIPC and FEPC for an STSR instruction following immediately after, the number of execution clocks of the LDSR instruction becomes 3.

Figure 8-9. Example of Execution Result of LDSR Instruction for EIPC and FEPC



Note System register 0 used for the LDSR and STSR instructions designates EIPC.

As described in Figure 8-9, when an STSR instruction is placed immediately after an LDSR instruction that uses the operand EIPC or FEPC, and that STSR instruction uses the LDSR instruction execution result, the interlock function causes a data wait time to occur, and the execution speed is lowered. This drop in execution speed can be avoided by placing STSR instructions that reference the execution result of the preceding LDSR instruction at least 3 instructions after the LDSR instruction.

8.4.5 Cautions when creating programs

When creating programs, pipeline disorder can be avoided and instruction execution speed can be raised by observing the following cautions.

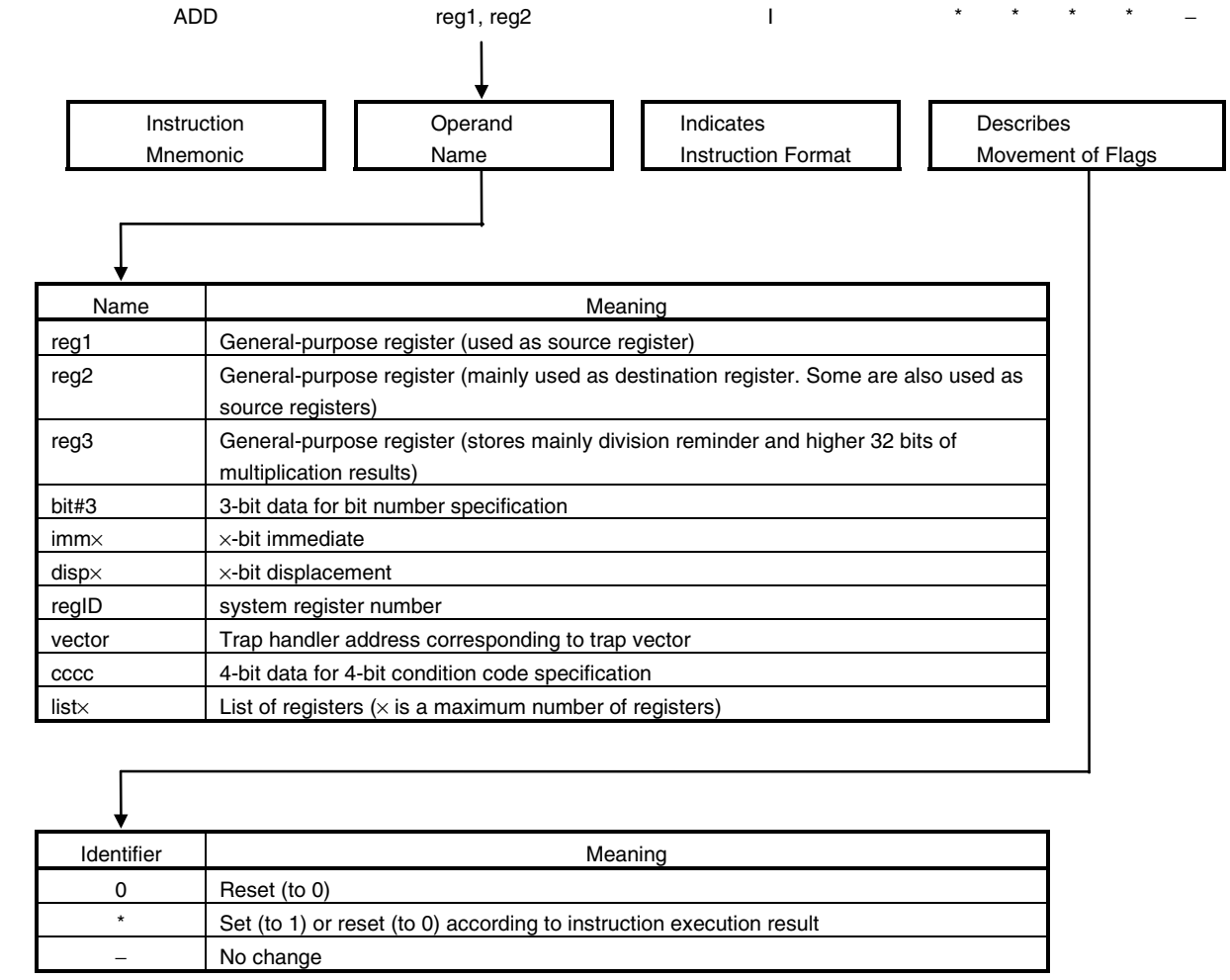
- Place instructions that use the execution result of load instructions (LD, SLD) at least 2 instructions after the load instruction.
- Place instructions that use the execution result of multiply instructions (MULH, MULHI) at least 2 instructions after the multiply instruction.
- If using the STSR instruction to read the setting results written to the EIPC or FEPC registers with the LDSR instruction, place the STSR instruction at least 3 instructions after the LDSR instruction.
- For the first branch destination instruction, use a 2-byte instruction, or a 4-byte instruction placed at the word boundary.

APPENDIX A INSTRUCTION MNEMONICS (IN ALPHABETICAL ORDER)

This appendix shows a list of the instruction mnemonics described previously.
These instruction mnemonics are listed in alphabetical order for easy reference.

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT

Convention



Name	Meaning
reg1	General-purpose register (used as source register)
reg2	General-purpose register (mainly used as destination register. Some are also used as source registers)
reg3	General-purpose register (stores mainly division reminder and higher 32 bits of multiplication results)
bit#3	3-bit data for bit number specification
imm×	×-bit immediate
disp×	×-bit displacement
regID	system register number
vector	Trap handler address corresponding to trap vector
cccc	4-bit data for 4-bit condition code specification
list×	List of registers (× is a maximum number of registers)

Identifier	Meaning
0	Reset (to 0)
*	Set (to 1) or reset (to 0) according to instruction execution result
–	No change

Table A-1. Instruction Mnemonics (in Alphabetical Order) (1/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
ADD	reg1, reg2	I	*	*	*	*	–	Add. Adds the word data of reg1 to the word data of reg2, and stores the result in reg2.
ADD	imm5, reg2	II	*	*	*	*	–	Add. Adds the 5-bit immediate data, sign-extended to word length, to the word data of reg2, and stores the result in reg2.
ADDI	imm16, reg1, reg2	VI	*	*	*	*	–	Add Immediate. Adds the 16-bit immediate data, sign-extended to word length, to the word data of reg1, and stores the result in reg2.
AND	reg1, reg2	I	–	0	*	*	–	And. ANDs the word data of reg2 with the word data of reg1, and stores the result in reg2.
ANDI	imm16, reg1, reg2	VI	–	0	*	*	–	And. ANDs the word data of reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in reg2.
Bcond	disp9	III	–	–	–	–	–	Branch on Condition Code. Tests a condition flag specified by an instruction. Branches if the specified condition is satisfied; otherwise, executes the next instruction. The branch destination PC holds the sum of the current PC value and 9-bit displacement which is the 8-bit immediate shifted 1 bit and sign-extended to word length.
BSH	reg2, reg3	XII	*	0	*	*	–	Byte Swap Halfword. Performs endian conversion.
BSW	reg2, reg3	XII	*	0	*	*	–	Byte Swap Word. Performs endian conversion.
CALLT	imm6	II	–	–	–	–	–	Call with Table Look Up. Based on CTBP contents, updates PC value and transfers control.
CLR1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Clear Bit. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Then clears the bit specified by the instruction bit field, of the byte data referenced by the generated address.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (2/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
CLR1	reg2 [reg1]	IX	–	–	–	*	–	Clear Bit. First, reads the data of reg1 to generate a 32-bit address. Then clears the bit specified by the data of lower 3 bits of reg2 of the byte data referenced by the generated address.
CMOV	cccc, reg1, reg2, reg3	XI	–	–	–	–	–	Conditional Move. Reg3 is set to reg1 if the condition specified by condition code “cccc” is satisfied; otherwise, set to the data of reg2.
CMOV	cccc, imm5, reg2, reg3	XII	–	–	–	–	–	Conditional Move. Reg3 is set to the data of 5-immediate, sign-extended to word length, if the condition specified by condition code “cccc” is satisfied; otherwise, set to the data of reg2.
CMP	reg1, reg2	I	*	*	*	*	–	Compare. Compares the word data of reg2 with the word data of reg1, and indicates the result by using the condition flags. To compare, the contents of reg1 are subtracted from the word data of reg2.
CMP	imm5, reg2	II	*	*	*	*	–	Compare. Compares the word data of reg2 with the 5-bit immediate data, sign-extended to word-length, and indicates the result by using the condition flags. To compare, the contents of the sign-extended immediate data are subtracted from the word data of reg2.
CTRET		X	*	*	*	*	*	Restore from CALLT. Fetches the restore PC and PWS from the appropriate system register and restores from the routine called by CALLT.
DI	–	X	–	–	–	–	–	Disables Interrupt. Sets the ID flag of the PSW to 1 to disable the acknowledgement of maskable interrupts; interrupts are immediately disabled at the start of this instruction execution.
DISPOSE	imm5, list12	XIII	–	–	–	–	–	Function Dispose. Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pops (loads data from the address specified by sp and adds 4 to sp) the general-purpose registers listed in list12.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (3/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
DISPOSE	imm5, list12, [reg1]	XIII	–	–	–	–	–	Function Dispose. Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pops (load data from the address specified by sp and adds 4 to sp) the general-purpose registers listed in list12, transfers control to the address specified by reg1.
DIV	reg1, reg2, reg3	XI	–	*	*	*	–	Divide Word. Divides the word data of reg2 by the word data of reg1, and stores the quotient in reg2 and the remainder in reg3. In the case of division by 0, overflow occurs and the quotient is undefined.
DIVH	reg1, reg2	I	–	*	*	*	–	Divide Halfword. Divides the word data of reg2 by the lower halfword data of reg1, and stores the quotient in reg2.
DIVH	reg1, reg2, reg3	XI	–	*	*	*	–	Divide Halfword. Divides word data of reg2 by lower halfword data of reg1, and stores the quotient in reg2 and the remainder in reg3.
DIVHU	reg1, reg2, reg3	XI	–	*	*	*	–	Divide Halfword Unsigned. Divides word data of reg2 by lower halfword data of reg1, and stores the quotient in reg2 and the remainder in reg3.
DIVU	reg1, reg2, reg3	XI	–	*	*	*	–	Divide Word Unsigned. Divides the word data of reg2 by the word data of reg1, and stores the quotient in reg2 and the remainder in reg3.
EI	–	X	–	–	–	–	–	Enable Interrupt. Resets the ID flag of the PSW to 0 and enables the acknowledgement of maskable interrupts at the beginning of the next instruction.
HALT	–	X	–	–	–	–	–	Halt. Stops the operating clock of the CPU and places the CPU in the HALT mode.
HSW	reg2, reg3	XII	*	0	*	*	–	Halfword Swap Word. Performs endian conversion.
JARL	disp22, reg2	V	–	–	–	–	–	Jump and Register Link. Saves the current PC value plus 4 to general register reg2, adds a 22-bit displacement, sign-extended to word length, to the current PC value, and transfers control to the PC. Bit 0 of the 22-bit displacement is masked by 0.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (4/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
JMP	[reg1]	I	–	–	–	–	–	Jump Register. Transfers control to the address specified by reg1. Bit 0 of the address is masked by 0.
JR	disp22	V	–	–	–	–	–	Jump Relative. Adds a 22-bit displacement, sign-extended to word length, to the current PC value, and transfers control to the PC. Bit 0 of the 22-bit displacement is masked by 0.
LD.B	disp16 [reg1], reg2	VII	–	–	–	–	–	Byte Load. Adds the data of reg1 to a 16-bit displacement, 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 then stored in reg2.
LD.H	disp16 [reg1], reg2	VII	–	–	–	–	–	Halfword Load. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Halfword data is read from this 32-bit address with bit 0 masked by 0, sign-extended to word length, and stored to reg2.
LD.W	disp16 [reg1], reg2	VII	–	–	–	–	–	Word Load. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Word data is read from this 32-bit address with bits 0 and 1 masked by 0, and stored in reg2.
LD.BU	disp16 [reg1], reg2	VII	–	–	–	–	–	Unsigned Byte Load. Adds the data of reg1 and the 16-bit displacement sign-extended to word length, and generates a 32-bit address. Then reads the byte data from the generated address, zero-extends it to word length, and stores it in reg2.
LD.HU	disp16 [reg1], reg2	VII	–	–	–	–	–	Unsigned Halfword Load. Adds the data of reg1 and the 16-bit displacement sign-extended to word length to generate a 32-bit address. Reads the halfword data from the address masking bit 0 of this 32-bit address by 0, zero-extends it to word length, and stores it in reg2.
LDSR	reg2, regID	IX	–	–	–	–	–	Load to System Register. Set the word data of reg2 to a system register specified by regID. If regID is PSW, the values of the corresponding bits of reg2 are set to the respective flags of the PSW.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (5/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
MOV	reg1, reg2	I	–	–	–	–	–	Move. Transfers the word data of reg1 to reg2.
MOV	imm5, reg2	II	–	–	–	–	–	Move. Transfers the value of a 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 a 16-bit immediate data, sign-extended to word length, to the word data of reg1, and stores the result in reg2.
MOVHI	imm16, reg1, reg2	VI	–	–	–	–	–	Move High Halfword. Adds word data, in which the higher 16 bits are defined by the 16-bit immediate data while the lower 16 bits are set to 0, to the word data of reg1 and stores the result in reg2.
MUL	reg1, reg2, reg3	XI	–	–	–	–	–	Multiply Word. Multiplies the word data of reg2 by the word data of reg1, and stores the result in reg2 and reg3 as double-word data.
MUL	imm9, reg2, reg3	XII	–	–	–	–	–	MultiplyWord. Multiplies the word data of reg2 by the 9-bit immediate data sign-extended to word length, and stores the result in reg2 and reg3.
MULH	reg1, reg2	I	–	–	–	–	–	Multiply Halfword. Multiplies the lower halfword data of reg2 by the lower halfword data of reg1, and stores the result in reg2 as word data.
MULH	imm5, reg2	II	–	–	–	–	–	Multiply Halfword. Multiplies the lower halfword data of reg2 by a 5-bit immediate data, sign-extended to halfword length, and stores the result in reg2 as word data.
MULHI	imm16, reg1, reg2	VI	–	–	–	–	–	Multiply Halfword Immediate. Multiplies the lower halfword data of reg1 by a 16-bit immediate data, and stores the result in reg2.
MULU	reg1, reg2, reg3	XI	–	–	–	–	–	Multiply Word Unsigned. Multiplies the word data of reg2 by the word data of reg1, and stores the result in reg2 and reg3 as double-word data. reg1 is not affected.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (6/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
MULU	imm9, reg2, reg3	XII	–	–	–	–	–	Multiply Word Unsigned. Multiplies the word data of reg2 by the 9-bit immediate data sign-extended to word length, and store the result in reg2 and reg3.
NOP	–	I	–	–	–	–	–	No Operation.
NOT	reg1, reg2	I	–	0	*	*	–	Not. Logically negates (takes 1's complement of) the word data of reg1, and stores the result in reg2.
NOT1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Not Bit. First, adds the data of reg1 to a 16-bit displacement, sigh-extended to word length, to generate a 32-bit address. The bit specified by the 3-bit field "bbb" is inverted at the byte data location referenced by the generated address.
NOT1	reg2 [reg1]	IX	–	–	–	*	–	Not Bit. First, reads reg1 to generate a 32-bit address. The bit specified by the lower 3 bits of reg2 of the byte data of the generated address is inverted.
OR	reg1, reg2	I	–	0	*	*	–	Or. ORs the word data of reg2 with the word data of reg1, and stores the result in reg2.
ORI	imm16, reg1, reg2	VI	–	0	*	*	–	Or Immediate. ORs the word data of reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in reg2.
PREPARE	list12, imm5	XIII	–	–	–	–	–	Function Prepare. The general-purpose register displayed in list12 is saved (4 is subtracted from sp, and the data is stored in that address). Next, the data is logically shifted 2 bits to the left, and the 5-bit immediate data zero-extended to word length is subtracted from sp.
PREPARE	list12, imm5, sp/imm	XIII	–	–	–	–	–	Function Prepare. The general-purpose register displayed in list12 is saved (4 is subtracted from sp, and the data is stored in that address). Next, the data is logically shifted 2 bits to the left, and the 5-bit immediate data zero-extended to word length is subtracted from sp. Then, the data specified by the third operand is loaded to ep.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (7/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
RETI	–	X	*	*	*	*	*	Return from Trap or Interrupt. Reads the restore PC and PSW from the appropriate system register, and restores from an exception or interrupt routine.
SAR	reg1, reg2	IX	*	0	*	*	–	Shift Arithmetic Right. Arithmetically shifts the word data of reg2 to the right by 'n' positions, where 'n' is specified by the lower 5 bits of reg1 (the MSB prior to shift execution is copied and set as the new MSB), and then writes the result to reg2.
SAR	imm5, reg2	II	*	0	*	*	–	Shift Arithmetic Right. Arithmetically shifts the word data of reg2 to the right by 'n' positions specified by the lower 5-bit immediate data, zero-extended to word length (the MSB prior to shift execution is copied and set as the new MSB), and then writes the result to reg2.
SASF	cccc, reg2	IX	–	–	–	–	–	Shift and Set Flag Condition. Reg2 is logically shifted left by 1, and its LSB is set to 1 if the condition specified by condition code "cccc" is satisfied; otherwise, LSB is set to 0.
SATADD	reg1, reg2	I	*	*	*	*	*	Saturated Add. Adds the word data of reg1 to the word data of reg2, and stores the result in reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored in reg2; if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. The SAT flag is set to 1.
SATADD	imm5, reg2	II	*	*	*	*	*	Saturated Add. Adds the 5-bit immediate data, sign-extended to word length, to the word data of reg2, and stores the result in reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored in reg2; if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. The SAT flag is set to 1.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (8/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
SATSUB	reg1, reg2	I	*	*	*	*	*	Saturated Subtract. Subtracts the word data of reg1 from the word data of reg2, and stores the result in reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored in reg2; if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. The SAT flag is set to 1.
SATSUBI	imm16, reg1, reg2	VI	*	*	*	*	*	Saturated Subtract Immediate. Subtracts a 16-bit immediate data, sign-extended to word length, from the word data of reg1, and stores the result in reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored in reg2; if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. The SAT flag is set to 1.
SATSUBR	reg1, reg2	I	*	*	*	*	*	Saturated Subtract Reverse. Subtracts the word data of reg2 from the word data of reg1, and stores the result in reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored in reg2; if the result exceeds the maximum negative value, the maximum negative value is stored in reg2. The SAT flag is set to 1.
SETF	cccc, reg2	IX	–	–	–	–	–	Set Flag Condition. The reg2 is set to 1 if the condition specified by condition code "cccc" is satisfied; otherwise, a 0 is stored in the register.
SET1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Set Bit. First, adds a 16-bit displacement, sign-extended to word length, to the data of reg1 to generate a 32-bit address. The bits, specified by the 3-bit field “bbb”, are set at the byte data location specified by the generated address.
SET1	reg2, [reg1]	IX	–	–	–	*	–	Set Bit. First, reads the data of general-purpose register reg1 to generate a 32-bit address. The bit specified by the data of the lower 3 bits of reg2 is set at the byte data location referenced by the generated address.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (9/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
SHL	reg1, reg2	IX	*	0	*	*	–	Shift Logical Left. Logically shifts the word data of reg2 to the left by 'n' positions (0 is shifted to the LSB side), where 'n' is specified by the lower 5 bits of reg1, and then writes the result to reg2.
SHL	imm5, reg2	II	*	0	*	*	–	Shift Logical Left. Logically shifts the word data of reg2 to the left by 'n' positions (0 is shifted to the LSB side), where 'n' is specified by a 5-bit immediate data, zero-extended to word length, and then writes the result to reg2.
SHR	reg1, reg2	IX	*	0	*	*	–	Shift Logical Right. Logically shifts the word data of reg2 to the right by 'n' positions (0 is shifted to the MSB side), where 'n' is specified by the lower 5 bits of reg1, and then writes the result to reg2.
SHR	imm5, reg2	II	*	0	*	*	–	Shift Logical Right. Logically shifts the word data of reg2 to the right by 'n' positions (0 is shifted to the MSB side), where 'n' is specified by a 5-bit immediate data, zero-extended to word length, and then writes the result to reg2.
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 then stored in reg2.
SLD.H	disp8 [ep], reg2	IV	–	–	–	–	–	Halfword Load. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Halfword data is read from this 32-bit address with bit 0 masked by 0, sign-extended to word length, and stored in reg2.
SLD.W	disp8 [ep], reg2	IV	–	–	–	–	–	Word Load. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Word data is read from this 32-bit address with bits 0 and 1 masked by 0, and stored in reg2.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (10/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
SLD.BU	disp4 [ep], reg2	IV	–	–	–	–	–	Unsigned Byte Load. 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.HU	disp5 [ep], reg2	IV	–	–	–	–	–	Unsigned Halfword Load. Adds the 5-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Halfword data is read from this 32-bit address with bit 0 masked by 0, zero-extended to word-length, and stored in reg2.
SST.B	reg2, disp7 [ep]	IV	–	–	–	–	–	Byte Store. Adds the 7-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the data of the lowest byte of reg2 in the generated address.
SST.H	reg2, disp8 [ep]	IV	–	–	–	–	–	Halfword Store. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the lower halfword of reg2 in the generated 32-bit address with bit 0 masked by 0.
SST.W	reg2, disp8 [ep]	IV	–	–	–	–	–	Word Store. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address, and stores the word data of reg2 in the generated 32-bit address with bits 0 and 1 masked by 0.
ST.B	reg2, disp16 [reg1]	VII	–	–	–	–	–	Byte Store. Adds the 16-bit displacement, sign-extended to word length, to the data of reg1 to generate a 32-bit address, and stores the lowest byte data of reg2 in the generated address.
ST.H	reg2, disp16 [reg1]	VII	–	–	–	–	–	Halfword Store. Adds the 16-bit displacement, sign-extended to word length, to the data of reg1 to generate a 32-bit address, and stores the lower halfword of reg2 in the generated 32-bit address with bit 0 masked by 0.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (11/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
ST.W	reg2, disp16 [reg1]	VII	–	–	–	–	–	Word Store. Adds the 16-bit displacement, sign-extended to word length, to the data of reg1 to generate a 32-bit address, and stores the word data of reg2 in the generated 32-bit address with bits 0 and 1 masked by 0.
STSR	regID, reg2	IX	–	–	–	–	–	Store Contents of System Register. Stores the contents of the system register specified by regID in reg2.
SUB	reg1, reg2	I	*	*	*	*	–	Subtract. Subtracts the word data of reg1 from the word data of reg2, and stores the result in reg2.
SUBR	reg1, reg2	I	*	*	*	*	–	Subtract Reverse. Subtracts the word data of reg2 from the word data of reg1, and stores the result in reg2.
SWITCH	reg1	I	–	–	–	–	–	Jump with Table Look Up. Adds the table entry address (address following the SWITCH instruction) and data of reg1 logically shifted to the left by 1 bit, and loads the halfword entry data specified by the table entry address. Next, logically shifts to the left by 1 bit the loaded data, and after sign-extending it to word length, branches to the target address added to the table entry address (instruction following the SWITCH instruction).
SXB	reg1	I	–	–	–	–	–	Sign Extend Byte. Sign-extends the lowermost byte of reg1 to word length.
SXH	reg1	I	–	–	–	–	–	Sign Extend Halfword. Sign-extends lower halfword of reg1 to word length.
TRAP	vector	X	–	–	–	–	–	Trap. Saves the restore PC and PSW to EIPC and EIPSW, respectively; sets the exception code (EICC and ECR) and the flags of the PSW (EP and ID flags); jumps to the address of the trap handler corresponding to the trap vector specified by vector number (0 to 31), and starts exception processing.
TST	reg1, reg2	I	–	0	*	*	–	Test. ANDs the word data of reg2 with the word data of reg1. The result is not stored, and only the flags are changed.

Table A-1. Instruction Mnemonics (in Alphabetical Order) (12/12)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
TST1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Test Bit. Adds the data of reg1 to a 16-bit displacement, sigh-extended to word length, to generate a 32-bit address. Performs the test on the bit specified by the 3-bit field “bbb” at the byte data location referenced by the generated address. If the specified bit is 0, the Z flag is set to 1; if the bit is 1, the Z flag is reset to 0. The byte data, including the specified bit, is not affected.
TST1	reg2, [reg1]	IX	–	–	–	*	–	Test Bit. First, reads the data of reg1 to generate a 32-bit address. If the bits indicated by the lower 3 bits of reg2 of the byte data of the generated address are 0, the Z flag is set, and if they are 1, reset is performed.
XOR	reg1, reg2	I	–	0	*	*	–	Exclusive Or. Exclusively ORs the word data of reg2 with the word data of reg1, and stores the result in reg2.
XORI	imm16, reg1, reg2	VI	–	0	*	*	–	Exclusive Or Immediate. Exclusively ORs the word data of reg1 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 to word length the lowest byte of reg1.
ZXH	reg1	I	–	–	–	–	–	Zero Extend Halfword. Zero-extends to word length the lower halfword of reg1.

APPENDIX B INSTRUCTION LIST

Table B-1. Mnemonic List (1/2)

Mnemonic	Function	Mnemonic	Function
	Load/store	SAR	Shift Arithmetic Right
		BSH	Byte Swap Half-word
LD.B	Load Byte	BSW	Byte Swap Word
LD.H	Load Half-word	HSW	Half-word Swap Word
LD.W	Load Word		
LD.BU	Load Byte Unsigned		(2-operand immediate)
LD.HU	Load Half-word Unsigned		
SLD.B	Load Byte	MOV	Move
SLD.H	Load Half-word	ADD	Add
SLD.W	Load Word	CMP	Compare
SLD.BU	Load Byte Unsigned	SATADD	Saturated Add
SLD.HU	Load Half-word Unsigned	SETF	Set Flag Condition
ST.B	Store Byte	SHL	Shift Logical Left
ST.H	Store Half-word	SHR	Shift Logical Right
ST.W	Store Word	SAR	Shift Arithmetic Right
SST.B	Store Byte	SASF	Shift and Set Flag Condition
SST.H	Store Half-word		
SST.W	Store Word		(3-operand register)
	Integer arithmetic operation/logical operation/	MUL	Multiply Word
	saturated operation	MULU	Multiply Word Unsigned
	(1-operand register)	DIVH	Divide Half-word
		DIV	Divide Word
ZXB	Zero Extended Byte	DIVHU	Divide Half-word Unsigned
ZXH	Zero Extended Half-word	DIVU	Divide Word Unsigned
SXB	Sign Extended Byte		
SXH	Sign Extended Half-word		(3-operand immediate)
	(2-operand register)	MOVHI	Move High Half-word
		MOVEA	Move Effective Address
MOV	Move	ADDI	Add Immediate
ADD	Add	MULHI	Multiply Half-word Immediate
SUB	Subtract	SATSUBI	Saturated Subtract Immediate
SUBR	Subtract Reverse	ORI	Or Immediate
MULH	Multiply Half-word	ANDI	And Immediate
DIVH	Divide Half-word	XORI	Exclusive Or Immediate
CMP	Compare	MUL	Multiply Word
SATADD	Saturated Add	MULU	Multiply Word Unsigned
SATSUB	Saturated Subtract		
SATSUBR	Saturated Subtract Reverse		Branch
TST	Test		
OR	Or	JMP	Jump Register
AND	And	JR	Jump Relative
XOR	Exclusive Or	JARL	Jump and Register Link
NOT	Not	Bcond	Branch on Condition Code
SHL	Shift Logical Left		
SHR	Shift Logical Right		

Table B-1. Mnemonic List (2/2)

Mnemonic	Function
	Bit manipulation
SET1	Set Bit
CLR1	Clear Bit
NOT1	Not Bit
TST1	Test Bit
	Special
LDSR	Load System Register
STSR	Store System Register
TRAP	Trap
RETI	Return from Trap or Interrupt
HALT	Halt
DI	Disable Interrupt
EI	Enable Interrupt
NOP	No Operation
SWITCH	Jump with Table Look Up
PREPARE	Function Prepare
DISPOSE	Function Dispose
CALLT	Call with Table Look Up
CTRET	Return from CALLT

Table B-2. Instruction Set (1/2)

Instruction Code b10 • • • • b5	Instruction Format		Format	Remarks
0 0 0 0 0 0	MOV	reg1, reg2	I	When reg1, reg2 = 0, NOP
0 0 0 0 0 1	NOT	reg1, reg2		
0 0 0 0 1 0	DIHV	reg1, reg2		
0 0 0 0 1 0	SWITCH	reg1		
0 0 0 0 1 1	JMP	[reg1]		
0 0 0 1 0 1	SATSUBR	reg1, reg2		
0 0 0 1 0 0	ZXB	reg1		
0 0 0 1 0 1	SATSUB	reg1, reg2		
0 0 0 1 0 1	SXB	reg1		
0 0 0 1 1 0	SATADD	reg1, reg2		
0 0 0 1 1 0	ZXH	reg1		
0 0 0 1 1 1	MULH	reg1, reg2		
0 0 1 1 0 0	SXH	reg1		
0 0 1 0 0 0	OR	reg1, reg2		
0 0 1 0 0 1	XOR	reg1, reg2		
0 0 1 0 1 0	AND	reg1, reg2		
0 0 1 0 1 1	TST	reg1, reg2		
0 0 1 1 0 0	SUBR	reg1, reg2		
0 0 1 1 0 1	SUB	reg1, reg2		
0 0 1 1 1 0	ADD	reg1, reg2		
0 0 1 1 1 1	CMP	reg1, reg2		
0 1 0 0 0 0	MOV	imm5, reg2	II	
0 1 0 0 0 1	SATADD	imm5, reg2		
0 1 0 0 0 X	CALLT	imm6		
0 1 0 0 1 0	ADD	imm5, reg2		
0 1 0 0 1 1	CMP	imm5, reg2		
0 1 0 1 0 0	SHR	imm5, reg2		
0 1 0 1 0 1	SAR	imm5, reg2		
0 1 0 1 1 0	SHL	imm5, reg2		
0 1 0 1 1 1	MULH	imm5, reg2		
0 0 0 0 1 1	SLD.BU	disp4 [ep], reg2	IV	
0 0 0 0 1 1	SLD.HU	disp5 [ep], reg2		
0 1 1 0 X X	SLD.B	disp7 [ep], reg2		
0 1 1 1 X X	SST.B	reg2, disp7 [ep]		
1 0 0 0 X X	SLD.H	disp8 [ep], reg2		
1 0 0 1 X X	SST.H	reg2, disp8 [ep]		
1 0 1 0 X X	SLD.W	disp8 [ep], reg2		
1 0 1 0 X X	SST.W	reg2, disp8 [ep]		
1 0 1 1 X X	Bcond	disp9	III	
1 1 0 0 0 0	ADDI	imm16, reg, reg2	VI	
1 1 0 0 0 1	MOVEA	imm16, reg1, reg2		
1 1 0 0 0 1	MOV	imm32, reg1		
1 1 0 0 1 0	MOVHI	imm16, reg1, reg2		
1 1 0 0 1 1	SATSUBI	imm16, reg1, reg2		
1 1 0 1 0 0	ORI	imm16, reg1, reg2		
1 1 0 1 0 1	XORI	imm16, reg1, reg2		
1 1 0 1 1 0	ANDI	im16, reg1, reg2		
1 1 0 1 1 1	MULHI	imm16, reg1, reg2		

Table B-2. Instruction Set (2/2)

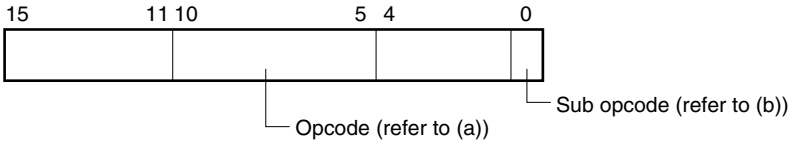
Instruction Code b10 • • • • b5	Instruction Format		Format	Remarks
1 1 1 0 0 0	LD.B	disp16 [reg1], reg2	VII	
1 1 1 0 0 1	LD.H	disp16 [reg1], reg2		
1 1 1 0 1 0	LD.W	disp16 [reg1], reg2		
1 1 1 0 1 0	ST.B	reg2, disp16 [reg1]		
1 1 1 0 1 1	ST.H	reg2, disp16 [reg1]		
1 1 1 0 1 1	ST.W	reg2, disp16 [reg1]		
1 1 1 1 0 X	LD.BU	disp16 [reg1], reg2		
1 1 1 1 1 1	LD.HU	disp16 [reg1], reg2		
1 1 1 1 0 X	JARL	disp22, reg2	V	When reg2 = 0, JR disp22
1 1 1 1 1 0	SET1	bit#3, disp16 [reg1]	VIII	
1 1 1 1 1 0	CLR1	bit#3, disp16 [reg1]		
1 1 1 1 1 0	NOT1	bit#3, disp16 [reg1]		
1 1 1 1 1 0	TST1	bit#3, disp16 [reg1]		
1 1 1 1 1 1	SETF	cccc, reg2	IX	
1 1 1 1 1 1	LDSR	reg2, regID		
1 1 1 1 1 1	STSR	regID, reg2		
1 1 1 1 1 1	SHR	reg1, reg2		
1 1 1 1 1 1	SAR	reg1, reg2		
1 1 1 1 1 1	SHL	reg1, reg2		
1 1 1 1 1 1	SASF	cccc, reg2		
1 1 1 1 1 1	CLR1	reg2, [reg1]		
1 1 1 1 1 1	NOT1	reg2, [reg1]		
1 1 1 1 1 1	SET1	reg2, [reg1]		
1 1 1 1 1 1	TST1	reg2, [reg1]		
1 1 1 1 1 1	TRAP	vector	X	
1 1 1 1 1 1	HALT			
1 1 1 1 1 1	RETI			
1 1 1 1 1 1	DI			
1 1 1 1 1 1	EI			
1 1 1 1 1 1	CTRET			
1 1 1 1 1 1	Undefined instruction			
1 1 1 1 1 1	DIVH	reg1, reg2, reg3	XI	
1 1 1 1 1 1	DIV	reg1, reg2, reg3		
1 1 1 1 1 1	DIVHU	reg1, reg2, reg3		
1 1 1 1 1 1	DIVU	reg1, reg2, reg3		
1 1 1 1 1 1	MUL	reg1, reg2, reg3		
1 1 1 1 1 1	MULU	reg1, reg2, reg3		
1 1 1 1 1 1	CMOV	cccc, reg1, reg2, reg3		
1 1 1 1 1 1	MUL	imm9, reg2, reg3	XII	
1 1 1 1 1 1	MULU	imm9, reg2, reg3		
1 1 1 1 1 1	CMOV	cccc, imm5 reg2, reg3		
1 1 1 1 1 1	BSH	reg2, reg3		
1 1 1 1 1 1	BSW	reg2, reg3		
1 1 1 1 1 1	HSW	reg2, reg3		
1 1 0 0 1 X	DISPOSE	imm5, list12	XIII	
1 1 0 0 1 X	DISPOSE	imm5, list12 [reg1]		
1 1 1 1 0 X	PREPARE	list12, imm5		
1 1 1 1 0 X	PREPARE	list12, imm5, sp/imm		

APPENDIX C INSTRUCTION OPCODE MAP

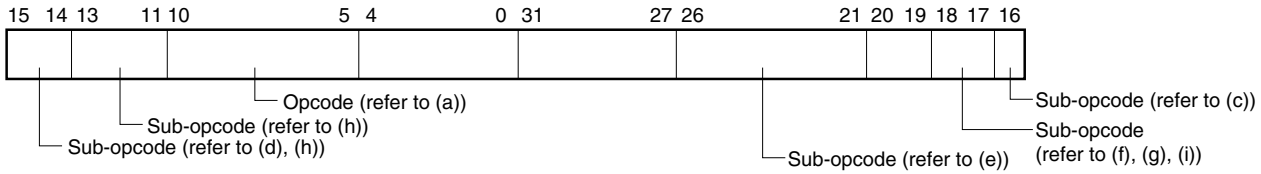
The opcode map for the instruction code is shown in (a) to (i).
For the operand conventions, refer to **Table 5-10 Remark 1** Operand conventions.

Instruction Codes

• 16-bit instruction format



• 32-bit instruction format



(a) Opcode

Bits 10 to 7 \ Bits 6 and 5	00	01	10	11	Format
0000	MOV R, r NOP[Note 1]	NOT	DIVH SWITCH[Note 2] Undefined[Note 3]	JMP[Note 4] SLD.BU[Note 5] SLD.HU[Note 6]	I, IV
0001	SATSUBR ZXB[Note 4]	SATSUB SXB[Note 4]	SATADD R, r ZXH[Note 4]	MULH SXH[Note 4]	I
0010	OR	XOR	AND	TST	I
0011	SUBR	SUB	ADD R, r	CMP R, r	I
0100	MOV imm5, r CALLT[Note 4]	SATADD imm5, r	ADD imm5, r	CMP imm5, r	II
0101	SHR imm5, r	SAR imm5, r	SHL imm5, r	MULH imm5, r Undefined[Note 4]	II
0110	SLD.B				IV
0111	SST.B				IV
1000	SLD.H				IV
1001	SST.H				IV
1010	SLD.W[Note 7] SST.W[Note 7]				IV
1011	Bcond				III
1100	ADDI	MOVEA MOV imm32 R[Note 4]	MOVHI DISPOSE[Note 4]	SATSUBI	VI, XIII
1101	ORI	XORI	ANDI	MULHI Undefined[Note 4]	VI
1110	LD.B	LD.H[Note 8] LD.W[Note 8]	ST.B	ST.H[Note 8] ST.W[Note 8]	VII
1111	JR/JARL LD.BU[Note 10] PREPARE[Note 11]		Bit manipulation 1[Note 9]	LD.HU[Note 10] Undefined[Note 11] Expansion 1[Note 12]	V, VII, VIII, XIII

Notes 1. If reg1 = r0 and reg2 = r0 (instruction without reg1 and reg2)

2. If reg1 ≠ r0 and reg2 = r0 (instruction with reg1 and without reg2)

3. If reg1 = r0 and reg2 ≠ r0 (instruction without reg1 and with reg2)

4. If reg2 = r0 (instruction without reg2)

5. If bit4 = 0 and reg2 ≠ r0 (instruction with reg2)

6. If bit4 = 1 and reg2 ≠ r0 (instruction with reg2)
7. Refer to (b)
8. Refer to (c)
9. Refer to (d)
10. If bit16 = 1 and reg2 ≠ r0 (instruction with reg2)
11. If bit16 = 1 and reg2 = r0 (instruction without reg2)
12. Refer to (e)

(b) Short format load/store instruction (displacement/sub-opcode)

Bits 10 to 7 \ Bit 0	0	1
0110	SLD.B	
0111	SST.B	
1000	SLD.H	
1001	SST.H	
1010	SLD.W	SST.W

(c) Load/store instruction (displacement/sub-opcode)

Bits 6 and 5 \ Bit 16	0	1
00	LD.B	
01	LD.H	LD.W
10	ST.B	
11	ST.H	ST.W

(d) Bit manipulation instruction 1 (sub-opcode)

Bit 15 \ Bit 14	0	1
0	SET1	NOT1
1	CLR 1	TST 1

(e) Extend 1 (sub-opcode)

Bits 26 to 23 \ Bits 22 and 21	00	01	10	11	Format
0000	SETF	LDSR	STSR	Undefined	IX
0001	SHR	SAR	SHL	Bit manipulation 2[Note 1]	IX
0010	TRAP	HALT	RETI[Note 2] CTRET[Note 2] Undefined	EI[Note 3] DI[Note 3] Undefined	X
0011	Undefined				—
0100	SASF	MUL R, r, w MULU R, r, w[Note 4]	MUL imm9, r, w MULU imm9, r, w[Note 4]		IX, XI, XII
0101	DIVH DIVHU[Note 4]		DIV DIVU[Note 4]		XI
0110	CMOV cccc, imm5, r, w	CMOV cccc, R, r, w	BSW[Note 5] BSH[Note 5] HSW[Note 5]	Undefined	XI, XII
0111 to 1111	Illegal Opcode				—

Notes 1. Refer to (f)

2. Refer to (g)

3. Refer to (h)

4. If bit17 = 1

5. Refer to (i)

(f) Bit manipulation instruction 2 (sub-opcode)

Bit 18 \ Bit 17	0	1
0	SET1	NOT1
1	CLR1	TST1

(g) Return instruction (sub-opcode)

Bit 18 \ Bit 17	0	1
0	RETI	Undefined
1	CTRET	

(h) PSW operation instruction (sub-opcode)

Bits 15 and 14 \ Bits 13 to 11	000	001	010	011	100	101	110	111
00	DI	Undefined						
01	Undefined							
10	EI	Undefined						
11	Undefined							

(i) Endian conversion instruction (sub-opcode)

Bit 18 \ Bit 17	0	1
0	BSW	BSH
1	HSW	Undefined

APPENDIX D INSTRUCTIONS ADDED TO V850E

The instruction codes of the V850E CPU are upwardly compatible with the instruction codes of the V850 CPU at the object code level. In the case of the V850E CPU, instructions that even if executed have no meaning in the case of the V850 CPU (mainly instructions that write to the r0 register) are extended as additional instructions.

The following table shows the V850 CPU instructions corresponding to the instruction codes added in the V850E CPU. Refer to this table when switching from products that incorporate the V850 CPU to products that incorporate the V850E CPU.

Table D-1. Instructions Added to V850E CPU and V850 CPU Instructions with Same Instruction Code (1/2)

Instructions Added in V850E CPU	V850 CPU Instructions with Same Instruction Code as V850E CPU
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
(RFU)	MULH imm5, r0
(RFU)	MULHI imm16, reg1, r0
BSH reg2, reg3	Illegal instruction
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	
HSW reg2, reg3	
MUL imm9, reg2, reg3	
MUL reg1, reg2, reg3	
MULU reg1, reg2, reg3	
MULU imm9, reg2, reg3	
SASF cccc, reg2	

Table D-1. Instructions Added to V850E CPU and V850 CPU Instructions with Same Instruction Code (2/2)

Instructions Added in V850E CPU	V850 CPU Instructions with Same Instruction Code as V850E CPU
CLR1 reg2, [reg1]	Undefined
LD.BU disp16 [reg1], reg2	
LD.HU disp16 [reg1], reg2	
NOT1 reg2, [reg1]	
PREPARE list12, imm5	
PREPARE list12, imm5, sp/imm	
SET1 reg2, [reg1]	
SLD.BU disp4 [ep], reg2	
SLD.HU disp5 [ep], reg2	
TST1 reg2, [reg1]	

APPENDIX E INDEX

APPENDIX F REVISION HISTORY

The history of revisions up to this edition is shown below. “Applied to:” indicates the chapters to which the revision was applied.

Edition	Contents	Applied to:
5th edition	Addition of V850E/MS2 (μPD703130)	Throughout
	Modification of description in **1.3 Product Development**	**CHAPTER 1 INTRODUCTION**
	Modification of description in **Figure 1-1 V850 Family Lineup**	
	Addition of description in **1.4 CPU Configuration**	
	Addition of description of HALT instruction in **5.3 Instruction Set**	**CHAPTER 5 INSTRUCTIONS**
	Addition of description of LD instruction in **5.3 Instruction Set**	
	Addition of description of SLD instruction in **5.3 Instruction Set**	
	Addition of **8.4 Pipeline Disorder**	**CHAPTER 8 PIPELINE**
	Modification of description in **APPENDIX C INSTRUCTION OPCODE MAP**	**APPENDIX C INSTRUCTION OPCODE MAP**
	Addition of description in **APPENDIX C (h) PSW operation instruction (sub-opcode)**	
6th edition	Modification of description of CLR1 instruction in **5.3 Instruction Set**	**CHAPTER 5 INSTRUCTIONS**
	Modification of description of NOT1 instruction in **5.3 Instruction Set**	
	Modification of description of SET1 instruction in **5.3 Instruction Set**	
	Modification of description of SLD1 instruction in **5.3 Instruction Set**	
	Addition of description of SST instruction in **5.3 Instruction Set**	
	Addition of **APPENDIX F REVISION HISTORY**	**APPENDIX F REVISION HISTORY**