NEC

User's Manual

V850 FAMILY™

32-bit Single-Chip Microcontroller

Architecture

Document No. U10243EJ7V0UM00 (7th edition)
Date Published March 2001 J CP(K)

[MEMO]

SUMMARY OF CONTENTS

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.

The export of these products from Japan is regulated by the Japanese government. The export of some or all of these products may be prohibited without governmental license. To export or re-export some or all of these products from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC 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 Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-3067-5800
Fax: 01-3067-5899

NEC Electronics (France) S.A.
Madrid Office
Madrid, Spain
Tel: 091-504-2787
Fax: 091-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

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 Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

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

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.2

Major Revisions in this Edition

Page	Description
Throughout	Change of target device
p. 18	Change of **Figure 1-1 V850 Family Product Development**
p. 20	Modification of description in **2.1.1 (1) General registers**
p. 21	Modification of **Figure 2-1 Program Register List**
p. 22	Modification of **Figure 2-2 List of Program Register Operations**
p. 103	Modification of **Table 5-10 List of Number of Instruction Execution Clock Cycles**
p. 106	Modification of description in **CHAPTER 6 INTERRUPTS AND EXCEPTIONS**
p. 111	Modification of description in **6.2.2 Exception trap**
p. 111	Modification of **Figure 6-5 Exception Trap Processing Format**

The mark ★ shows major revised points.

INTRODUCTION

Target Readers

This manual is intended for users who wish to understand the functions of the V850 CPU core in the V850 Family in designing systems using the products of the V850 Family.

- V850 CPU Core Products
  - V851™ Note 1 : μPD703000, 703001, 70P3000
  - V852™ Note 2 : μPD703002, 70P3002
  - V853™ : μPD703003A, 70F3003A, 703003A(A)Note 3, 70F3003A(A)Note 3, 703004A, 703025A, 703025A(A), 70F3025A
  - V854™ Note 2 : μPD703006Note 1, 703008, 70F3008, 703008Y, 70F3008Y
  - V850/SA1™ : μPD703014A, 703014AY, 703014B, 703014BY, 703015Note 2, 703015YNote 2, 703015A, 703015AY, 703015B, 703015BY, 70F3015BNote 3, 70F3015BYNote 3, 703017A, 70F3017A, 703017AY, 70F3017AY
  - V850/SB1™ : μPD703030ANote 3, 703030AYNote 3, 703031A, 703031AY, 703032A, 70F3032A, 703032AY, 70F3032AY, 703033A, 70F3033A, 703033AY, 70F3033AY
  - V850/SB2™ : μPD703034A, 703034AY, 703035A, 70F3035A, 703035AY, 70F3035AY, 703036ANote 3, 703036AYNote 3, 703037A, 70F3037A, 703037AY, 70F3037AY
  - V850/SF1™ : μPD703078Y, 703079Y, 70F3079Y
  - V850/SV1™ : μPD703038Note 3, 703038YNote 3, 70F3038Note 3, 70F3038YNote 3, 703039, 703039Y, 703040, 703040Y, 703041, 703041Y, 70F3040, 70F3040Y

Notes 1. Maintenance parts
2. Discontinued
3. Under development

Purpose

This manual is intended for users to understand the functions of the V850 Family Architecture described in the Organization below.

Organization

This manual contains the following information:

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

How to read this manual

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

To know about the hardware functions:
→ Read the **User's Manual Hardware** of each device.

To learn about the functions of a specific instruction in detail:
→ Read **CHAPTER 5 INSTRUCTIONS**.

To know about the electrical specifications:
→ Read the **DATA SHEET** of each device.

To understand the overall functions of the V850 Family:
→ Read this manual in the order of **CONTENTS**.

With the V850 Family, data consisting of 2 bytes is called a half-word, and data consisting of 4 bytes is called a word.

Conventions

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

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

	K (kilo) ... 2^{10} = 1,024
	M (mega) ... 2^{20} = $1,024^2$
	G (giga) ... 2^{30} = $1,024^3$
Data 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**

Product Name		Data Sheet	User's Manual Hardware	User's Manual Architecture
V851	μPD703000, 703001	U10987E	U10935E	This manual
	μPD70P3000	U10988E		
V852	μPD703002	U11826E	U10038E	
	μPD70P3002	U11827E		
V853	μPD703003A, 703004A, 703025A	U13188E	U10913E	
	μPD70F3003A, 70F3025A	U13189E		
V854	μPD70F3008	U12756E	U11969E	
	μPD70F3008Y	U12755E		
	μPD703006, 703008, 703008Y	—		
V850/SA1	μPD703014A, 703014AY, 703015A, 703015AY, 703017A, 703017AY	U14526E	U12768E	
	μPD70F3017A, 70F3017AY	U14527E		
V850/SB1	μPD703031A, 703031AY, 703033A, 703033AY, 70F3033A, 70F3033AY	U14734E	U13850E	
	μPD703032A, 703032AY, 70F3032A, 70F3032AY	U14893E		
V850/SB2	μPD703034A, 703034AY, 703035A, 703035AY, 70F3035A, 70F3035AY	U14780E		
	μPD703037A, 703037AY, 70F3037A, 70F3037AY	U14894E		
V850/SF1	μPD703078Y, 703079Y, 70F3079Y	U15183E	U14665E	
V850/SV1	μPD703039, 703039Y, 703040, 703040Y, 703041, 703041Y	U13953E	U14462E	
	μPD70F3040, 70F3040Y	U14662E		

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

Document Name		Document Number
IE-703002-MC (In-circuit Emulator for V852, V853, V854, V850/SA1, V850/SB1, V850/SB2, V850/SF1, V850/SV1)		U11595E
IE-703003-MC-EM1 (V853 Peripheral I/O Board)		U11596E
IE-703008-MC-EM1 (V854 Peripheral I/O Board)		U12420E
IE-703017-MC-EM1 (V850/SA1 Peripheral I/O Board)		U12898E
IE-703037-MC-EM1 (V850/SB1, V850/SB2 Peripheral I/O Board)		U14151E
IE-703040-MC-EM1 (V850/SV1 Peripheral I/O Board)		U14337E
CA850 (Ver. 2.30 or Later) (C Compiler Package)	Operation	U14568E
	C Language	U14566E
	Project Manager	U14569E
	Assembly Language	U14567E
ID850 (Ver.2.20 or Later) (Integrated Debugger)	Operation Windows™ Based	U14580E
SM850 (Ver.2.20 or Later) (System Simulator)	Operation Windows Based	U14782E
SM850 (Ver. 2.00 or Later) (System Simulator)	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)		U13737E
RD850 Pro (Ver.3.01) (Task Debugger)		U13916E
AZ850 (Ver. 3.0) (System Performance Analyzer)		U14410E
PG-FP3 (Flash Memory Programmer)		U13502E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

The V850 Family is a collection of NEC's single-chip microcontrollers that have a CPU core using the RISC microprocessor technology of the V800 Series™, with on-chip ROM/RAM and peripheral I/Os, etc.

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

The V850 Family has products that incorporate the V850 and V850E CPUs, however this manual targets products that incorporate the V850 CPU.

This chapter briefly outlines the V850 Family.

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
- Automobile 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 hitherto employed 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 Family of microcontrollers was developed to satisfy this need. This family uses RISC architecture that can provide 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 CISC single-chip microcontrollers at a lower total cost.

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

1.2 Architecture Features

- High-performance 32-bit architecture for embedded control
  - Number of instructions: 74
  - 32-bit general 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: 16 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)

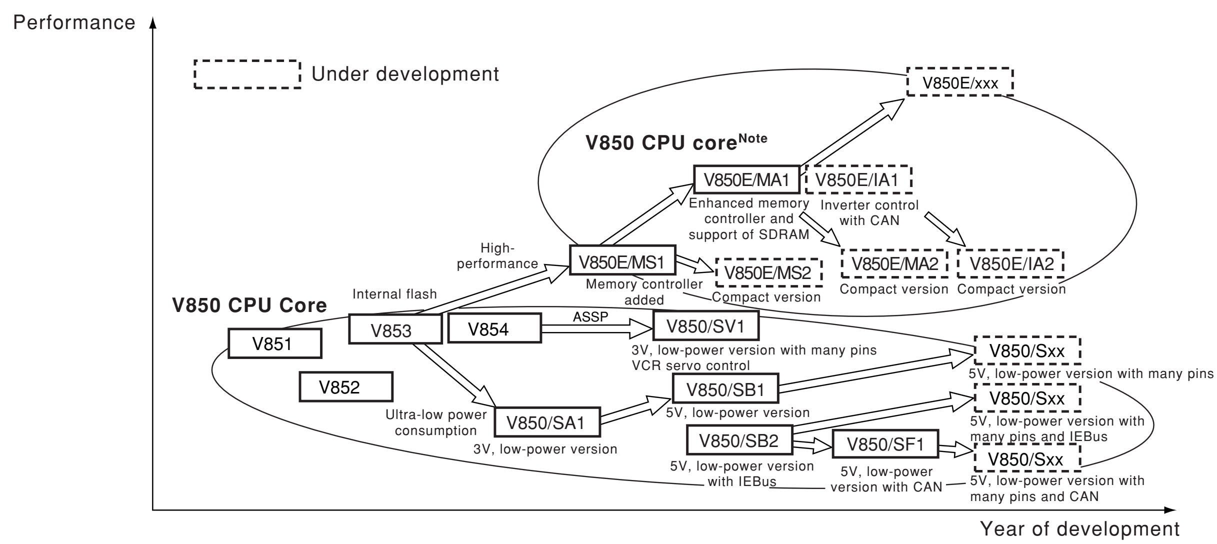
1.3 Product Development

The V850 Family consists of single-chip microcontrollers that use a RISC microprocessor core.

The V850 Family includes the V851, V852, V853, V854, V850/SA1, V850/SB1, V850/SB2, V850/SF1, and V850/SV1 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 products incorporating the V850 CPU are single-chip control-system microcontrollers, and the products incorporating the V850E CPU are single-chip microcontrollers that have enhanced external bus interface performance and that support not only the control-system but also data processing.

★ **Figure 1-1. V850 Family Product Development**

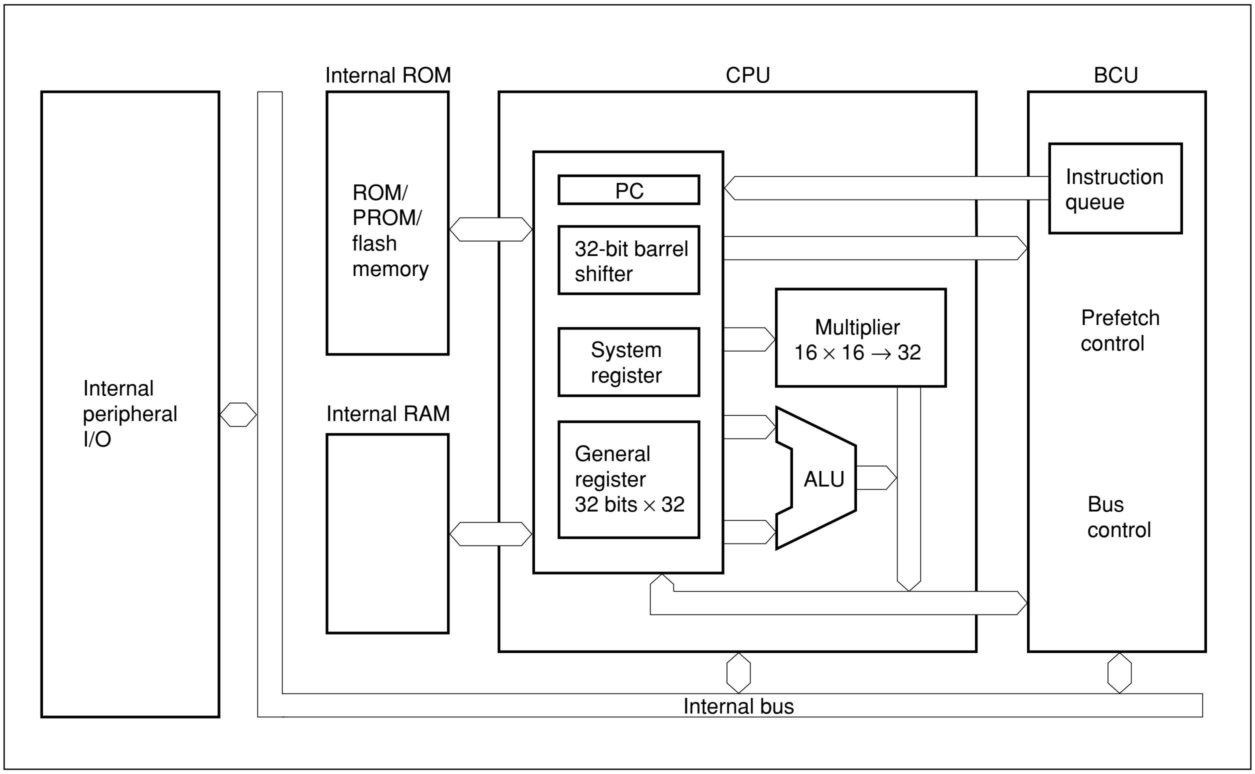


Note For details of the V850E CPU core architecture, refer to **V850E/MS1 User's Manual Architecture (U12197E)** and **V850E1 User's Manual Architecture (U14559E)**.

1.4 CPU Internal Configuration

Figure 1-2 shows the internal configuration of the V850 Family.

Figure 1-2. Internal Configuration



The function of each hardware block is as follows:

CPU ································ Executes almost all instructions such as address calculation, arithmetic and logical operation, and data transfer in one clock by using a 5-stage pipeline. Contains dedicated hardware such as a multiplier (16 × 16 bits) and a barrel shifter (32 bits/clock) to execute complicated instructions at high speeds.

Internal ROM ··················· ROM, EPROM, or flash memory mapped from address 00000000H. Can be accessed by the CPU in one clock during instruction fetch.

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.

BCU ································ Starts the required bus cycle based on a physical address obtained by the CPU. If the CPU does not issue a request for starting a bus cycle, the BCU generates a prefetch address, and prefetches an instruction code. The prefetched instruction code is loaded to an internal instruction queue.

CHAPTER 2 REGISTER SET

The register sets of the V850 Family can be classified into two types: program register sets that can be used for general programming, and system register sets that can control the execution environment. All the registers are 32 bits wide.

2.1 Program Registers

2.1.1 Program register set

(1) General registers

The V850 Family has thirty-two general 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. r2 may be used by the real-time OS. If the real-time OS used does not use r2, r2 can be used as a register for variable.

Figure 2-1. Program Register List

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. List of Program Register Operations

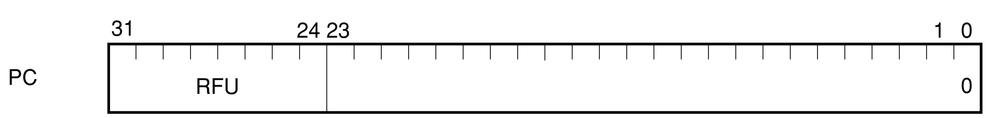
Name	Usage	Operation
r0	Zero register	Always holds 0.
r1	Assembler-reserved register	Used as working register for address generation.
r2	Address/data variable register (if real-time OS used does not use r2)	
r3	Stack pointer	Used for stack frame generation when function is called.
r4	Global pointer	Used 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 to r5, and r31 used by the assembler and C compiler, refer to **C Compiler Package (CA850) User's Manual**.

(2) Program counter

This register holds an instruction address during program execution. The lower 24 bits of this register are valid, and bits 31 through 24 are reserved fields (fixed to 0). If a carry occurs from bit 23 to 24, 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 Family 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

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 an exception or interrupt occurs.

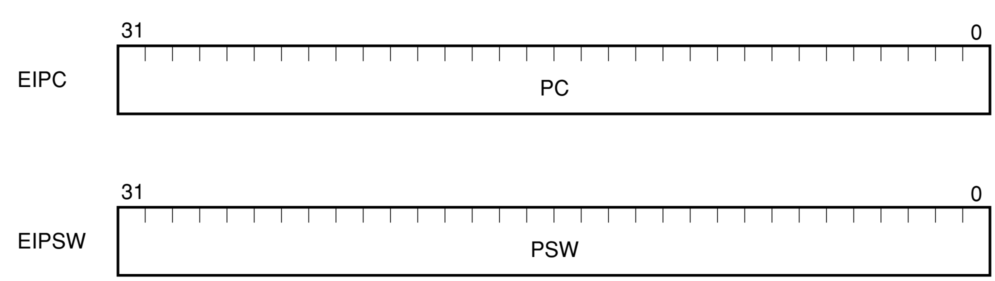
If an NMI occurs, however, the contents of the PC and PSW are saved to NMI status saving registers.

When an exception or interrupt occurs, the address of the following instruction is saved to the EIPC register. If an interrupt occurs while a division (DIVH) instruction is 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 24 through 31 of the EIPC and bits 8 through 31 of the EIPSW are fixed to 0.



2.2.2 NMI status saving registers

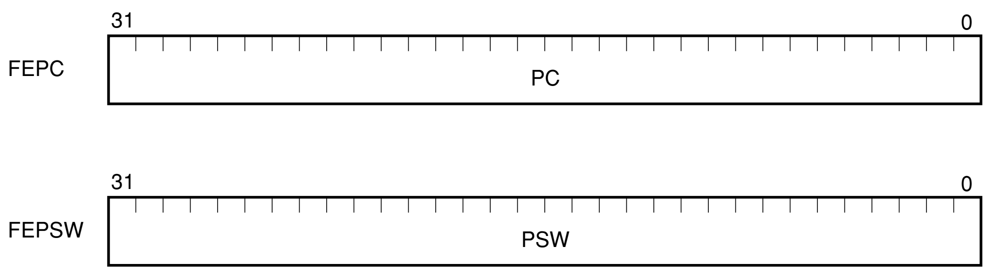
The V850 Family is provided with two NMI status saving registers: FEPC and FEPSW.

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) instruction is executed, the address of the division instruction under execution is saved).

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

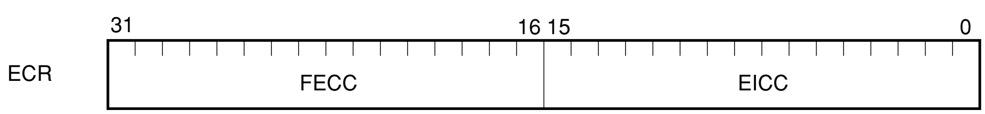
Bits 24 through 31 of the FEPC and bits 8 through 31 of the FEPSW are fixed to 0.



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, therefore no data can be written to it 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 under execution.

(1/2)

	31 to 8	7	6	5	4	3	2	1	0
PSW	RFU	NP	EP	ID	SAT	CY	OV	S	Z

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 granted. 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. 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 accepted. ID = 0: Interrupt can be accepted ID = 1: Interrupt cannot be accepted
4	SAT[Note]	Saturated Indicates that an overflow has occurred in a saturation 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 does not saturate. To reset this flag, load data to PSW. This flag is neither set to 1 nor reset to 0 by a 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

Remark Description of Note is given on the next page.

(2/2)

Note In the case of saturation instructions, the SAT, S, and OV flags will be set accordingly by 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	SAT-S-OV	Result of Saturation Processing
Maximum positive value is exceeded	1 0 1	7FFFFFFFH
Maximum negative value is exceeded	1 1 1	80000000H
Other	0 × 0	Operation result

2.2.5 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	√	√
6 to 31	Reserved		

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

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

CHAPTER 3 DATA TYPE

3.1 Data Format

The V850 Family supports the following data types:

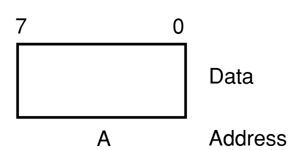
- Integer (8, 16, 32 bits)
- Unsigned integer (8, 16, 32 bits)
- Bit

3.1.1 Data type and addressing

The V850 Family supports three types of data lengths: word (32 bits), half-word (16 bits), and byte (8 bits). Byte 0 of any data is always the least significant byte (this is called little endian) and 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 memory.

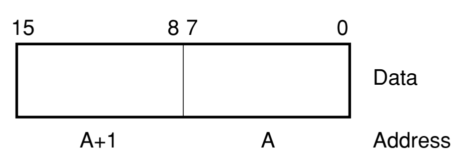
(1) Byte (BYTE)

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.



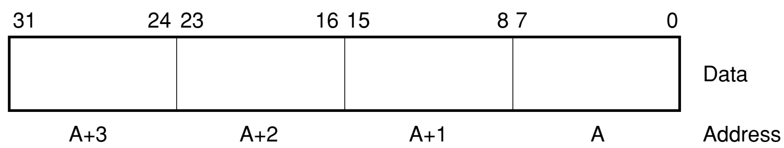
(2) Half-word (HALF-WORD)

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



(3) Word (WORD)

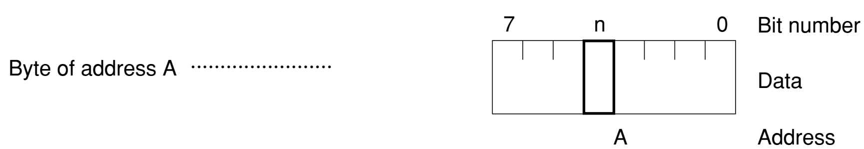
A word is 4-byte (32-bit) contiguous data that starts from any word boundary[Note]. Each bit is assigned a number from 0 to 31. The LSB is bit 0 and the MSB is bit 31. A word is specified by its address A (with the 2 lowest bits fixed to 0), and occupies 4 bytes A, A+1, A+2, and A+3.



(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 See **3.3 Data Alignment**.



3.2 Data Representation

3.2.1 Integer

With the V850 Family, 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
Half-word	16 bits	–32,768 to +32,767
Word	32 bits	–2,147,483,648 to +2,147,483,647

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
Half-word	16 bits	0 to 65,535
Word	32 bits	0 to 4,294,967,295

3.2.3 Bit

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

- Set
- Clear
- Invert
- Test

3.3 Data Alignment

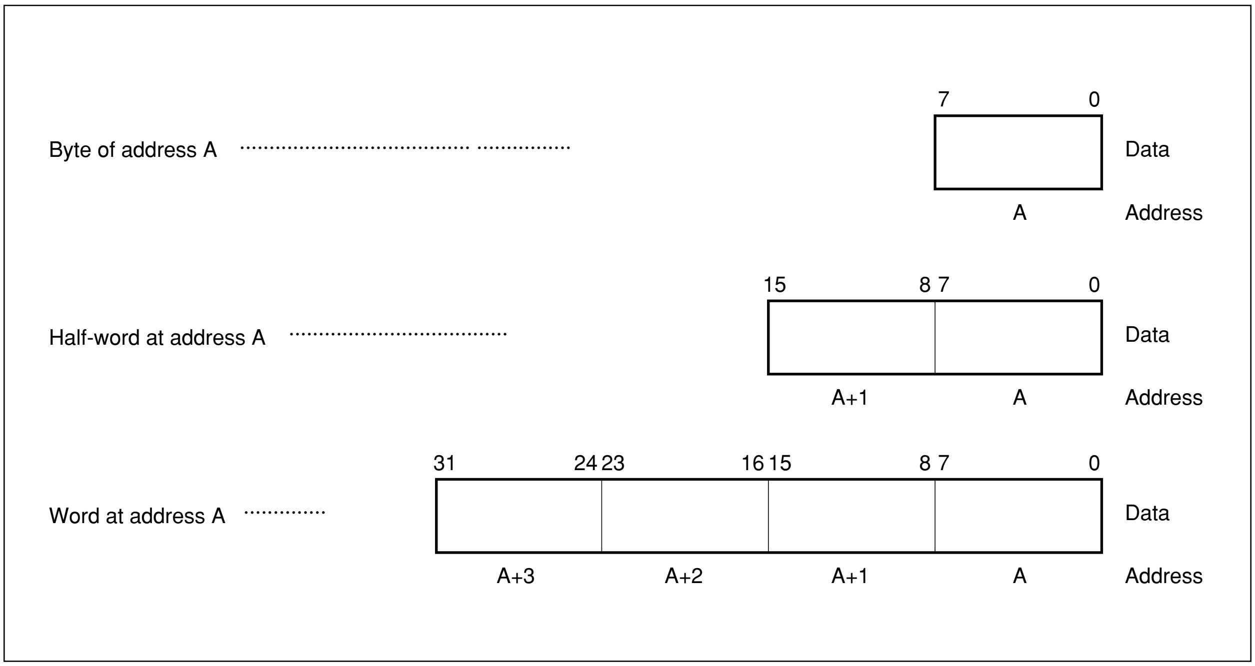
With the V850 Family, word data to be allocated in memory must be aligned at an appropriate boundary. Therefore, word data must be aligned at a word boundary (the lower 2 bits of the address are 0), and half-word data must be aligned at a half-word boundary (the lowest bit of the address is 0). If data is not aligned at a boundary, 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 half-word data) automatically masked. Byte data can be placed at any address.

Note that the process of aligning is called alignment.

The V850 Family supports a 4-GB linear address space. Both memory and I/O are mapped to this address space (**memory-mapped I/O**). The V850 Family 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 (**little endian**).

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



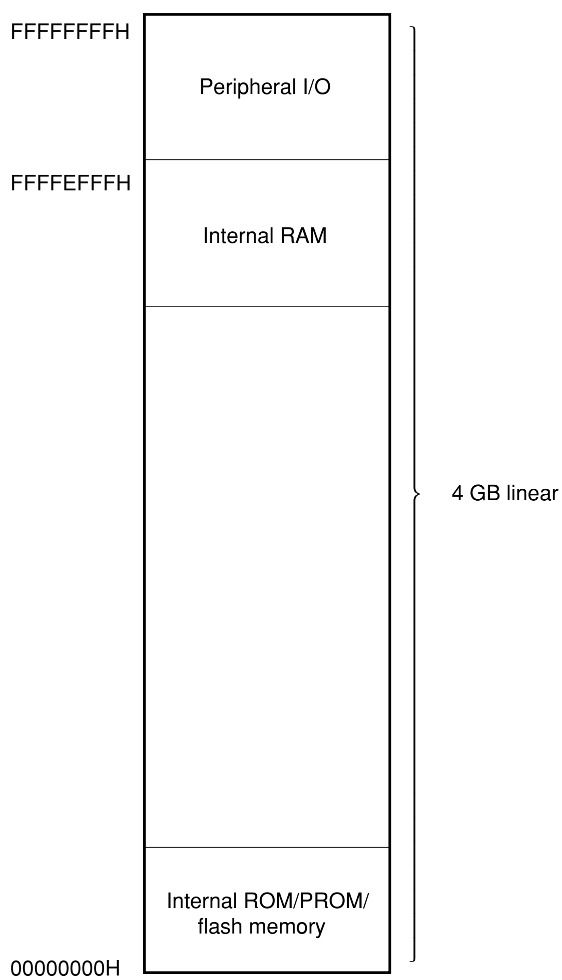
4.1 Memory Map

The V850 Family 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 16 MB for instruction addressing.

Figure 4-1 shows the memory map of the V850 Family.

The capacity of the on-chip ROM and RAM depends on each product. For details, refer to the memory map section in the User's Manual Hardware of each product.

Figure 4-1. Memory Map



4.2 Addressing Modes

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 address (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 the Bcond disp9, JR disp22, and JARL disp22, reg2 instructions.

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

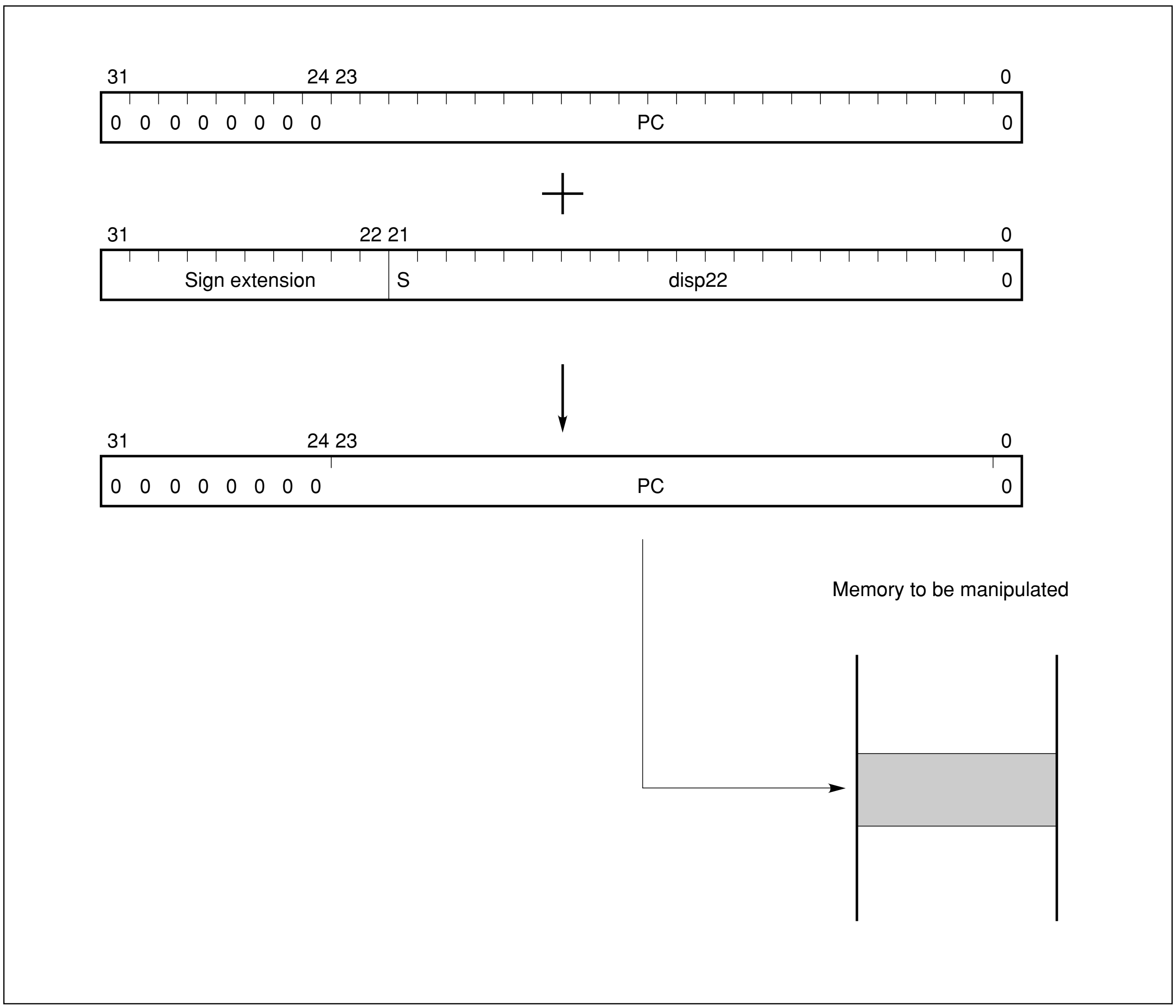
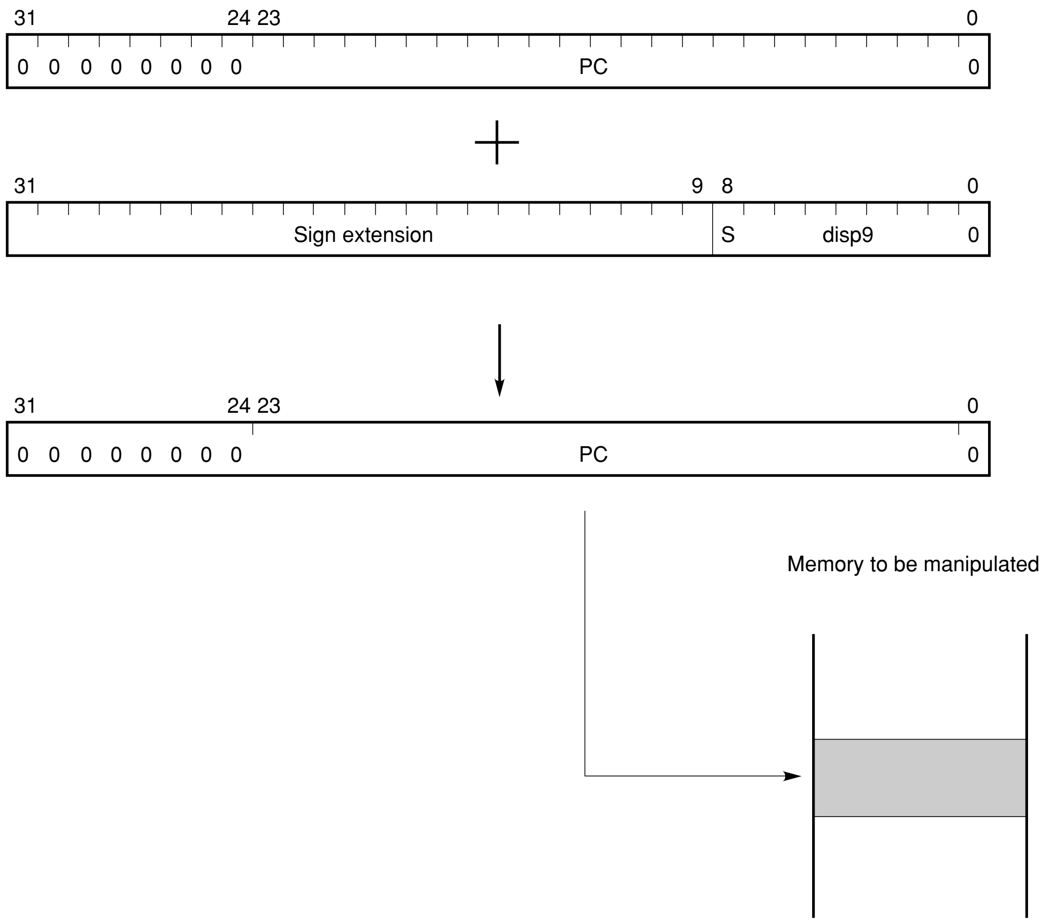


Figure 4-3. Relative Addressing (Bcond disp9)

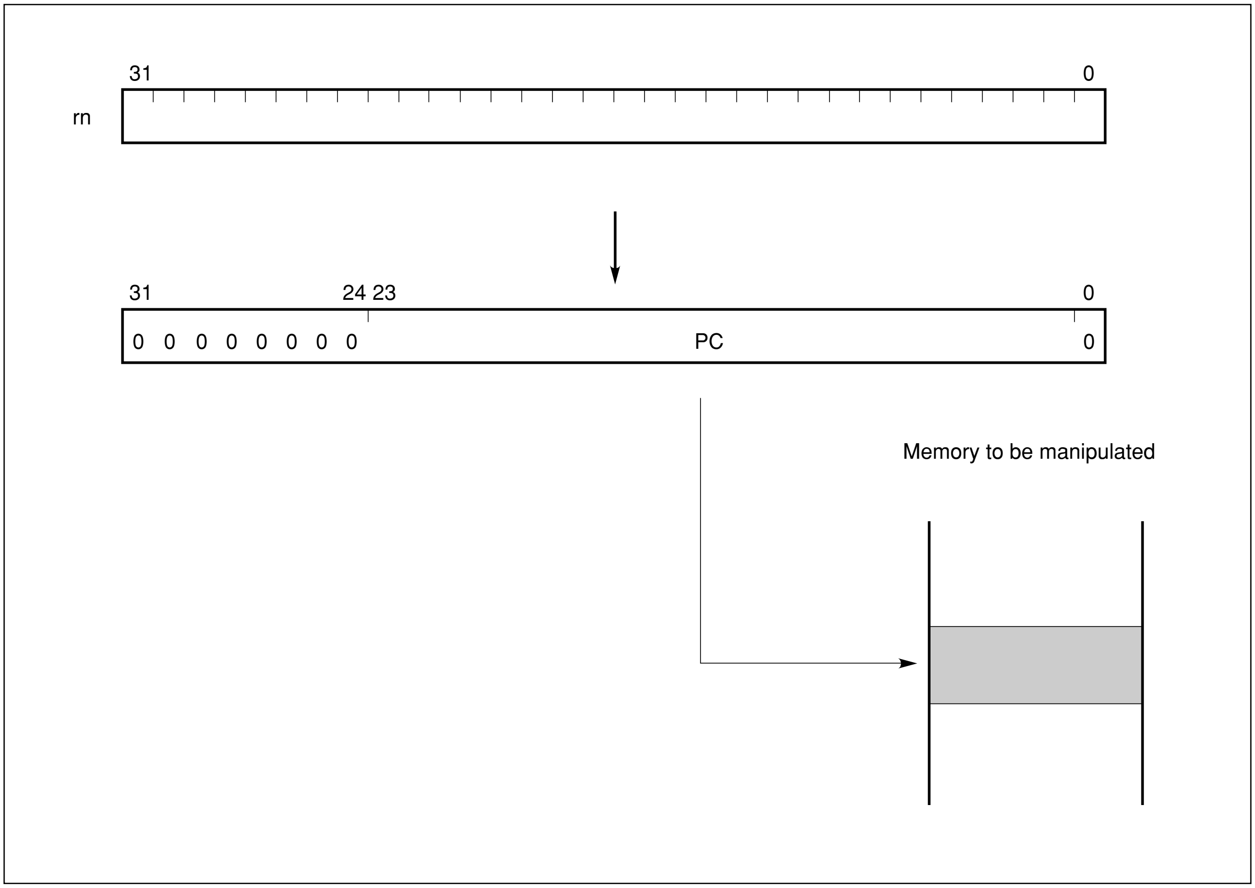


(2) Register addressing (register indirect)

The contents of a general 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 register (or system register) specified in the general register specification field is accessed as an 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: Operand that is 5-bit immediate data to specify the trap vector (00H to 1FH), and is used by the TRAP instruction.

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

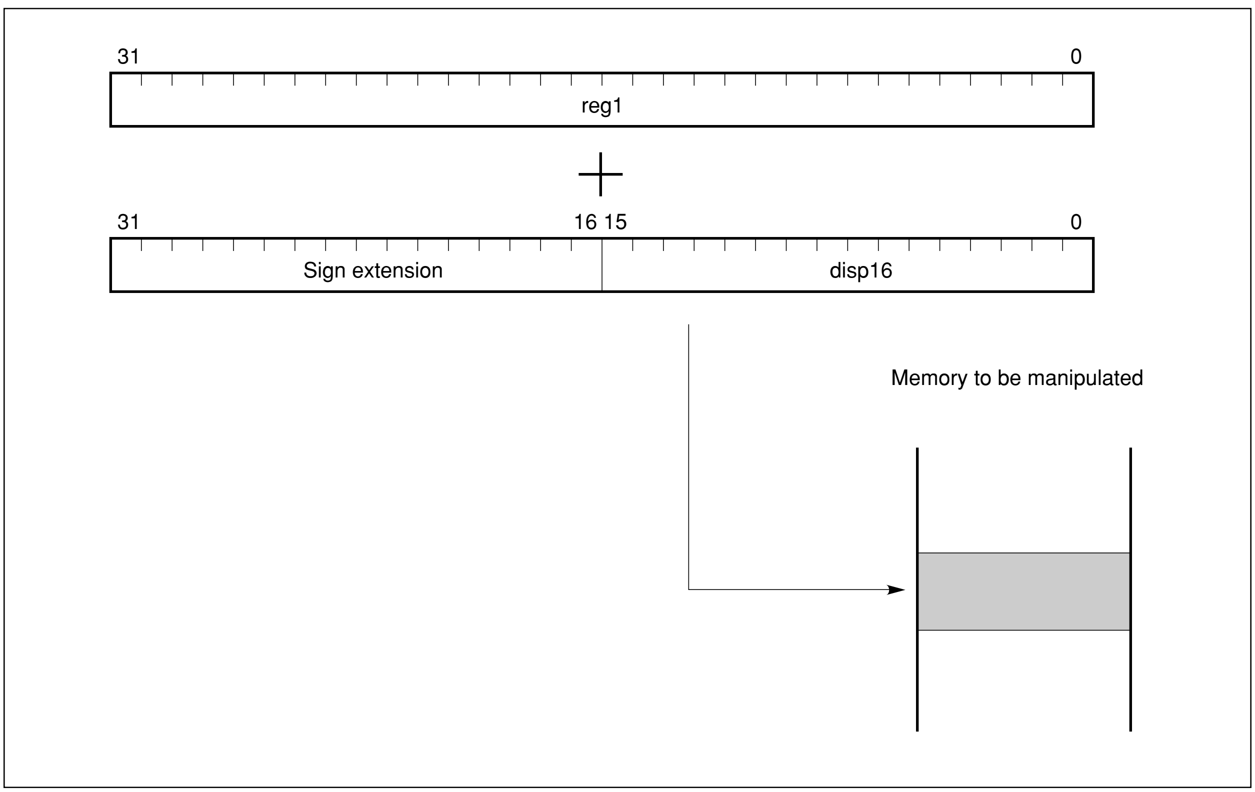
(3) Based addressing

The following two types of based addressing are supported:

(a) Type 1

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

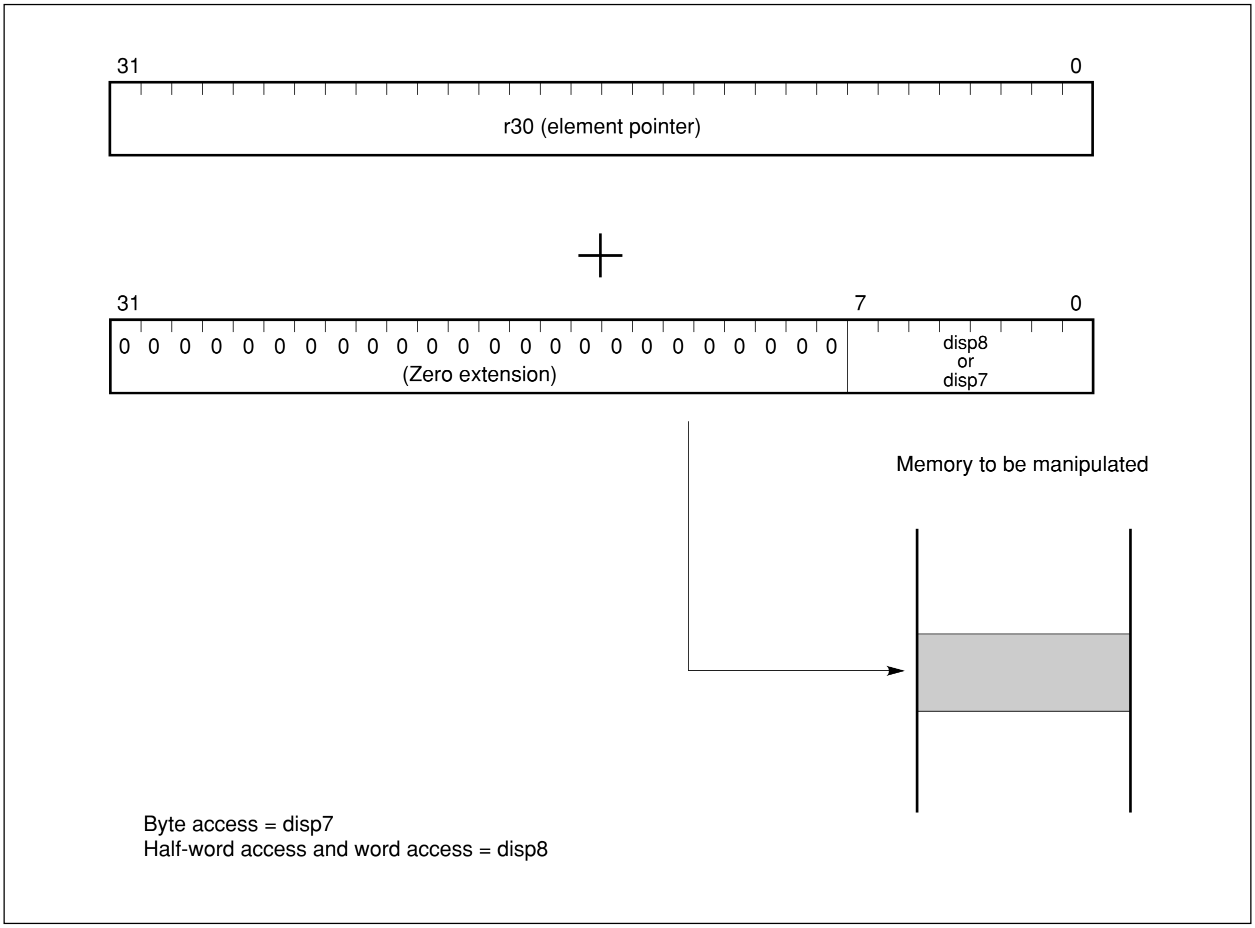
Figure 4-5. Based Addressing (Type 1)



(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.

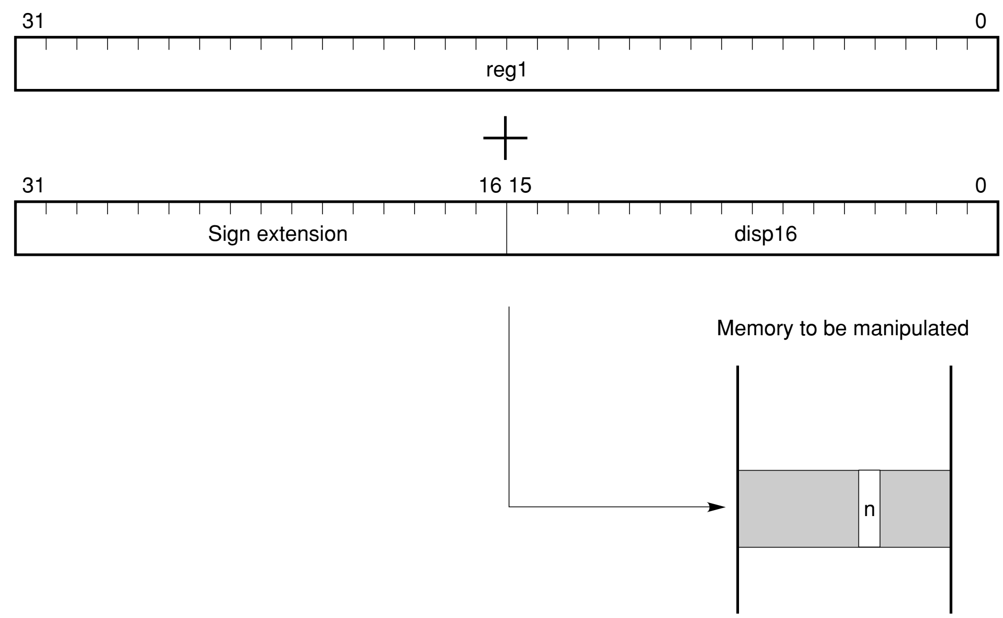
Figure 4-6. Based Addressing (Type 2)



(4) Bit addressing

This addressing is used to access 1 bit (specified with bit#3 of 3-bit data) among 1 byte of the memory space to be manipulated by using an operand address which is the sum of the contents of a general register and a 16-bit displacement sign-extended to a 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)

5.1 Instruction Format

The V850 Family 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.

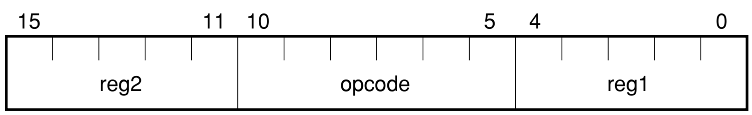
Some instructions have an unused field (RFU). This field is reserved for future expansion and must be fixed to 0.

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 31) → higher address

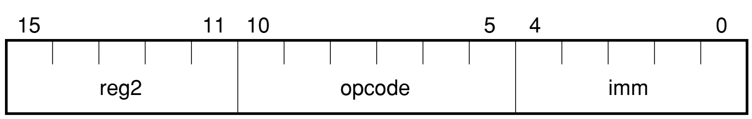
(1) reg-reg instruction (Format I)

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



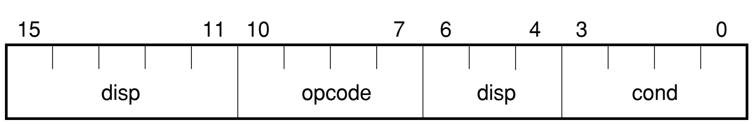
(2) imm-reg instruction (Format II)

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



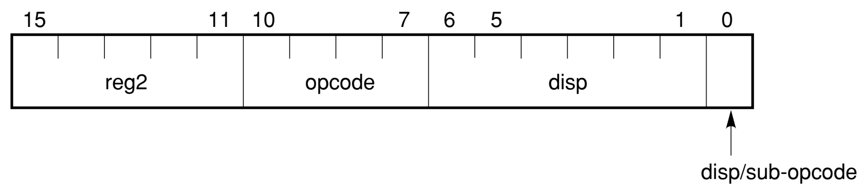
(3) Conditional branch instruction (Format III)

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



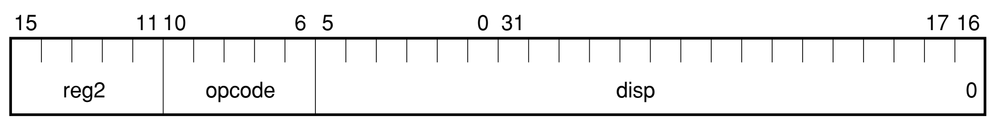
(4) 16-bit load/store instruction (Format IV)

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



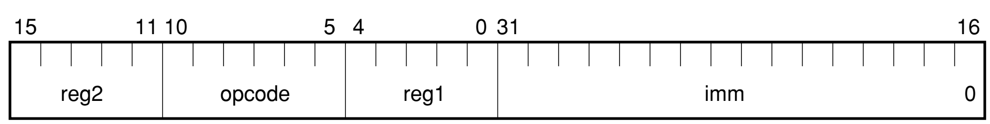
(5) Jump instruction (Format V)

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



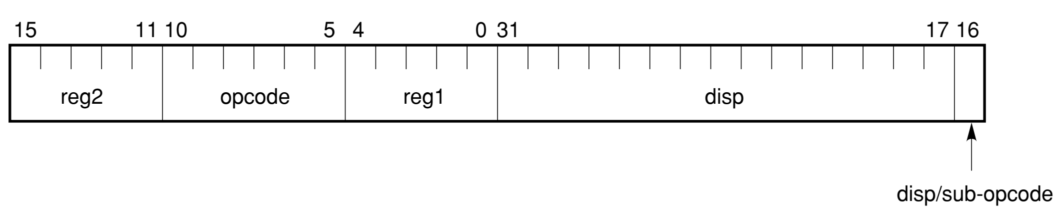
(6) 3-operand instruction (Format VI)

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



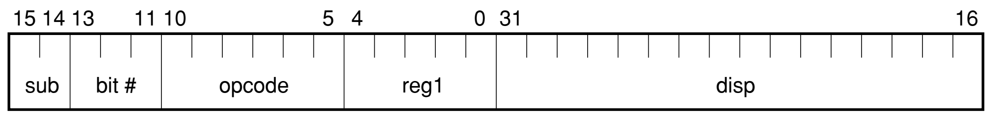
(7) 32-bit load/store instruction (Format VII)

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



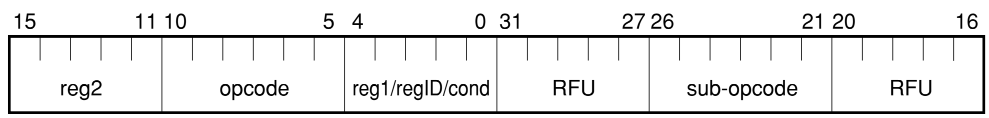
(8) Bit manipulation instruction (Format VIII)

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



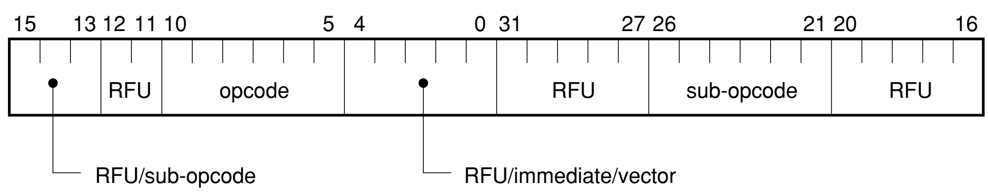
(9) Extended instruction format 1 (Format IX)

A 32-bit instruction format having a 6-bit op code field, 6-bit sub-op code, and two general 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 op code field and 6-bit sub op code.



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
MULH
MULHI
DIVH
CMP
SETF

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 and shift instructions. 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

Branch instructions .... Branch operations 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 a 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 in any of the categories of instructions described above.

Table 5-7. Special Instructions

LDSR
STSR
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 description of an operand:

Symbol	Meaning
reg1	General register (used as source register)
reg2	General register (mainly used as destination register. Some are also used as source registers)
bit#3	3-bit data for specification bit number
imm×	×-bit immediate
disp×	×-bit displacement
regID	System register number
vector	5-bit data for trap vector (00H to 1FH) specification
cccc	4-bit data for condition code specification
ep	Element pointer (r30)

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

Symbol	Meaning
←	Assignment
GR [ ]	General register
SR [ ]	System register
zero-extend (n)	Zero-extends n to word
sign-extend (n)	Sign-extends n to word
load-memory (a, b)	Reads data of size b from address a
store-memory (a, b, c)	Writes data b of size c to address a
load-memory-bit (a, b)	Reads bit b from address a
store-memory-bit (a, b, c)	Writes c to bit b of address a
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)
Halfword	Half-word (16 bits)
Word	Word (32 bits)
+	Add
–	Subtract
||	Bit concatenation
×	Multiply
÷	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.

Op code 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
d	1-bit data of displacement
i	1-bit data of immediate
cccc	4-bit data for condition code specification
bbb	3-bit data for bit number specification

Flag Indicates the flags which 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	Mnemonic	Function
	Load/Store instructions		Logical operation instructions
SLD.B	Load Byte	TST	Test
SLD.H	Load Half-Word	OR	Or
SLD.W	Load Word	ORI	Or Immediate
LD.B	Load Byte	AND	And
LD.H	Load Half-Word	ANDI	And Immediate
LD.W	Load Word	XOR	Exclusive-Or
SST.B	Store Byte	XORI	Exclusive-Or Immediate
SST.H	Store Half-Word	NOT	Not
SST.W	Store Word	SHL	Shift Logical Left
ST.B	Store Byte	SHR	Shift Logical Right
ST.H	Store Half-Word	SAR	Shift Arithmetic Right
ST.W	Store Word		Branch instructions
	Arithmetic instructions	JMP	Jump
MOV	Move	JR	Jump Relative
MOVHI	Move High Half-Word	JARL	Jump and Register Link
MOVEA	Move Effective Address	Bcond	Branch on Condition Code
ADD	Add		Bit manipulation instructions
ADDI	Add Immediate	SET1	Set Bit
SUB	Subtract	CLR1	Clear Bit
SUBR	Subtract Reverse	NOT1	Not Bit
MULH	Multiply Half-Word	TST1	Test Bit
MULHI	Multiply Half-Word Immediate		Special instructions
DIVH	Divide Half-Word	LDSR	Load System Register
CMP	Compare	STSR	Store System Register
SETF	Set Flag Condition	TRAP	Trap
	Saturate instructions	RETI	Return from Trap or Interrupt
SATADD	Saturated Add	HALT	Halt
SATSUB	Saturated Subtract	DI	Disable Interrupt
SATSUBI	Saturated Subtract Immediate	EI	Enable Interrupt
SATSUBR	Saturated Subtract Reverse	NOP	No Operation

Most of these mnemonics are acronyms of functions.

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

Op code

(1)

```
15              0
rrrrr001110RRRRR
```

(2)

```
15              0
rrrrr010010iiiii
```

Flag

CY	1 if a carry occurs from MSB; otherwise, 0.
OV	1 if an 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 register reg1 to the word data of general register reg2, and stores the result in general register reg2. The data of general register reg1 is not affected.
(2) Adds 5-bit immediate data, sign-extended to word length, to the word data of general register reg2, and stores the result in general register reg2.

ADDI

Add Immediate

Instruction format ADDI imm16, reg1, reg2

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

Format Format VI

Op code

15 0	31 16
rrrrr110000RRRRR	iiiiiiiiiiiiiiii

Flag

CY 1 if a carry occurs from MSB; otherwise, 0.
OV 1 if an 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 register reg1, and stores the result in general register reg2. The data of general register reg1 is not affected.

AND

And

Instruction format AND reg1, reg2

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

Format Format I

Op code

```
15             0
rrrrr001010RRRRR
```

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 register reg2 with the word data of general register reg1, and stores the result in general register reg2. The data of general 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

Op code

```
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 register reg1 with the value of the 16-bit immediate data, zero-extended to word length, and stores the result in general register reg2. The data of general 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

Op code

```
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 a condition flag specified by the instruction. Branches if a 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. List of 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 S and OV flags will therefore, have no meaning.

CLR1

Clear Bit

Instruction format CLR1 bit#3, disp16 [reg1]

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

Format Format VIII

Op code

15 … 0	31 … 16
10bbb111110RRRRR	dddddddddddddddd

Flag

CY	–
OV	–
S	–
Z	1 if bit NO.bit#3 of memory disp16 [reg1] = 0.
	0 if bit NO.bit#3 of memory disp16 [reg1] = 1.
SAT	–

Instruction CLR1 Clear Bit

Explanation Adds the data of general register reg1 to the 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Then clears the bit, specified by the bit number of 3 bits, of the byte data referenced by the generated address. Not specified bit is not affected.

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

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

Op code

(1)
```
15            0
rrrrr001111RRRRR
```

(2)
```
15            0
rrrrr010011iiiii
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg2 with the word data of general register reg1, and indicates the result by using the condition flags. To compare, the contents of general register reg1 are subtracted from the word data of general register reg2. The data of general registers reg1 and reg2 is not affected.

(2) Compares the word data of general 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 is subtracted from the word data of general register reg2. The data of general register reg2 is not affected.

DI

Disable Interrupt

Instruction format DI

Operation PSW.ID ← 1 (Disables maskable interrupt)

Format Format X

Op code

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 executing 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 (NMI) are not affected by this instruction.

DIVH

Divide Half-Word

Instruction format DIVH reg1, reg2

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

Format Format I

Op code

```
15            0
rrrrr000010RRRRR
```

Flag

CY	–
OV	1 if an 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 Divides the word data of general register reg2 by the lower half-word data of general register reg1, and stores the quotient in general register reg2. If the data is divided by 0, Overflow occurs, and the quotient is undefined. The data of general register reg1 is not affected.

Remark The remainder is not stored. 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 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 registers reg1 and reg2 will retain their original values prior to the start of execution.

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

EI

Enable Interrupt

Instruction format EI

Operation PSW.ID ← 0 (enables maskable interrupt)

Format Format X

Op code

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

Op code

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 exited by any of the following three events:

- RESET input
- NMI input
- Maskable interrupt (when ID of PSW = 0)

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

JARL

Jump and Register Link

Instruction format JARL disp22, reg2

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

Format Format V

Op code

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 register reg2, adds the current PC value and 22-bit displacement, sign-extended to word length, and transfers control to that PC. Bit 0 of the 22-bit displacement is masked to 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 saves the PC return address to general register reg2. The JMP instruction, which is equivalent to a subroutine-return instruction, can be used to specify the general register containing the return address saved during the JARL subroutine-call instruction, to restore the program counter.

JMP

Jump Register

Instruction format JMP [reg1]

Operation PC ← GR [reg1]

Format Format I

Op code

```
15               0
0000000000011RRRRR
```

Flag

CY –
OV –
S –
Z –
SAT –

Instruction JMP Jump Register

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

Remark When using this instruction as the subroutine-return instruction, specify the general register containing the return address saved during the JARL subroutine-call instruction, to restore the program counter. When using the JARL instruction, which is equivalent to the subroutine-call instruction, store the PC return address in general register reg2.

JR

Jump Relative

Instruction format JR disp22

Operation PC ← PC + sign-extend (disp22)

Format Format V

Op code

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 that PC. Bit 0 of the 22-bit displacement is masked to 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

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, Halfword))
(3) adr ← GR [reg1] + sign-extend (disp16)
GR [reg2] ← Load-memory (adr, Word)

Format Format VII

Op code

(1)

15 0	31 16
rrrrr111000RRRRR	dddddddddddddddd

(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.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction (1) LD.B Load Byte
(2) LD.H Load Half-Word
(3) LD.W Load Word

Explanation

(1) Adds the data of general 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 then stored in general register reg2.

(2) Adds the data of general register reg1 to a 16-bit displacement sign-extended to word length to generate a 32-bit address. Half-word data is read from this 32-bit address with its bit 0 masked to 0, sign-extended to word length, and stored in general register reg2.

(3) Adds the data of general 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 to 0, and stored in general register reg2.

Caution

When the data of general register reg1 is added to a 16-bit displacement sign-extended to word length, the lower bits of the result may be masked to 0 depending on the type of data to be accessed (half word, word) to generate an address.

LDSR

Load to System Register

Instruction format LDSR reg2, regID

Operation SR [regID] ← GR [reg2]

Format Format IX

Op code

```
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 – (See **Remark** below.)
OV – (See **Remark** below.)
S – (See **Remark** below.)
Z – (See **Remark** below.)
SAT – (See **Remark** below.)

Instruction LDSR Load to System Register

Explanation Loads the word data of general register reg2 to a system register specified by the system register number (regID). The data of general 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, the reserved bits remain at 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

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

Format
(1) Format I
(2) Format II

Op code

(1)

15	0
rrrrr000000RRRRR	

(2)

15	0
rrrrr010000iiiii	

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Instruction
(1) MOV Move Register
(2) MOV Move Immediate (5-Bit)

Explanation
(1) Transfers the word data of general register reg1 to general register reg2. The data of general register reg1 is not affected.
(2) Transfers the value of a 5-bit immediate data, sign-extended to word length, to general register reg2.

MOVEA

Move Effective Address

Instruction format MOVEA imm16, reg1, reg2

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

Format Format VI

Op code

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 register reg1, and stores the result in general register reg2. The data of general register reg1 is not affected. The flags are not affected by the addition.

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

Op code

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 register reg1 and stores the result in general register reg2. The data of general register reg1 is not affected. The flags are not affected by the addition.

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

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

Op code

(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 half-word data of general register reg2 by the half-word data of general register reg1, and stores the result in general register reg2 as word data. The data of general register reg1 is not affected.
(2) Multiplies the lower half-word data of general register reg2 by a 5-bit immediate data, sign-extended to half-word length, and stores the result in general register reg2.

Remark
The higher 16 bits of general 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

Op code

```
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 half-word data of general register reg1 by the 16-bit immediate data, and stores the result in general register reg2. The data of general register reg1 is not affected.

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

NOP

No Operation

Instruction format NOP

Operation Executes nothing and consumes at least one clock.

Format Format I

Op code

```
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 op code 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

Op code

```
15                0
rrrrr000001RRRRR
```

Flag

CY	–
OV	–
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 register reg1, and stores the result in general register reg2. The data of general register reg1 is not affected.

NOT1

Not Bit

Instruction format NOT1 bit#3, disp16 [reg1]

Operation

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

Format Format VIII

Op code

15 0	31 16
01bbb111110RRRRR	dddddddddddddddd

Flag

CY –
OV –
S –
Z 1 if bit NO.bit#3 of memory disp16 [reg1] = 0.
0 if bit NO.bit#3 of memory disp16 [reg1] = 1.
SAT –

Instruction NOT1 Not Bit

Explanation

Adds the data of general register reg1 to a 16-bit displacement, sign-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. The 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 is executed, and does not indicate the content 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

Op code

```
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 register reg2 with the word data of general register reg1, and stores the result in general register reg2. The data of general 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

Op code

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

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

Op code

15 0	31 16
0000011111100000	0000000101000000

Flag

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

Instruction RETI Return from Trap or Interrupt

Explanation This instruction restores the restore PC and PSW from the appropriate system register and 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 the 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 the 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 the EIPC and EIPSW.

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

Caution When restoring from an NMI or exception processing 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 processing using the RETI instruction:
  PSW.NP = 1 and PSW.EP = 0
- When restoring from an exception processing using the RETI instruction:
  PSW.EP = 1

Use the LDSR instruction for setting the flags.

Interrupts are not accepted 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

Op code

(1)

15 0	31 16
rrrrr111111RRRRR	0000000010100000

(2)

15 0
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 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 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 to general register reg2. If the number of shifts is 0, general register reg2 retains the same value prior to instruction execution. The data of general register reg1 is not affected.
(2) Arithmetically shifts the word data of general 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 to general register reg2. If the number of shifts is 0, general register reg2 retains the same value prior to instruction execution.

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

Op code

(1)
```
15              0
rrrrr000110RRRRR
```

(2)
```
15              0
rrrrr010001iiiii
```

Flag

CY	1 if a carry occurs from MSB; otherwise, 0.
OV	1 if an 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 register reg1 to the word data of general register reg2, and stores the result in general 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 register reg1 is not affected.
(2) Adds a 5-bit immediate data, sign-extended to word length, to the word data of general register reg2, and stores the result in general 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.

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

Op code

```
15           0
rrrrr000101RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg1 from the word data of general register reg2, and stores the result in general 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 register reg1 is not affected.

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

Op code

15 0	31 16
rrrrr110011RRRRR	iiiiiiiiiiiiiiii

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg1, and stores the result in general 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 register reg1 is not affected.

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

Op code

```
15              0
rrrrr000100RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg2 from the word data of general register reg1, and stores the result in general 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 register reg1 is not affected.

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

Op code

15 0	31 16
rrrrr1111110cccc	0000000000000000

Flag
CY –
OV –
S –
Z –
SAT –

Instruction SETF Set Flag Condition

Explanation The general register reg2 is set to 1 if a condition specified by condition code "cccc" is satisfied; otherwise, 0 are stored in the register. One of the codes shown in Table 5-9 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 "conditional branch" by checking the result of evaluation equivalent to a_n. 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 to register Ra. By performing a logical operation to 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 to general 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 SET1 bit#3, disp16 [reg1]

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

Format Format VIII

Op code

15 0	31 16
00bbb111110RRRRR	dddddddddddddddd

Flag

CY –
OV –
S –
Z 1 when bit NO.bit#3 of memory disp16 [reg1] = 0.
0 when bit NO.bit#3 of memory disp16 [reg1] = 1
SAT –

Instruction SET1 Set Bit

Explanation Adds the 16-bit displacement, sign-extended to word length, to the data of general register reg1 to generate a 32-bit address. The bit, specified by the 3-bit field “bbb”, is set at the byte data location referenced by the generated address. The 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 is 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

Op code

(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 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 register reg1 (0 is shifted to the LSB side), and then writes the result to general register reg2. If the number of shifts is 0, general register reg2 retains the same value prior to instruction execution. The data of general register reg1 is not affected.

(2) Logically shifts the word data of general 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 to general register reg2. If the number of shifts is 0, general 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

Op code

(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 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 register reg1 (0 is shifted to the MSB side). This instruction then writes the result to general register reg2. If the number of shifts is 0, general register reg2 retains the same value prior to instruction execution. The data of general register reg1 is not affected.

(2) Logically shifts the word data of general 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 to general register reg2. If the number of shifts is 0, general register reg2 retains the same value prior to instruction execution.

SLD

Load

Instruction format

(1) SLD.B disp7 [ep], reg2
(2) SLD.H disp8 [ep], reg2
(3) SLD.W disp8 [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, Halfword))
(2) adr ← ep + zero-extend (disp8)
GR [reg2] ← Load-memory (adr, Word)

Format

Format IV

Op code

(1)
```
15              0
rrrrr0110ddddddd
```

(2)
```
15              0
rrrrr1000ddddddd
```

ddddddd is the higher 7 bits of disp8.

(3)
```
15              0
rrrrr1010dddddd0
```

dddddd is the higher 6 bits of disp8.

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

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. Half-word data is read from this 32-bit address with bit 0 masked to 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 to 0, and stored in reg2.

Caution

When the element pointer is added to the 8-bit displacement zero extended to word length, the lower bits of the result may be masked to 0 depending on the type of data to be accessed (half word, word).

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], Halfword)
(2) adr ← ep + zero-extend (disp8)
Store-memory (adr, GR [reg2], Word)

Format
Format IV

Op code

(1)
```
15             0
rrrrr0111ddddddd
```

(2)
```
15             0
rrrrr1001ddddddd
```

ddddddd is the higher 7 bits of disp8.

(3)
```
15             0
rrrrr1010dddddd1
```

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 at 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 half-word data of reg2 at the generated 32-bit address with bit 0 masked to 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 at the generated 32-bit address with bits 0 and 1 masked to 0.

Caution

When the element pointer is added to the 8-bit displacement zero-extended to word length, the lower bits of the result may be masked to 0 depending on the type of data to be accessed (half word, word).

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], Halfword)
(2) adr ← GR [reg1] + sign-extend (disp16)
Store-memory (adr, GR [reg2], Word)

Format

Format VII

Op code

```
    15              0 31             16
(1) |rrrrr111010RRRRR|dddddddddddddddd|

    15              0 31             16
(2) |rrrrr111011RRRRR|ddddddddddddddd0|
```

ddddddddddddddd is the higher 15 bits of disp16.

```
    15              0 31             16
(3) |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 register reg1 to generate a 32-bit address, and stores the lowest byte data of general register reg2 at the generated address.

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

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

Caution

When the data of general register reg1 is added to a 16-bit displacement sign-extended to word length, the lower bits of the result may be masked to 0 depending on the type of data to be accessed (half word, word) to generate an address.

STSR

Store Contents of System Register

Instruction format STSR regID, reg2

Operation GR [reg2] ← SR [regID]

Format Format IX

Op code

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 system register number (regID) in general 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 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] – [reg1]

Format Format I

Op code

```
15            0
rrrrr001101RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg1 from the word data of general register reg2, and stores the result in general register reg2. The data of general register reg1 is not affected.

SUBR

Subtract Reverse

Instruction format SUBR reg1, reg2

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

Format Format I

Op code

```
15             0
rrrrr001100RRRRR
```

Flag

CY	1 if a borrow to MSB occurs; otherwise, 0.
OV	1 if an 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 register reg2 from the word data of general register reg1, and stores the result in general register reg2. The data of general register reg1 is not affected.

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

Op code

15 0	31 16
00000111111iiiii	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 to 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

Op code

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

TST1

Test Bit

Instruction format TST1 bit#3, disp16 [reg1]

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

Format Format VIII

Op code

15 0	31 16
11bbb111110RRRRR	dddddddddddddddd

Flag

CY	–
OV	–
S	–
Z	1 if bit NO.bit#3 of memory disp16 [reg1] = 0. 0 if bit NO.bit#3 of memory disp16 [reg1] = 1.
SAT	–

Instruction TST1 Test Bit

Explanation Adds the data of general 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.

XOR

Exclusive Or

Instruction format XOR reg1, reg2

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

Format Format I

Op code

```
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 register reg2 with the word data of general register reg1, and stores the result in general register reg2. The data of general 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

Op code

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

5.4 Number of Instruction Execution Clock Cycles

The number of instruction execution clock cycles differs depending on the combination of instructions. For details, see **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)

Instructions	Mnemonic	Operand	Byte	Execution clock
				i – r – l
Load/store	SLD.B	disp7 [ep], r	2	1 – 1 – 2
	SLD.H	disp8 [ep], r	2	1 – 1 – 2
	SLD.W	disp8 [ep], r	2	1 – 1 – 2
	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
	LD.B	disp16 [R], r	4	1 – 1 – 2
	LD.H	disp16 [R], r	4	1 – 1 – 2
	LD.W	disp16 [R], r	4	1 – 1 – 2
	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
	MOVEA	imm16, R, r	4	1 – 1 – 1
	MOVHI	imm16, R, r	4	1 – 1 – 1
	DIVH	R, r	2	36 – 36 – 36
	MULH	R, r	2	1 – 1 – 2
	MULH	imm5, r	2	1 – 1 – 2
	MULHI	imm16, R, r	4	1 – 1 – 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
	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

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

Instructions	Mnemonic	Operand		Byte	Execution clock
					i – r – l
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
Branch	JMP	[R]		2	3 – 3 – 3
	JR	disp22		4	3 – 3 – 3
	JARL	disp22, r		4	3 – 3 – 3
	Bcond	disp9	When condition is satisfied	2	3 – 3 – 3
			When condition is not satisfied	2	1 – 1 – 1
Bit manipulation	SET1	bit#3, disp16 [R]		4	4 – 4 – 4
	CLR1	bit#3, disp16 [R]		4	4 – 4 – 4
	NOT1	bit#3, disp16 [R]		4	4 – 4 – 4
	TST1	bit#3, disp16 [R]		4	3 – 3 – 3
Special	LDSR	R, SR		4	1 – 1 – Note
	STSR	SR, r		4	1 – 1 – 1
	NOP	–		2	1 – 1 – 1
	DI	–		4	1 – 1 – 1
	EI	–		4	1 – 1 – 1
	TRAP	vector		4	4 – 4 – 4
	HALT	–		4	1 – 1 – 1
	RETI	–		4	4 – 4 – 4
	Undefined instruction code trap			4	4 – 4 – 4

Note When accessing EIPC, FEPC: 3
When accessing EIPSW, FEPSW, PSW: 1

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

Operand

Symbol	Meaning
R: reg1	General register (used as source register)
r: reg2	General register (mainly used as destination register)
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: Halfword	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

Execution clock

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	If result of instruction execution is quoted by immediately subsequent instruction

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. There is no major difference between interrupts and exceptions in terms of control flow.

The V850 Family 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 the occurrence of an exception event (exception trap).

The interrupts and exceptions supported in the V850 Family 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 handling. 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 Family handles the following four types of interrupts/exceptions:
  - Non-maskable interrupt
  - Maskable interrupt
  - Software exception
  - Exception trap

Table 6-1. List of Interrupt/Exception Codes

Interrupt/Exception Cause		Classification	Exception Code	Handler Address	Restore PC
Name	Trigger				
NMI	NMI input	Interrupt	0010H	00000010H	next PC[Note 2]
Maskable interrupt	**Note 1**	Interrupt	**Note 1**	**Note 1**	next PC[Note 2]
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 op code	Exception	0060H	00000060H	next PC[Note 3]

Notes 1. Differs depending on the type of the maskable interrupt.

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

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

The restore PC is the PC saved to the 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 INTC unit (interrupt controller). The INTC is different for each device in the V850 Family due to the variations of on-chip peripherals, interrupt/exception causes and exception codes.

6.1 Interrupt Processing

6.1.1 Maskable interrupts

A maskable interrupt can be masked by the program status word (PSW).

The INTC issues an interrupt request to the CPU, based on the acknowledged 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.

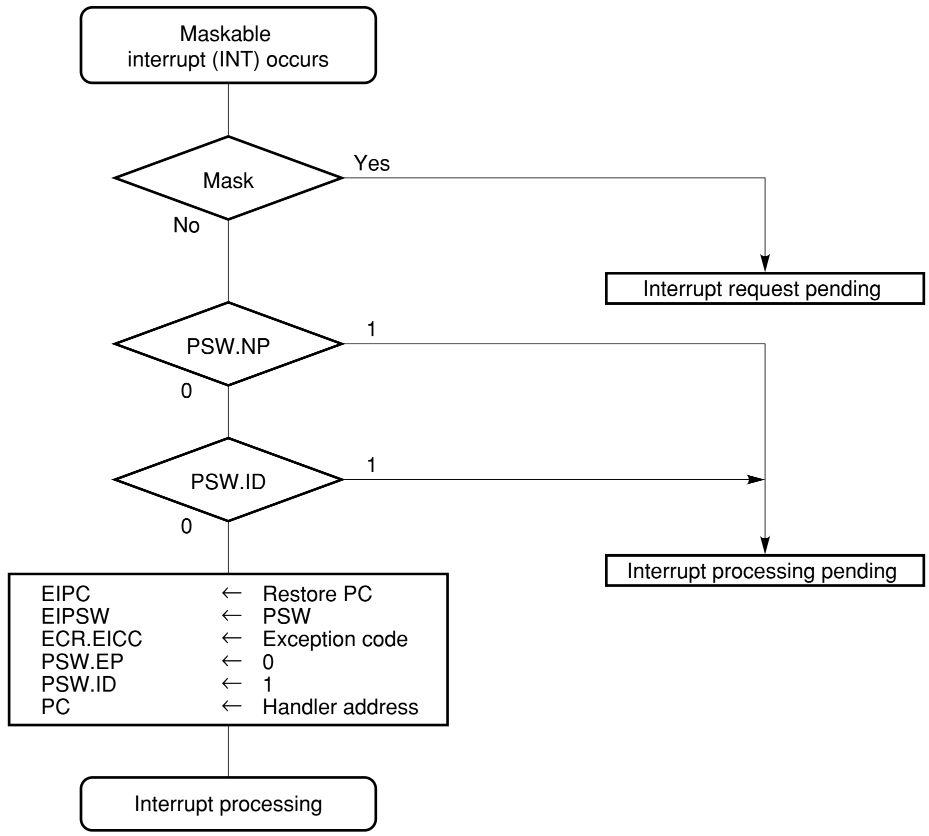
(1) Saves restore PC to EIPC.
(2) Saves current PSW to EIPSW.
(3) Writes exception code to lower half-word of ECR (EICC).
(4) Sets ID bit of PSW and clears EP bit.
(5) Sets handler address for each interrupt to PC and transfers control.

The EIPC and EIPSW are used as the status saving registers. 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 INTC, or when an interrupt processing routine is currently being executed (when the NP bit of the PSW is 1 or when the ID bit of the PSW is 1). New maskable interrupt processing is started by the pending INT input when the mask condition is cleared and the NP and ID bits of the PSW are reset to 0 by the LDSR and RETI instructions.

The EIPC and EIPSW must be saved by the program to enable nesting of interrupts because only one set of EIPC and EIPSW is provided. Bits 31 through 24 of the EIPC and bits 31 through 8 of the EIPSW are fixed to 0.

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

Figure 6-1. Maskable Interrupt Processing Format



6.1.2 Non-maskable interrupts

A non-maskable interrupt cannot be disabled by an instruction and therefore can be always acknowledged. The non-maskable interrupt of the V850 Family is generated by NMI input.

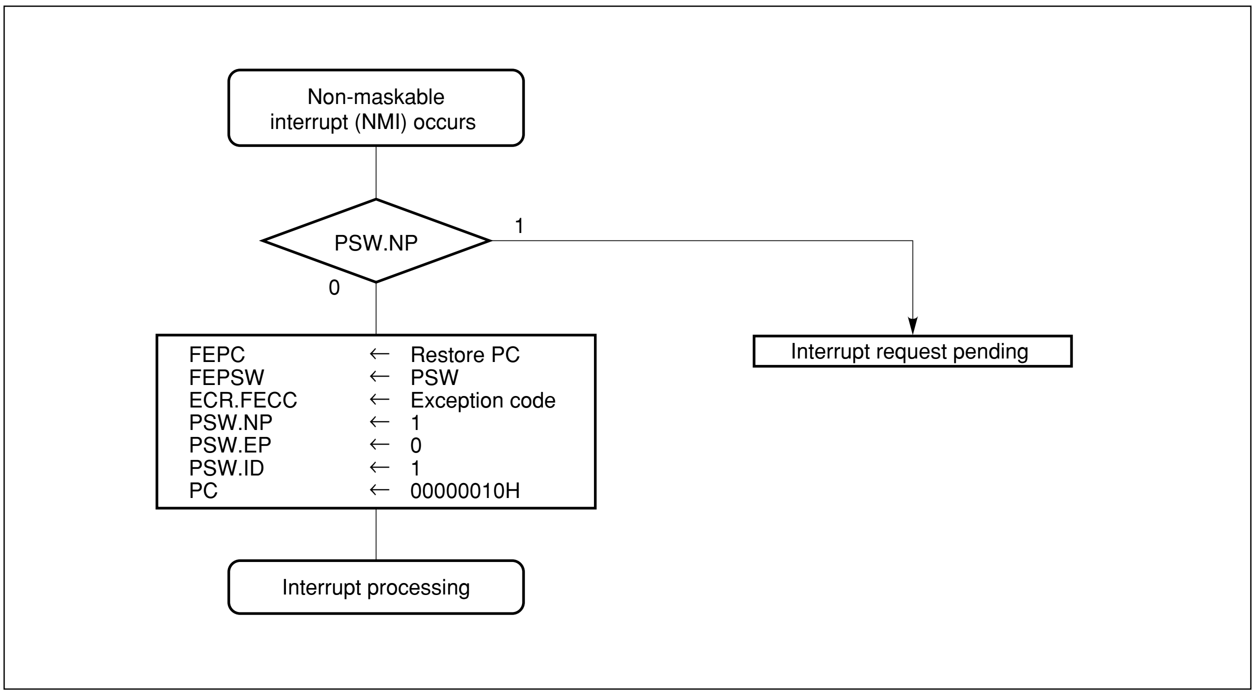
When a non-maskable interrupt is generated by NMI input, the processor performs the following steps, and transfers control to the handler routine.

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

The FEPC and FEPSW are used as the status saving registers. Non-maskable interrupts are held pending in the INTC when another non-maskable interrupt is currently being executed (when the NP bit of the PSW is 1). New non-maskable interrupt processing is started by the pending non-maskable interrupt request when the NP bit of the PSW is reset to 0 by the RETI and LDSR instructions.

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

Figure 6-2. Non-maskable Interrupt Processing Format



6.2 Exception Processing

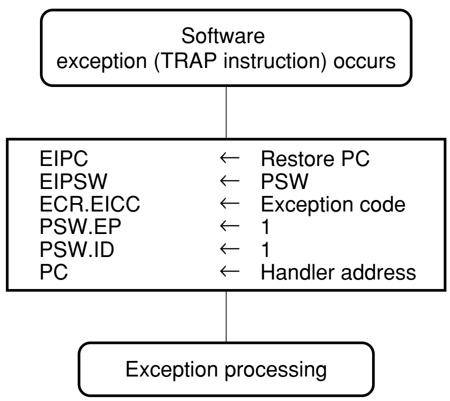
6.2.1 Software exceptions

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



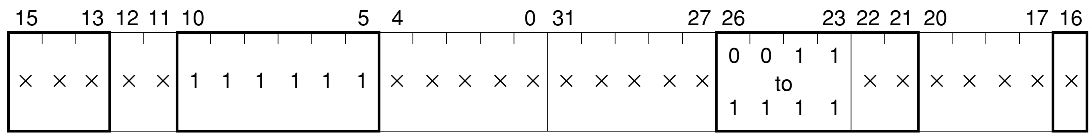
Handler address: 00000040H (vector = 0nH)
00000050H (vector = 1nH)

6.2.2 Exception trap

An exception trap is an interrupt requested when an instruction is illegally executed. The exception trap of the V850 Family is generated by an illegal op code instruction code trap (ILGOP: ILleGal OPcode trap).

An illegal op code instruction has an instruction code with an op code (bits 5 through 10) of 111111B and a sub-op code (bits 23 through 26) of 0011B through 1111B. When this kind of an illegal op code instruction is executed, an illegal op code instruction code trap occurs.

Figure 6-4. Illegal Instruction Code



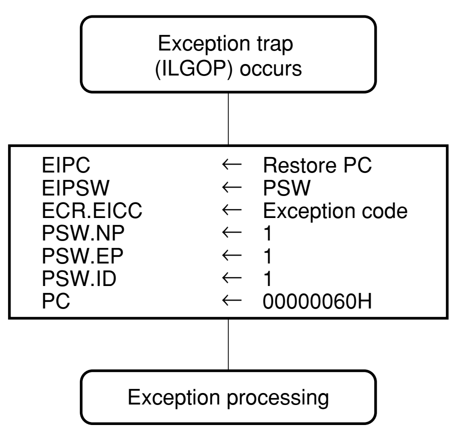
Remark × : don't care
☐: Op code/sub-op code position

If an exception trap 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.
★ (4) Sets NP, EP, and ID bits of PSW.
(5) 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 The operation is not guaranteed when an instruction that has not been defined either as an instruction or an illegal instruction is executed.

6.3 Restoring from Interrupt/Exception

All restoration from interrupt/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.

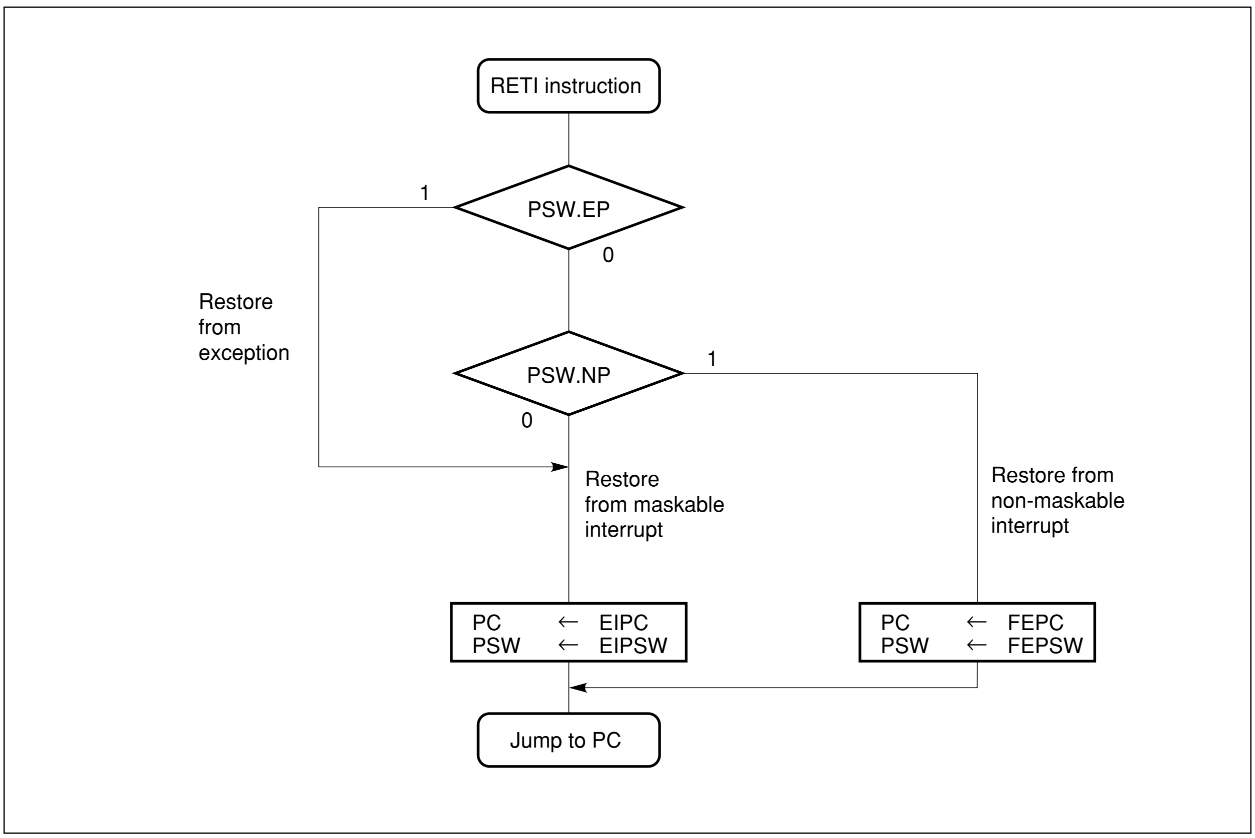
(1) If the EP bit of the PSW is 0 and the NP bit of the PSW is 1, the restore PC and PSW are read from the FEPC and FEPSW; otherwise, the restore PC and PSW are read from the EIPC and EIPSW.
(2) Control is transferred to the address of the restored PC and PSW.

When execution has returned from exception processing or non-maskable interrupt processing, 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 .... PSW's NP bit = 1, EP bit = 0
To restore from exception processing .... PSW's EP bit = 1

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

Figure 6-6. Restoring 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, re-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 register	EIPC EIPSW	Undefined Undefined
NMI status saving register	FEPC FEPSW	Undefined Undefined
Exception cause register (ECR)	FECC EICC	0000H 0000H
Program status word	PSW	00000020H
General register	r0 r1 to r31	Fixed to 00000000H Undefined

7.2 Start Up

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

CHAPTER 8 PIPELINE

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

The processor uses a 5-stage pipeline.

The operation to be performed in each stage is as follows:

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 instruction is executed.
MEM (memory access)	Memory at specified address is accessed.
WB (write back)	Result of execution is written to register.

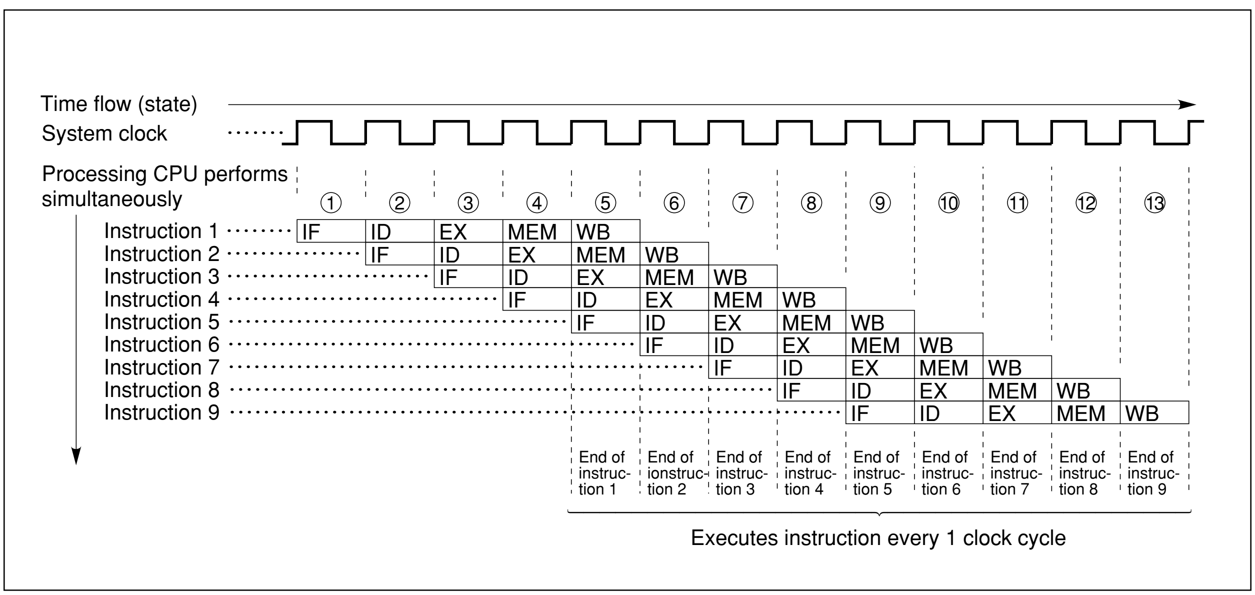
8.1 Outline of Operation

The instruction execution sequence of the V850 Family 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 8-1 shows the processing of the CPU when nine standard instructions are executed in succession.

Figure 8-1. 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 write back. Because five instructions can be processed at the same time, however, a standard instruction can be executed in 1 clock cycle on average.

8.2 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/PROM 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 a path wait time. Access times are shown in Figure 8-2 below.

Table 8-1. Access Times (in Clocks)

Stage \ Resource (Bus Width)	Internal ROM/PROM (32 Bits)	Internal RAM (32 Bits)	Internal Peripheral I/O (8/16 Bits)	External Memory (16 Bits)
Instruction fetch	1	3	Impossible	3 + n
Memory access (MEM)	3	1	3 + n	3 + n

Remark n: Wait number

8.2.1 Load instructions

[Instructions] LD, SLD

[Pipeline]

	①	②	③	④	⑤	⑥
Load instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[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 load instruction, data wait time occurs. For details, see **8.3 Pipeline Disorder**.

8.2.2 Store instructions

[Instructions] ST, SST

[Pipeline]

	①	②	③	④	⑤	⑥
Store instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[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.

8.2.3 Arithmetic operation instructions (excluding multiply and divide instructions)

[Instructions] MOV, MOVEA, MOVHI, ADD, ADDI, CMP, SUB, SUBR, SETF

[Pipeline]

	①	②	③	④	⑤	⑥
Arithmetic operation instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

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

8.2.4 Multiply instructions

[Instructions] MULH, MULHI

[Pipeline] (1) When the next instruction is not a multiply instruction

	①	②	③	④	⑤	⑥
Multiply instruction	IF	ID	EX1	EX2	WB	
Next instruction		IF	ID	EX	MEM	WB

(2) When the next instruction is a multiply instruction

	①	②	③	④	⑤	⑥
Multiply instruction 1	IF	ID	EX1	EX2	WB	
Multiply instruction 2		IF	ID	EX1	EX2	WB

[Description] The pipeline consists of 5 stages, IF, ID, EX1, EX2, and WB. There is no MEM stage. 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. For details, see **8.3 Pipeline Disorder**.

8.2.5 Divide instructions

[Instructions] DIVH

[Pipeline]

	①	②	③	④		㊲	㊳	㊴	㊵	㊶	㊷
Divide instruction	IF	ID	EX1	EX2		EX35	EX36	MEM	WB		
Next instruction		IF	–	–		–	ID	EX	MEM	WB	
Next to next instruction							IF	ID	EX	MEM	WB

– : Idle inserted for wait

[Description] The pipeline consists of 40 stages, IF, ID, EX1 to EX36, MEM, and WB. The EX stage requires 36 clocks. No operation is performed in the MEM stage, because memory is not accessed.

8.2.6 Logical operation instructions

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

[Pipeline]

	①	②	③	④	⑤	⑥
Logical operation instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. No operation is performed in the MEM stage, because memory is not accessed.

8.2.7 Saturation operation instructions

[Instructions] SATADD, SATSUB, SATSUBI, SATSUBR

[Pipeline]

	①	②	③	④	⑤	⑥
Saturation operation instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

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

8.2.8 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

	①	②	③	④	⑤	⑥
Conditional branch instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

(b) When the condition is realized

	①	②	③	④	⑤	⑥	⑦	⑧
Conditional branch instruction	IF	ID	EX	MEM	WB			
Next instruction		IF ×	ID ×					
Next to next instruction			IF ×					
Branch destination instruction				IF	ID	EX	MEM	WB

IF ×: Instruction fetch that is not executed

ID ×: Instruction decode 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.

(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 3. The IF stage of the next instruction and next to next instruction of the branch instruction is not executed.

(2) Unconditional branch instructions

[Instructions] JMP, JR, JARL, BR

[Pipeline]

	①	②	③	④	⑤	⑥	⑦	⑧
Unconditional branch instruction	IF	ID	EX	MEM	WB *			
Next instruction		IF ×						
Branch destination instruction				IF	ID	EX	MEM	WB

IF ×: Instruction fetch that is not executed

WB *: No operation is performed in the case of the JMP instruction, 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 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. 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.

8.2.9 Bit manipulation instructions

(1) SET1, CLR1, NOT1

[Pipeline]

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
SET1, CLR1, NOT1 instruction	IF	ID	EX1	MEM	EX2	EX3	MEM	WB		
Next instruction		IF	–	–	–	ID	EX	MEM	WB	
Next to next instruction						IF	ID	EX	MEM	WB

– : Idle inserted for wait

[Description] The pipeline consists of 8 stages, IF, ID, EX1, MEM, EX2, EX3, 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 3 and 2 clocks, respectively.

(2) TST1

[Pipeline]

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
TST1 instruction	IF	ID	EX1	MEM	EX2	EX3	MEM	WB		
Next instruction		IF	–	–	–	ID	EX	MEM	WB	
Next to next instruction						IF	ID	EX	MEM	WB

– : Idle inserted for wait

[Description] The pipeline consists of 8 stages, IF, ID, EX1, MEM, EX2, EX3, 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 stage require 3 and 2 clocks, respectively.

8.2.10 Special instructions

(1) LDSR, STSR

[Pipeline]

	①	②	③	④	⑤	⑥
LDSR, STSR instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the MEM stage, because memory is not accessed. Also, 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. For details, see **8.3 Pipeline Disorder**.

(2) NOP

[Pipeline]

	①	②	③	④	⑤	⑥
NOP instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[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]

	①	②	③	④	⑤	⑥
EI, DI instruction	IF	ID	EX	MEM	WB	
Next instruction		IF	ID	EX	MEM	WB

[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]

	①	②	③	④	⑤	⑥			HALT release				
HALT instruction	IF	ID	EX	MEM	WB								
Next instruction		IF	–	–	–	–		–	ID	EX	MEM	WB	
Next to next instruction									IF	ID	EX	MEM	WB

– : 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]

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
TRAP instruction	IF	ID1	ID2	EX	MEM	WB			
Next instruction		IF ×							
Jump destination instruction					IF	ID	EX	MEM	WB

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, MEM, and WB. However, no operation is performed in the MEM stage, because memory is not accessed. The ID stage requires 2 clocks. Also, the IF stage of the next instruction and next to next instruction is not executed.

(6) RETI

[Pipeline]

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
RETI instruction	IF	ID1	ID2	EX	MEM	WB			
Next instruction		IF ×							
Jump destination instruction					IF	ID	EX	MEM	WB

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 and next to next instruction is not executed.

8.3 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.

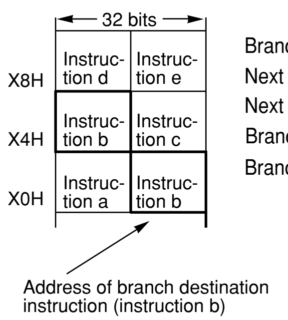
8.3.1 Alignment hazard

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.

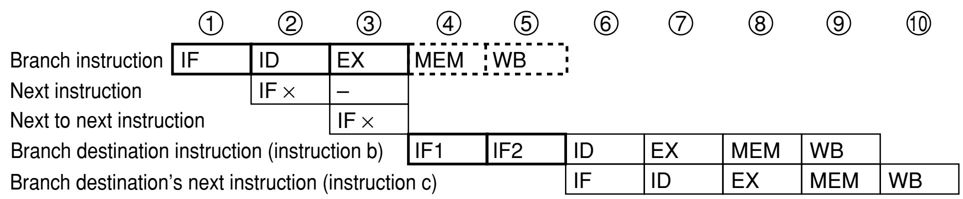
For example, the instructions a to e are placed from address X0H, and that instruction b consists of 4 bytes, and the other instructions each consist of 2 bytes. In this case, instruction b is placed at X2H (A1=1, A0=0), and is not word aligned (A1=0, A0=0). Therefore, when this instruction b becomes 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 8-2. Align Hazard Example

(a) Memory map



(b) Pipeline



IF ×: Instruction fetch that is not executed

–: Idle inserted for wait

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.

8.3.2 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 ended using that register. This is called a hazard. The V850 Family has an interlock function that causes the CPU to automatically handle this hazard by delaying the ID stage of the next instruction.

The V850 Family 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 8-3. Example of Execution Result of Load Instruction

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
Load instruction 1 (LD [R4], R6)	IF	ID	EX	MEM	WB				
Instruction 2 (ADD 2, R6)		IF	IL	ID	EX	MEM	WB		
Instruction 3			IF	–	ID	EX	MEM	WB	
Instruction 4					IF	ID	EX	MEM	WB

IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

↓: Short path

As described in Figure 8-3, 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.

8.3.3 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 Family's 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 8-4. Example of Execution Result of Multiply Instruction

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
Multiply instruction 1 (MULH 3, R6)	IF	ID	EX1	EX2	WB				
Instruction 2 (ADD 2, R6)		IF	IL	ID	EX	MEM	WB		
Instruction 3			IF	–	ID	EX	MEM	WB	
Instruction 4					IF	ID	EX	MEM	WB

IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

↓: Short path

As described in Figure 8-4, 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.

8.3.4 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 Family's 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 8-5. Example of Execution Result of LDSR Instruction for EIPC and FEPC

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
LDSR instruction (LDSR R6, 0) Note	IF	ID	EX	MEM	WB					
STSR instruction (STSR 0, R7) Note		IF	IL	IL	ID	EX	MEM	WB		
Next instruction			IF	–	–	ID	EX	MEM	WB	
Next to next instruction						IF	ID	EX	MEM	WB

IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

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

As described in Figure 8-5, 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.

8.3.5 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.

8.4 Additional Items Related to Pipeline

8.4.1 Harvard architecture

The V850 Family uses the Harvard architecture to operate an instruction fetch path from internal ROM and a memory access path to internal RAM independently. This eliminates path arbitration conflicts between the IF and MEM stages and allows orderly pipeline operation.

(1) V850 Family (Harvard architecture)

The MEM stage of instruction 1 and the IF stage of instruction 4, as well as the MEM stage of instruction 2 and the IF stage of instruction 5 can be executed simultaneously with orderly pipeline operation.

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
Instruction 1	IF	ID	EX	MEM	WB				
Instruction 2		IF	ID	EX	MEM	WB			
Instruction 3			IF	ID	EX	MEM	WB		
Instruction 4				IF	ID	EX	MEM	WB	
Instruction 5					IF	ID	EX	MEM	WB

(2) Not V850 Family (Other than Harvard architecture)

The MEM stage of instruction 1 and the IF stage of instruction 4, in addition to the MEM stage of instruction 2 and the IF stage of instruction 5 are in contention, causing path waiting to occur and slower execution time due to disorderly pipeline operation.

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
Instruction 1	IF	ID	EX	MEM	WB						
Instruction 2		IF	ID	–	EX	MEM	WB				
Instruction 3			IF	–	ID	–	EX	MEM	WB		
Instruction 4					IF	–	ID	EX	MEM	WB	
Instruction 5							IF	ID	EX	MEM	WB

–: Idle inserted for wait

8.4.2 Short path

The V850 Family provides on chip a short path that allows the use of the execution result of the preceding instruction by the following instruction before write back (WB) is completed for the previous instruction.

Example 1. Execution result of arithmetic operation instruction and logical operation used by instruction following immediately after

- V850 Family (on-chip short path)
  The execution result of the preceding instruction can be used for the ID stage of the instruction following immediately after as soon as the result is out (EX stage), without having to wait for write back to be completed.

	①	②	③	④	⑤	⑥
ADD 2, R6	IF	ID	EX	MEM	WB	
MOV R6, R7		IF	ID	EX	MEM	WB

- Not V850 Family (No short path)
  The ID stage of the instruction following immediately after is delayed until write back of the previous instruction is completed.

	①	②	③	④	⑤	⑥	⑦	⑧
ADD 2, R6	IF	ID	EX	MEM	WB			
MOV R6, R7		IF	–	–	ID	EX	MEM	WB

–: Idle inserted for wait

↓: Short path

Example 2. Data read from memory by the load instruction used by instruction following immediately after

- V850 Family (on-chip short path)
  The execution result of the preceding instruction can be used for the ID stage of the instruction following immediately after as soon as the result is out (MEM stage), without having to wait for write back to be completed.

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
LD [R4], R6	IF	ID	EX	MEM	WB				
ADD 2, R6		IF	IL	ID	EX	MEM	WB		
Next instruction			IF	–	ID	EX	MEM	WB	
Next to next instruction					IF	ID	EX	MEM	WB

- Not V850 Family (No short path)
  The ID stage of the instruction following immediately after is delayed until write back of the previous instruction is completed.

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
LD [R4], R6	IF	ID	EX	MEM	WB					
ADD 2, R6		IF	–	–	ID	EX	MEM	WB		
Next instruction					IF	ID	EX	MEM	WB	
Next to next instruction						IF	ID	EX	MEM	WB

IL: Idle inserted for data wait by interlock function

–: Idle inserted for wait

↓: Short path

APPENDIX A INSTRUCTION MNEMONIC (IN ALPHABETICAL ORDER)

This appendix summarizes the instruction mnemonic list described previously.
Instructions are listed in the alphabetical order of their mnemonics.
The illustration and table shown below indicates how to read this appendix and what each symbol and word means.

Instruction Mnemonic	Operand	Format	CY OV S Z SAT

Convention

ADD	reg1, reg2	I	* * * * –
Instruction mnemonic	Operand name	Indicates instruction format	Describes movements of flags

Name	Meaning
reg1	General register (used as source register)
reg2	General register (mainly used as destination register. Some are also used as source registers)
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

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

Table A-1. Instruction Mnemonic (In Alphabetical Order) (1/7)

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. 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	–	–	–	–	–	Conditional branch (if Carry). Tests a condition flag specified by an instruction. Branches if a 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.
CLR1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Bit clear. Adds the data of reg1 to 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.
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.
DI	–	X	–	–	–	–	–	Disables maskable interrupt. Sets the ID flag of the PSW to 1 to disable the acknowledgement of maskable interrupts from acceptance; interrupts are immediately disabled at the start of this instruction execution.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (2/7)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
DIVH	reg1, reg2	I	–	*	*	*	–	Signed divide. Divides the word data of reg2 by the lower half-word data of reg1, and stores the quotient to reg2.
EI	–	X	–	–	–	–	–	Enables maskable interrupt. Resets the ID flag of the PSW to 0 and enables the acknowledge-ment of maskable interrupts at the beginning of next instruction.
HALT	–	X	–	–	–	–	–	CPU halt. Stops the operating clock of the CPU and places the CPU in the HALT mode.
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 to 0.
JMP	[reg1]	I	–	–	–	–	–	Register indirect unconditional branch. Transfers control to the address specified by reg1. Bit 0 of the address is masked to 0.
JR	disp22	V	–	–	–	–	–	Unconditional branch. Adds a 22-bit displace-ment, 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 to 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	–	–	–	–	–	Half-word load. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Half-word data is read from this 32-bit address with its bit 0 masked to 0, sign-extended to word length, and stored in 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 to 0, and stored in reg2.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (3/7)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
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.
MOV	reg1, reg2	I	–	–	–	–	–	Moves data. Transfers the word data of reg1 to reg2.
MOV	imm5, reg2	II	–	–	–	–	–	Moves data. Transfers the value of a 5-bit immediate data, sign-extended to word length, to reg2.
MOVEA	imm16, reg1, reg2	VI	–	–	–	–	–	Moves 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	–	–	–	–	–	Moves higher half-word. 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.
MULH	reg1, reg2	I	–	–	–	–	–	Signed multiply. Multiplies the lower half-word data of reg2 by the lower half-word data of reg1, and stores the result in reg2 as word data.
MULH	imm5, reg2	II	–	–	–	–	–	Signed multiply. Multiplies the lower half-word data of reg2 by a 5-bit immediate data, sign-extended to half-word length, and stores the result in reg2 as word data.
MULHI	imm16, reg1, reg2	VI	–	–	–	–	–	Signed multiply. Multiplies the lower half-word data of reg1 by a 16-bit immediate data, and stores the result in reg2.
NOP	–	I	–	–	–	–	–	No operation.
NOT	reg1, reg2	I	–	0	*	*	–	Logical 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	–	–	–	*	–	Bit not. First, adds the data of reg1 to a 16-bit displacement, sign-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.
OR	reg1, reg2	I	–	0	*	*	–	Logical sum. ORs the word data of reg2 with the word data of reg1, and stores the result in reg2.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (4/7)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
ORI	imm16, reg1, reg2	VI	–	0	*	*	–	Logical sum. ORs the word data of reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result in reg2.
RETI	–	X	*	*	*	*	*	Returns from exception or interrupt routine. Restores the restore PC and PSW from the appropriate system register, and restores from exception or interrupt routine.
SAR	reg1, reg2	IX	*	0	*	*	–	Arithmetic right shift. 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	*	*	–	Arithmetic right shift. Arithmetically shifts the word data of reg2 to the right by ‘n’ positions specified by the 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.
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 general register reg2. However, if the result exceeds the positive maximum 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.
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.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (5/7)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
SATSUBI	imm16, reg1, reg2	VI	*	*	*	*	*	Saturated subtract. Subtracts a 16-bit immediate 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 a condition specified by condition code “cccc” is satisfied; otherwise, a 0 is stored in the register.
SET1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Bit set. 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 bit field “bbb” is set at the byte data location specified by the generated address.
SHL	reg1, reg2	IX	*	0	*	*	–	Logical left shift. 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 writes the result to reg2.
SHL	imm5, reg2	II	*	0	*	*	–	Logical left shift. 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 writes the result to reg2.
SHR	reg1, reg2	IX	*	0	*	*	–	Logical right shift. 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 writes the result to reg2.
SHR	imm5, reg2	II	*	0	*	*	–	Logical right shift. 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 writes the result to reg2.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (6/7)

Instruction Mnemonic	Operand	Format	CY	OV	S	Z	SAT	Instruction Function
SLD.B	disp7 [ep], reg2	IV	–	–	–	–	–	Byte load. Adds the 7-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and stored in reg2.
SLD.H	disp8 [ep], reg2	IV	–	–	–	–	–	Half-word load. Adds the 8-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Half-word data is read from this 32-bit address with bit 0 masked to 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 to 0, 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 at the generated address.
SST.H	reg2, disp8 [ep]	IV	–	–	–	–	–	Half-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 lower half-word of reg2 at the generated 32-bit address with bit 0 masked to 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 at the generated 32-bit address with bits 0 and 1 masked to 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 at the generated address.
ST.H	reg2, disp16 [reg1]	VII	–	–	–	–	–	Half-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 lower half-word of reg2 at the generated 32-bit address with bit 0 masked to 0.

Table A-1. Instruction Mnemonic (In Alphabetical Order) (7/7)

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 at the generated 32-bit address with bits 0 and 1 masked to 0.
STSR	regID, reg2	IX	–	–	–	–	–	Stores contents of system register. Stores the contents of a 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.
TRAP	vector	X	–	–	–	–	–	Software trap. Saves the restore PC and PSW to the system register; sets the exception code and the flags of the PSW; jumps to the address of the trap handler corresponding to the trap vector specified by vector number, 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.
TST1	bit#3, disp16 [reg1]	VIII	–	–	–	*	–	Bit test. Adds the data of 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.
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.

APPENDIX B INSTRUCTION LIST

Table B-1. Mnemonic List

Mnemonic	Function	Mnemonic	Function
	Load/store		(3-operand)
LD.B	Load Byte	MOVHI	Move High Halfword
LD.H	Load Halfword	MOVEA	Move Effective Address
LD.W	Lord Word	ADDI	Add Immediate
SLD.B	Load Byte	MULHI	Multiply Halfword Immediate
SLD.H	Load Halfword	SATSUBI	Saturated Subtract Immediate
SLD.W	Load Word	ORI	Or Immediate
ST.B	Store Byte	ANDI	And Immediate
ST.H	Store Halfword	XORI	Exclusive Or Immediate
ST.W	Store Word		Branch
SST.B	Store Byte		
SST.H	Store Halfword	JMP	Jump Register
SST.W	Store Word	JR	Jump Relative
	Integer arithmetic operation/logical operation/saturated operation (2-operand register)	JARL	Jump and Register Link
		Bcond	Branch on condition code
			Bit manipulation
MOV	Move	SET1	Set Bit
ADD	Add	CLR1	Clear Bit
SUB	Subtract	NOT1	Not Bit
SUBR	Subtract Reverse	TST1	Test Bit
MULH	Multiply Halfword		Special
DIVH	Divide Halfword		
CMP	Compare	LDSR	Load System Register
SATADD	Saturated Add	STSR	Store System Register
SATSUB	Saturated Subtract	TRAP	Trap
SATSUBR	Saturated Subtract Reverse	RETI	Return from Trap or Interrupt
TST	Test	HALT	Halt
OR	Or	DI	Disable Interrupt
AND	And	EI	Enable Interrupt
XOR	Exclusive Or	NOP	No operation
NOT	Not		
SHL	Shift Logical Left		
SHR	Shift Logical Right		
SAR	Shift Arithmetic Right		
	(2-operand immediate)		
MOV	Move		
ADD	Add		
CMP	Compare		
SATADD	Saturated Add		
SETF	Set Flag Condition		
SHL	Shift Logical Left		
SHR	Shift Logical Right		
SAR	Shift Arithmetic Right		

Table B-2. Instruction Set

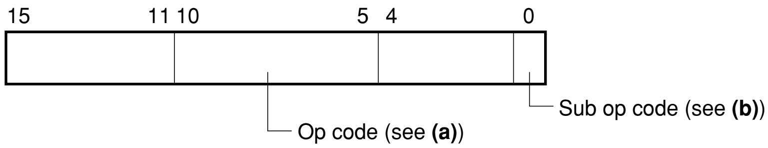
Instruction Code b10 • • • • b5	Instruction Format		Format	Remarks
0 0 0 0 1 0	MOV	reg1, reg2	I	When reg1, reg2 = 0, NOP
0 0 0 0 0 1	NOT	reg1, reg2		
0 0 0 0 1 0	DIVH	reg1, reg2		
0 0 0 0 1 1	JMP	[reg1]		
0 0 0 1 0 0	SATSUBR	reg1, reg2		
0 0 0 1 0 1	SATSUB	reg1, reg2		
0 0 0 1 1 0	SATADD	reg1, reg2		
0 0 0 1 1 1	MULH	reg1, reg2		
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 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 1 1 0 × ×	SLD.B	disp7 [ep], reg2	IV	
0 1 1 1 × ×	SST.B	reg2, disp7 [ep]		
1 0 0 0 × ×	SLD.H	disp8, [ep], reg2		
1 0 0 1 × ×	SST.H	reg2, disp8 [ep]		
1 0 1 0 × ×	SLD.W	disp8 [ep], reg2		
1 0 1 0 × ×	SST.W	reg2, disp8 [ep]		
1 0 1 1 × ×	Bcond	disp9	III	
1 1 0 0 0 0	ADDI	imm16, reg1, reg2	VI	
1 1 0 0 0 1	MOVEA	imm16, reg1, reg2		
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	imm16, reg1, reg2		
1 1 0 1 1 1	MULHI	imm16, reg1, reg2		
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 ×	JARL	disp22, reg2	V	When reg2 = r0, 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	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	Undefined instruction			

APPENDIX C INSTRUCTION OP CODE MAP

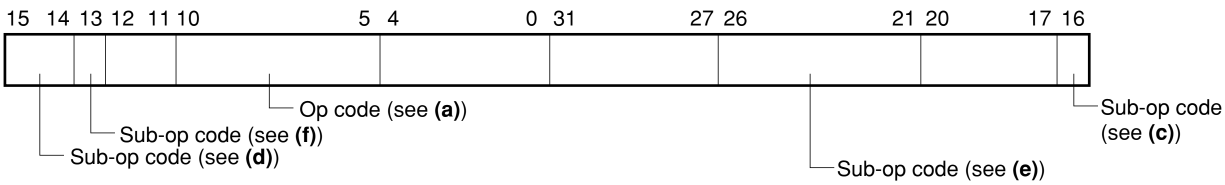
The following tables (a) through (f) show the op code maps corresponding to instruction codes.

Instruction code

- **16-bit length instruction format**



- **32-bit length instruction format**



(a) Op code

Bits 6 to 5 / Bits 10 to 7	00	01	10	11	Format
0000	MOV/NOP	NOT	DIVH	JMP	I
0001	SATSUBR	SATSUB	SATADD	MULH	
0010	OR	XOR	AND	TST	
0011	SUBR	SUB	ADD R, r	CMP R,r	
0100	MOV imm5, r	SATADD	ADD imm5, r	CMP imm5, r	II
0101	SHR imm5, r	SAR imm5, r	SHL imm5, r	MULH	
0110	SLD.B				IV
0111	SST.B				
1000	SLD.H				
1001	SST.H				
1010	SLD.W/SST.W[Note 1]				
1011	Bcond				III
1100	ADDI	MOVEA	MOVHI	SATSUBI	VI
1101	ORI	XORI	ANDI	MULHI	
1110	LD.B	LD.H/LD.W[Note 2]	ST.B	ST.H/ST.W[Note 2]	V/VII/VIII/IX/X
1111	JARL		Bit manipulation[Note 3]	Extension 1[Note 4]	

Notes 1. See **(b)**.
2. See **(c)**.
3. See **(d)**.
4. See **(e)**.

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

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-op code)

Bits 6 to 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 (sub-op code)

Bit 15 \ Bit 14	0	1
0	SET1	NOT1
1	CLR1	TST1

(e) Extension 1 (sub-op code)

Bits 26 to 23 \ Bits 22 to 21	00	01	10	11
0000	SETF	LDSR	STSR	Undefined
0001	SHR R, r	SAR R, r	SHL R, r	Undefined
0010	TRAP	HALT	RETI	Extension 2[Note]
0011 to 1111	Illegal instruction			

Note See **(f)**.

(f) Extension 2 (sub-op code)

Bit 15 \ Bits 14 to 13	00	01	10	11
0	DI	Undefined		
1	EI			

APPENDIX D INDEX

[V]

[W]

[X]

[Z]

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: ____________________

Document number: ____________________ Page number: __________

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 01.2