NEC

User's Manual

V850E1

32-bit Microprocessor Core

Architecture

Document No. U14559EJ2V0UM00 (2nd 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 Revision in This Edition (1/2)

Page	Description
Throughout	• Addition of following products (under development) to target products NB85ET, NU85E, NU85ET, μPD703108, 703114, 70F3114, 703116 • Deletion of following product from target products μPD703117 • Change of following products from “under development” to “developed” μPD703106, 703107, 70F3107
p.22	Change of **Note** in **Figure 2-1 Registers**
p.24	Change of **Table 2-2 System Register Numbers**
p.27	Addition of **Note** to **Figure 2-6 Program Status Word (PSW)**
p.29	Addition of **Note** to **Section 2.2.6 Exception/debug trap status saving registers (DBPC, DBPSW)**
p.30	Change of **Caution** in **Section 2.2.8 Debug interface register (DIR)**
p.32	Change of **Caution** in **Section 2.2.9 Breakpoint control registers 0 and 1 (BPC0, BPC1)**
p.32	Change of **Figure 2-11 Breakpoint Control Registers 0 and 1 (BPC0, BPC1)**
p.33	Change of **Caution** in **Section 2.2.10 Program ID register (ASID)**
p.34	Change of **Caution** in **Section 2.2.11 Breakpoint address setting registers 0 and 1 (BPAV0, BPAV1)**
p.34	Change of **Caution** in **Section 2.2.12 Breakpoint address mask registers 0 and 1 (BPAM0, BPAM1)**
p.35	Change of **Caution** in **Section 2.2.13 Breakpoint data setting registers 0 and 1 (BPDV0, BPDV1)**
p.35	Change of **Caution** in **Section 2.2.14 Breakpoint data mask registers 0 and 1 (BPDM0, BPDM1)**
p.53	Addition of **Caution** to **Section 5.2 (10) Debug function instructions**
p.69	Addition of **Caution** to **DBRET** in **Section 5.3 Instruction Set**
p.70	Addition of **Caution** to **DBTRAP** in **Section 5.3 Instruction Set**
p.161	Change of and adding **Note** to **Table 5-6 List of Number of Instruction Execution Clock Cycles (NB85E, NB85ET, NU85E, and NU85ET)**
p.164	Change of **Note** to **Table 5-7 List of Number of Instruction Execution Clock Cycles (V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2)**
p.166	Addition of **Note** to **Table 6-1 Interrupt/Exception Codes**
p.172	Addition of **Caution** to **Section 6.2.3 Debug trap**
p.180	Addition of **Remark** and **Example** to **Section 8.1.2 2-clock branch**
p.181	Addition of **Caution** to **Section 8.1.3 Efficient pipeline processing**
p.182	Correction of description in **Section 8.2 (2) V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2**
p.182	Correction of description in **Section 8.2.1 (2) SLD instructions**
p.183	Correction of description in **Section 8.2.3 Multiply instructions**
p.184	Addition of **Remark** to **Section 8.2.4 (3) Divide instructions**
p.187	Correction of description in **Section 8.2.8 (2) TST1 instruction**
p.188	Addition of **Remark** to **Section 8.2.9 (3) DI, EI instructions**
p.190	Addition of **Caution** to **Section 8.2.9 (7) NOP instruction**
p.193	Addition of **Section 8.3 Pipeline Disorder**
p.197	Addition of **Section 8.4 Additional Items Related to Pipeline**
p.201	Addition of **Note** to **Table A-1 Instruction Function List (in Alphabetical Order)**
pp.211, 213	Addition of **Note** to **Table A-2 Instruction List (in Format Order)**

Major Revision in This Edition (2/2)

Page	Description
p.214	Correction of **Figure** in **Appendix B (2) 32-bit format instruction**
p.215	Addition of **Remark** to **Appendix B [a] Opcode**
p.217	Addition of **Remark** to **Appendix B [e] Expansion 1 (sub-opcode)**
pp.219, 220	Addition of **Note** to **Appendix C DIFFERENCES WITH ARCHITECTURE OF V850 CPU**
p.222	Addition of **Note** to **Table D-1 Instructions Added to V850E1 CPU and V850 CPU Instructions with Same Instruction Code**

The mark ★ shows major revised points.

PREFACE

Target Readers

This manual is intended for users who wish to understand the functions of the V850E1 CPU core for designing application systems using the V850E1 CPU core.

- Products incorporating V850E1 CPU core
  - NB85E[Note], NB85ET[Note], NU85E[Note], NU85ET[Note]
  - V850E/MA1™ : μPD703103[Note], 703105[Note], 703106, 703107, 70F3107
  - V850E/MA2™ : μPD703108[Note]
  - V850E/IA1™ : μPD703116[Note], 70F3116[Note]
  - V850E/IA2™ : μPD703114[Note], 70F3114[Note]

Note Under development

Purpose

This manual is intended for users to understand the architecture of the V850E1 CPU core described in the Organization below.

Organization

This manual contains the following information:

- Register set
- Data type
- Instruction format and instruction set
- Interrupt and exception
- Pipeline

How to Use this Manual

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

To learn about the hardware functions,
→ Read **User's Manual Hardware** of each product.

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

Conventions

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

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

K (Kilo): $2^{10} = 1,024$
M (Mega): $2^{20} = 1,024^2$
G (Giga): $2^{30} = 1,024^3$

Related Documents

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

• Device related documents

Document Name	Document No.
NB85E User's Manual Hardware	A13971E
NB85ET User's Manual Hardware	A14342E
NU85E User's Manual Hardware	A14874E
NU85ET User's Manual Hardware	A15015E
V850E/MA1 User's Manual Hardware	U14359E
V850E/MA2 User's Manual Hardware	U14980E
V850E/IA1 User's Manual Hardware	U14492E
V850E/IA2 User's Manual Hardware	U15195E

• Development tool related documents

Document Name		Document No.
IE-V850E-MC, IE-V850E-MC-A (In-Circuit Emulator)		U14487E
IE-V850E-MC-EM1-A (Peripheral I/O Board for V850E1)		To be prepared
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2 (Peripheral I/O Board for V850E1)		U14482E
IE-703107-MC-EM1 (Peripheral I/O Board for V850E/MA1, V850E/MA2)		U14481E
IE-703116-MC-EM1(Peripheral I/O Board for V850E/IA1)		To be prepared
IE-703114-MC-EM1(Peripheral I/O Board for V850E/IA2)		To be prepared
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 (1/2)

LIST OF FIGURES (2/2)

LIST OF TABLES

CHAPTER 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), and
- factory automation equipment such as NC (Numerical Control) machine tools and various controllers.

The great majority of these systems conventionally employ 8-bit or 16-bit microcontrollers. However, the performance level of these microcontrollers has become inadequate in recent years as control operations have risen in complexity, leading to the development of increasingly complicated instruction sets and hardware 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 of 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 saturate, bit manipulate, and multiply/divide (executed by a hardware multiplier) instructions, which are especially suited for digital servo control systems. Moreover, instruction formats are designed for maximum compiler coding efficiency, allowing the reduction of object code sizes.

The V850E1 CPU is a 32-bit RISC CPU core for ASIC, newly developed as the CPU core central to system LSI in the current age of system-on-a-chip. This core includes not only the control functions of the V850 CPU, the CPU core incorporated in the V850 Family, but also supports data processing through its enhanced external bus interface performance, and the addition of features such as C language switch statement processing, table lookup branching, stack frame creation/deletion, data conversion, and other high-level language supporting instructions.

In addition, because the instruction codes are upwardly compatible with the V850 CPU at the object code level, the software resources of systems that incorporate the V850 CPU can be used unchanged.

1.1 Features

(1) High-performance 32-bit architecture for embedded control
- Number of instructions: 83
- 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: 64 MB linear
  Data space: 4 GB linear

(2) Special instructions
- Saturation operation instructions
- Bit manipulation instructions
- Multiply instructions (On-chip hardware multiplier executing multiplication in 1, 2, or 4 clocks)
  16 bits × 16 bits → 32 bits
  32 bits × 32 bits → 32 bits or 64 bits

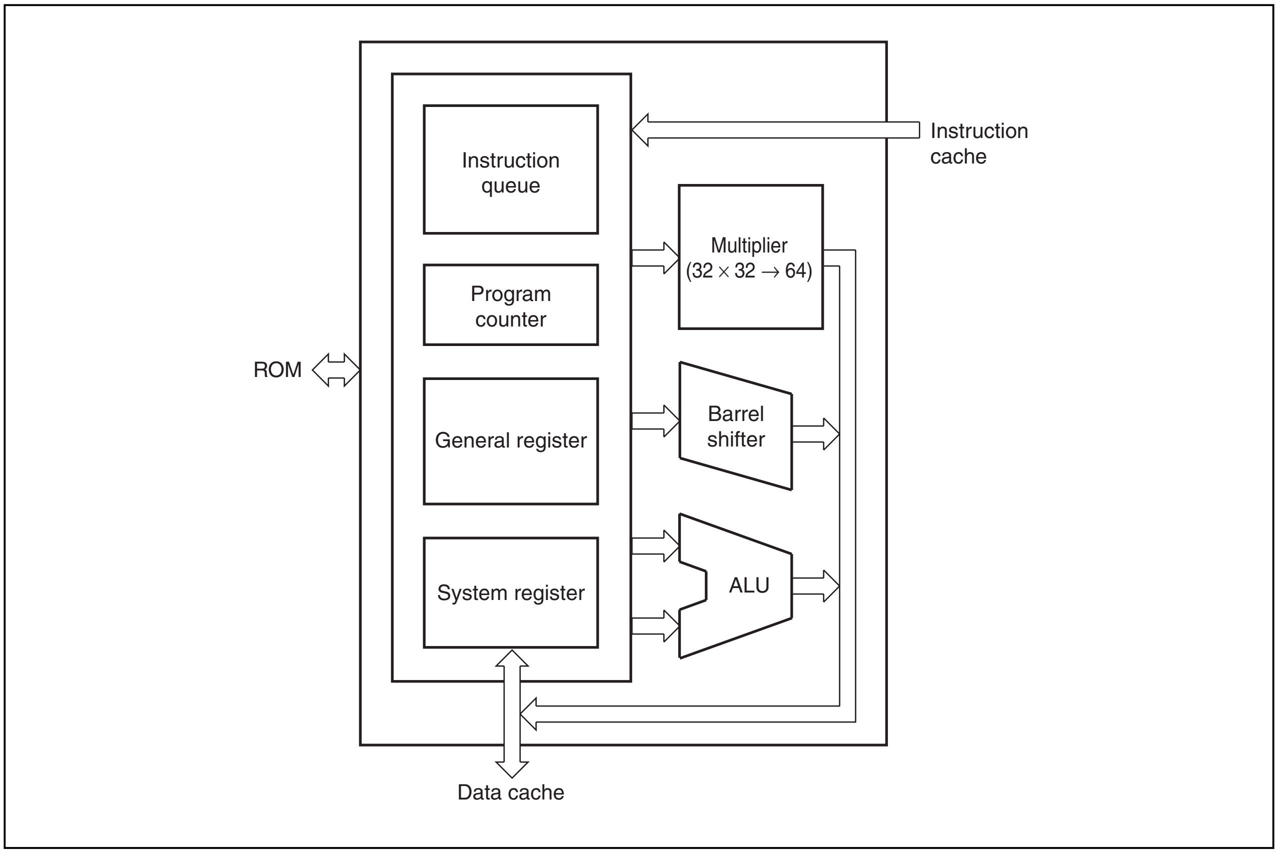
1.2 Internal Configuration

The V850E1 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.

It contains dedicated hardware such as a multiplier (32 × 32 bits) and a barrel shifter (32 bits/clock) to execute complicated instructions at high speeds.

Figure 1-1 shows the internal block diagram.

Figure 1-1. Internal Block Diagram of V850E1 CPU



CHAPTER 2 REGISTER SET

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

Figure 2-1. Registers

(a) Program registers

31 0
r0 (Zero register)
r1 (Assembler-reserved register)
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)

(b) System registers

31 0	
EIPC (Interrupt status saving register)	
EIPSW (Interrupt status saving register)	
FEPC (NMI status saving register)	
FEPSW (NMI status saving register)	
ECR (Exception cause register)	
PSW (Program status word)	
CTPC (CALLT caller status saving register)	
CTPSW (CALLT caller status saving register)	
DBPC (Exception/debug trap status saving register)	
DBPSW (Exception/debug trap status saving register)	
CTBP (CALLT base pointer)	
DIR (Debug interface register)	Note
BPC0 (Breakpoint control register 0)	Note
BPC1 (Breakpoint control register 1)	Note
ASID (Program ID register)	Note
BPAV0 (Breakpoint address setting register 0)	Note
BPAV1 (Breakpoint address setting register 1)	Note
BPAM0 (Breakpoint address mask register 0)	Note
BPAM1 (Breakpoint address mask register 1)	Note
BPDV0 (Breakpoint data setting register 0)	Note
BPDV1 (Breakpoint data setting register 1)	Note
BPDM0 (Breakpoint data mask register 0)	Note
BPDM1 (Breakpoint data mask register 1)	Note

Note These registers are reserved for debug function. Only the NU85E and NU85ET can be used. The NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2 cannot be used.

2.1 Program Registers

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

Table 2-1. Program Registers

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

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

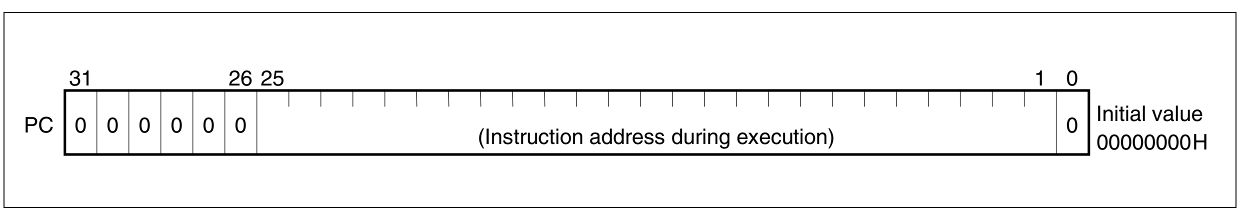
(1) General registers (r0 to r31)

Thirty-two general registers, r0 to r31, are provided. All these registers can be used for data variable or address variable. 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 is sometimes used by the real-time OS. When the real-time OS to be used is not using r2, r2 can be used as a variable register.

(2) Program counter (PC)

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

Figure 2-2. Program Counter (PC)



2.2 System Registers

The system registers control the status and holds information on interrupts.

System registers can be read or written by specifying the relevant system register number from the following list using the LDSR and STSR instructions.

Table 2-2. System Register Numbers

Register No.	Register Name	Operand Specifiability	
		LDSR Instruction	STSR Instruction
0	Interrupt status saving register (EIPC)	○	○
1	Interrupt status saving register (EIPSW)	○	○
2	NMI status saving register (FEPC)	○	○
3	NMI status saving register (FEPSW)	○	○
4	Exception cause register (ECR)	×	○
5	Program status word (PSW)	○	○
6 to 15	(Numbers reserved for future function expansion (operation cannot be guaranteed if accessed))	×	×
16	CALLT caller status saving register (CTPC)	○	○
17	CALLT caller status saving register (CTPSW)	○	○
18	Exception/debug trap status saving register (DBPC)	○Note 1	○
19	Exception/debug trap status saving register (DBPSW)	○Note 1	○
20	CALLT base pointer (CTBP)	○	○
★ 21	Debug interface register (DIR)	○Note 1	○Note 1
22	Breakpoint control registers 0 and 1 (BPC0, BPC1)Note 2	○Note 1	○Note 1
23	Program ID register (ASID)	○	○
24	Breakpoint address setting registers 0 and 1 (BPAV0, BPAV1)Note 2	○Note 1	○Note 1
25	Breakpoint address mask registers 0 and 1 (BPAM0, BPAM1)Note 2	○Note 1	○Note 1
26	Breakpoint data setting registers 0 and 1 (BPDV0, BPDV1)Note 2	○Note 1	○Note 1
27	Breakpoint data mask registers 0 and 1 (BPDM0, BPDM1)Note 2	○Note 1	○Note 1
28 to 31	(Numbers reserved for future function expansion (operation cannot be guaranteed if accessed))	×	×

★ **Notes 1.** These registers can be accessed only in debug mode of the NU85E and NU85ET. Accessing to these registers of the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2 is prohibited. If they are accessed, the operation is not guaranteed.

2. The actual register to be accessed is specified by the CS bit of the DIR register.

Caution When returning using the RETI instruction after setting bit 0 of EIPC, FEPC, or CTPC to 1 using the LDSR instruction, the value of bit 0 is ignored (because bit 0 of the PC is fixed to 0). Therefore, be sure to set an even number (bit 0 = 0) when setting a value in EIPC, FEPC, or CTPC.

Remark O: Accessible
×: Inaccessible

2.2.1 Interrupt status saving registers (EIPC, EIPSW)

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

If a software exception or maskable interrupt occurs, the contents of the program counter (PC) are saved to EIPC, and the contents of the program status word (PSW) are saved to EIPSW (if a non-maskable interrupt (NMI) occurs, the contents are saved to NMI status saving registers (FEPC, FEPSW)).

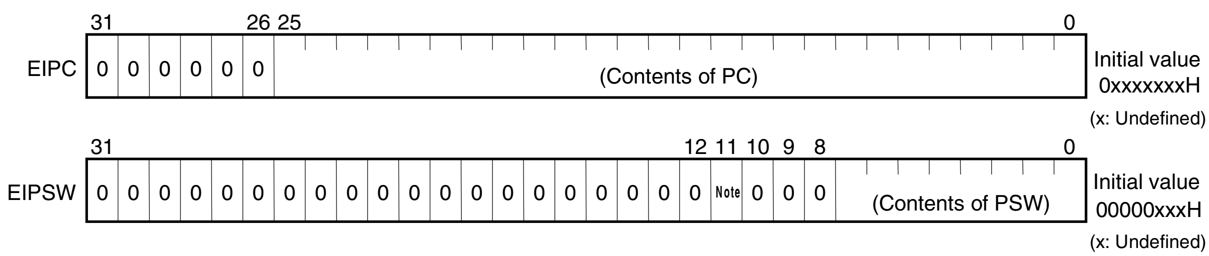
Except for part of instructions, the address of the instruction next to the one executed when the software exception or maskable interrupt has occurred is saved to the EIPC (see **Table 6-1 Interrupt/Exception Codes**).

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 31 to 26 of the EIPC and bits 31 to 12, 10 to 8 of the EIPSW are reserved for future function expansion (fixed to 0).

Figure 2-3. Interrupt Status Saving Registers (EIPC, EIPSW)



Note Contents of SS flag in PSW

2.2.2 NMI status saving registers (FEPC, FEPSW)

Two NMI status saving registers are provided: FEPC and FEPSW.

If a non-maskable interrupt (NMI) occurs, the contents of the program counter (PC) are saved to FEPC, and the contents of the program status word (PSW) are saved to FEPSW.

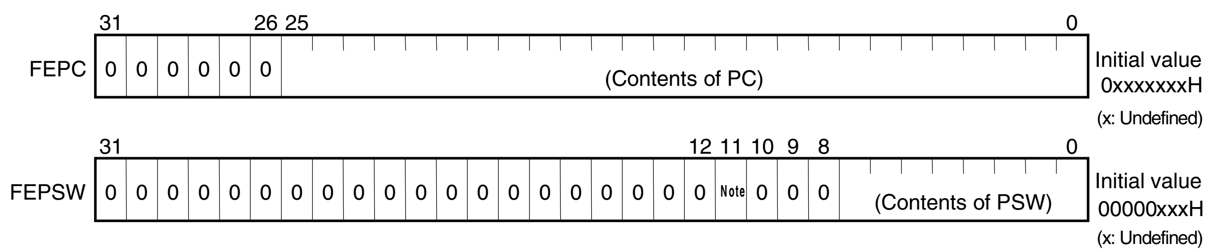
Except for part of instructions, the address of the instruction next to the one executed when the NMI has occurred is saved to the FEPC (see **Table 6-1 Interrupt/Exception Codes**).

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

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

Bits 31 to 26 of the FEPC and bits 31 to 12, 10 to 8 of the FEPSW are reserved for future function expansion (fixed to 0).

Figure 2-4. NMI Status Saving Registers (FEPC, FEPSW)

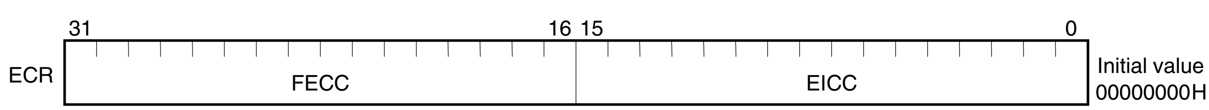


Note Contents of SS flag in PSW

2.2.3 Exception cause register (ECR)

The exception cause register (ECR) holds the cause information when an exception or interrupt occurs. The ECR holds an exception code which identifies each interrupt source (see **Table 6-1 Interrupt/Exception Codes**). This is a read-only register, and therefore, no data can be written to it by using the LDSR instruction.

Figure 2-5. Exception Cause Register (ECR)



Bit Position	Bit Name	Function
31 to 16	FECC	Exception code of non-maskable interrupt (NMI)
15 to 0	EICC	Exception code of exception or maskable interrupt

2.2.4 Program status word (PSW)

The program status word (PSW) 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 bits in this 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, interrupt requests are already disabled even while the LDSR instruction is executing.

Bits 31 to 12, 10 to 8 are reserved for future function expansion (fixed to 0).

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

	31																			12	11	10	9	8	7	6	5	4	3	2	1	0	
PSW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SS	0	0	0	NP	EP	ID	SAT	CY	OV	S	Z	Initial value 00000020H

Bit Position	Flag Name	Function
11	SS[Note 1]	Operates with single-step execution when this flag is set to 1 (debug trap occurs each time instruction is executed). This flag is cleared to 0 when branching to the interrupt servicing routine. When the SE bit of DIR register is 0, this flag is not set (fixed to 0).
7	NP	Indicates that non-maskable interrupt (NMI) processing is in progress. This flag is set to 1 when NMI request is acknowledged, and multiple interrupts are disabled. 0: NMI processing is not in progress 1: NMI processing is in progress
6	EP	Indicates that exception processing is in progress. This flag is set to 1 when an exception occurs. Even when this bit is set, interrupt requests can be acknowledged. 0: Exception processing is not in progress 1: Exception processing is in progress
5	ID	Indicates whether maskable interrupt request can be acknowledged. 0: Interrupt can be acknowledged 1: Interrupt cannot be acknowledged
4	SAT[Note 2]	Indicates that an overflow has occurred in a saturate operation and the result is saturated. This is a cumulative flag. When the result is saturated, the flag is set to 1 and is not cleared to 0 even if the next result does not saturate. To clear this flag to 0, use the LDSR instruction. This flag is neither set to 1 nor cleared to 0 by execution of arithmetic operation instruction. 0: Not saturated 1: Saturated
3	CY	Indicates whether carry or borrow occurred as a result of the operation. 0: Carry or borrow did not occur 1: Carry or borrow occurred
2	OV[Note 2]	Indicates whether overflow occurred as a result of the operation. 0: Overflow did not occur 1: Overflow occurred

Remark See the following page for an explanation of the Note 2.

★ **Note 1.** Only the NU85E and NU85ET can be used. The NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2 cannot be used.

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

	31																			12	11	10	9	8	7	6	5	4	3	2	1	0	
PSW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SS	0	0	0	NP	EP	ID	SAT	CY	OV	S	Z	Initial value 00000020H

Bit Position	Flag Name	Function
1	S[Note 2]	Indicates whether the result of the operation is negative. 0: Result is positive or zero 1: Result is negative
0	Z	Indicates whether the result of the operation is zero. 0: Result is not zero 1: Result is zero

Note 2. In the case of saturate instructions, the SAT, S, and OV flags will be set according to the result of the operation as shown in the table below. Note that the SAT flag is set to 1 only when the OV flag has been set to 1 during saturate operation.

Status of Operation Result	Status of Flag			Operation Result of Saturation Processing
	SAT	OV	S	
Maximum positive value is exceeded	1	1	0	7FFFFFFFH
Maximum negative value is exceeded	1	1	1	80000000H
Positive (Not exceeding maximum value)	Holds the value before operation	0	0	Operation result
Negative (Not exceeding maximum value)			1	

2.2.5 CALLT caller status saving registers (CTPC, CTPSW)

Two CALLT caller status saving registers are provided: CTPC and CTPSW.

If a CALLT instruction is executed, the contents of the program counter (PC) are saved to CTPC, and the contents of the program status word (PSW) are saved to CTPSW.

The contents saved to the CTPC are the address of the instruction next to the CALLT instruction.

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

Bits 31 to 26 of the CTPC and bits 31 to 12, 10 to 8 of the CTPSW are reserved for future function expansion (fixed to 0).

Figure 2-7. CALLT Caller Status Saving Registers (CTPC, CTPSW)

	31	30	29	28	27	26	25 … 0	
CTPC	0	0	0	0	0	0	(Contents of PC)	Initial value 0xxxxxxxH (x: Undefined)

	31 … 12	11	10	9	8	7 … 0	
CTPSW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Note	0	0	0	(Contents of PSW)	Initial value 00000xxxH (x: Undefined)

Note Contents of SS flag in PSW

2.2.6 Exception/debug trap status saving registers (DBPC, DBPSW)

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

If an exception trap or debug trap[Note] occurs, the contents of the program counter (PC) are saved to DBPC, and the contents of the program status word (PSW) are saved to DBPSW.

The contents saved to the DBPC are the address of the instruction next to the one executed when the exception trap or debug trap has occurred.

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

Bits 31 to 26 of the DBPC and bits 31 to 12, 10 to 8 of the DBPSW are reserved for future function expansion (fixed to 0).

★ **Note** The NB85E and NB85ET do not support debug trap.

Figure 2-8. Exception/Debug Trap Status Saving Registers (DBPC, DBPSW)

	31	30	29	28	27	26	25 … 0	
DBPC	0	0	0	0	0	0	(Contents of PC)	Initial value 0xxxxxxxH (x: Undefined)

	31 … 12	11	10	9	8	7 … 0	
DBPSW	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Note	0	0	0	(Contents of PSW)	Initial value 00000xxxH (x: Undefined)

Note Contents of SS flag in PSW

2.2.7 CALLT base pointer (CTBP)

The CALLT base pointer (CTBP) is used to specify a table address and to generate a target address (bit 0 is fixed to 0).

Bits 31 to 26 are reserved for future function expansion (fixed to 0).

Figure 2-9. CALLT Base Pointer (CTBP)

	31					26	25 to 1	0	
CTBP	0	0	0	0	0	0	(Base address)	0	Initial value 0xxxxxxxH

(x: Undefined)

2.2.8 Debug interface register (DIR)

The debug interface register (DIR) indicates the control and status of the debug function.

The values of the bits in this register can be changed by using the LDSR instruction. Changed values become valid immediately after execution of this instruction.

This register can only be set in the debug mode (DM bit = 1).

Bits 14 to 8, 6 to 4, 2, and 1 are undefined in the user mode (DM bit = 0).

Bits 31 to 15 and 7 are reserved for future function expansion (fixed to 0).

★ **Caution Use of the debug interface register (DIR) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-10. Debug Interface Register (DIR) (1/2)

	31 to 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DIR	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SQ	RE	CS	CE	MA	AE	SE	0	IN	T1	T0	CM	MT	AT	DM	Initial value 00000040H

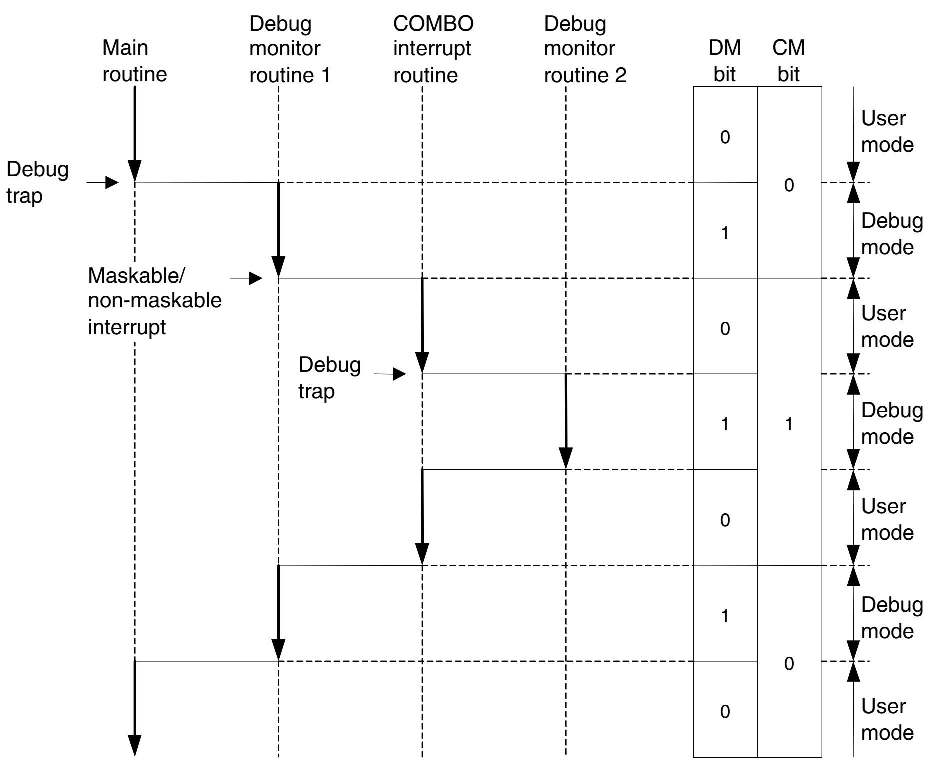
Bit Position	Bit Name	Function
14	SQ	Sets sequential break mode. 0: Normal break mode 1: Sequential break mode
13	RE	Sets range break mode. 0: Normal break mode 1: Range break mode
12	CS	Sets break register bank. 0: Bank 0 register (channel 0 control register) selected 1: Bank 1 register (channel 1 control register) selected
11	CE	Enables/disables COMBO interrupt. 0: COMBO interrupt disabled 1: COMBO interrupt enabled
10	MA	Enables/disables misalign access exception detection. 0: Misalign access exception detection disabled 1: Misalign access exception detection enabled

Figure 2-10. Debug Interface Register (DIR) (2/2)

Bit Position	Bit Name	Function
9	AE	Enables/disables alignment error exception detection. 0: Alignment error exception detection disabled 1: Alignment error exception detection enabled
8	SE	Enables/disables writing to SS flag of PSW. 0: Writing to SS flag disabled (SS flag is fixed to 0) 1: Writing to SS flag enabled
6	IN[Note 1]	Set to 1 by break register reset.
5	T1[Note 1]	Set to 1 by channel 1 break generation.
4	T0[Note 1]	Set to 1 by channel 0 break generation.
3	CM[Note 2]	Set to 1 by shift to COMBO interrupt routine or debug monitor routine 2.
2	MT[Note 1]	Set to 1 by detection of misalign access exception.
1	AT[Note 1]	Set to 1 by detection of alignment error exception.
0	DM[Note 2]	Set to 1 by shift to debug mode.

Notes 1. Can only be cleared to 0 by LDSR instruction.

2. The DM and CM bits change as shown below.



2.2.9 Breakpoint control registers 0 and 1 (BPC0, BPC1)

Breakpoint control registers 0 and 1 (BPC0, BPC1) indicate the control and status of the debug function.

One or other of these registers is enabled by the setting of the CS bit of DIR register.

The values of the bits in these registers can be changed by using the LDSR instruction. Changed values become valid immediately after execution of this instruction.

These registers can only be set in the debug mode (DM bit = 1). In the user mode, bit 0 = 0, and bits 23 to 15, 11 to 7, and 4 to 1 are undefined.

Bits 31 to 24, 14 to 12, 6, and 5 are reserved for future function expansion (fixed to 0).

★ **Caution Use of breakpoint control registers 0 and 1 (BPC0, BPC1) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-11. Breakpoint Control Registers 0 and 1 (BPC0, BPC1) (1/2)

	31	30	29	28	27	26	25	24	23 to 16	15	14	13	12	11, 10	9	8	7	6	5	4	3	2	1	0	
★ BPC0	0	0	0	0	0	0	0	0	BP ASID	IE	0	0	0	TY	VD	VA	MD	0	0	TE	BE	FE	WE	RE	Initial value 00xxxxx0H (x: Undefined)
★ BPC1	0	0	0	0	0	0	0	0	BP ASID	IE	0	0	0	TY	VD	VA	MD	0	0	TE	BE	FE	WE	RE	Initial value 00xxxxx0H (x: Undefined)

Bit Position	Bit Name	Function
23 to 16	BP ASID	Sets the program ID that generates a break (valid only when IE bit = 1).
15	IE	Sets the comparison of the BP ASID bit and the program ID set in the ASID register. 0: Not compared 1: Compared
11, 10	TY	Sets the type of access for which a break is detected. 0,0: Access by all data types 0,1: Byte access (including bit manipulation) 1,0: Half-word access 1,1: Word access Note that the contents set in this register are ignored in the case of an instruction fetch.
9	VD	Sets the match condition of the data comparator. 0: Break on a match 1: Break on a mismatch
8	VA	Sets the match condition of the address comparator. 0: Break on a match 1: Break on a mismatch
★ 7	MD	Sets the operation of the data comparator. If this bit is set to 1, the latency of access break is delay by 1 clock because even an data access break occurs before execution. 0: Break on match of data and condition. 1: Whether data coincides (data comparator) is ignored regardless of setting of VD bit or BPDVx and BPDMx registers

Figure 2-11. Breakpoint Control Registers 0 and 1 (BPC0, BPC1) (2/2)

Bit Position	Bit Name	Function
4	TE[Note]	Enables/disables trigger output. 0: Trigger output disabled 1: Trigger output enabled (outputs corresponding trigger before break occurs in channel 0 or 1).
3	BE[Note]	Sets whether or not a break in channel 0 or 1 is reported to CPU. 0: Not reported. 1: Reported (break).
2	FE	Enables/disables break on instruction fetch. 0: Break disabled 1: Break enabled
1	WE	Enables/disables break on data write. 0: Break disabled 1: Break enabled
0	RE	Enables/disables break on data read. 0: Break disabled 1: Break enabled

★ **Note** Use of TE and BE bits is possible only for the NU85ET.

2.2.10 Program ID register (ASID)

This register sets the ID of the program currently under execution.

Bits 31 to 8 are reserved for future function expansion (fixed to 0).

★ **Caution Use of the program ID register (ASID) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-12. Program ID Register (ASID)

	31																							8	7 to 0	
ASID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ASID	Initial value 000000xxH (x: Undefined)

Bit Position	Flag Name	Function
7 to 0	ASID	ID of program currently under execution

2.2.11 Breakpoint address setting registers 0 and 1 (BPAV0, BPAV1)

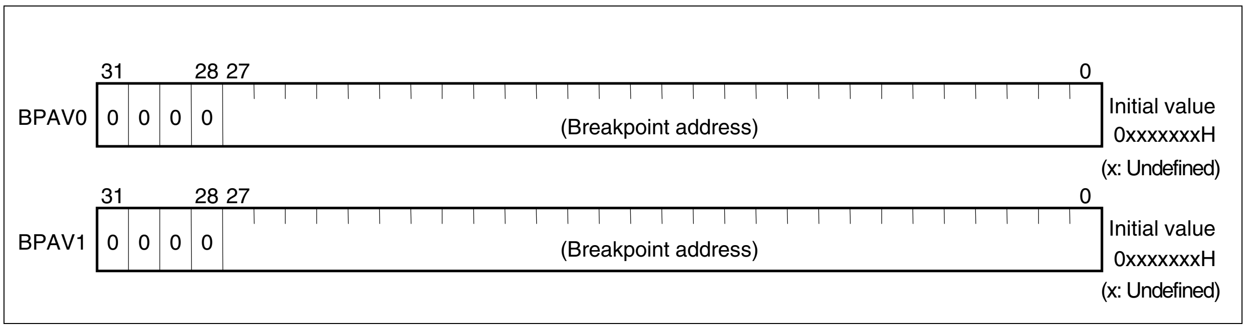
These registers set the breakpoint addresses to be used by the address comparator.

One or other of these registers is enabled by the setting of the CS bit of DIR register.

Bits 31 to 28 are reserved for future function expansion (fixed to 0).

★ **Caution Use of breakpoint address setting registers 0 and 1 (BPAV0, BPAV1) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-13. Breakpoint Address Setting Registers 0 and 1 (BPAV0, BPAV1)



2.2.12 Breakpoint address mask registers 0 and 1 (BPAM0, BPAM1)

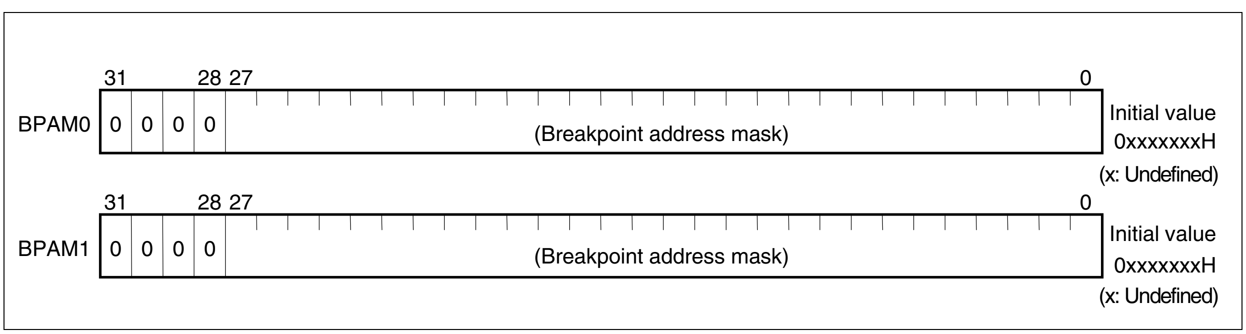
These registers set the bit mask for address comparison (masked by 1).

One or other of these registers is enabled by the setting of the CS bit of DIR register.

Bits 31 to 28 are reserved for future function expansion (fixed to 0).

★ **Caution Use of breakpoint address mask registers 0 and 1 (BPAM0, BPAM1) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-14. Breakpoint Address Mask Registers 0 and 1 (BPAM0, BPAM1)



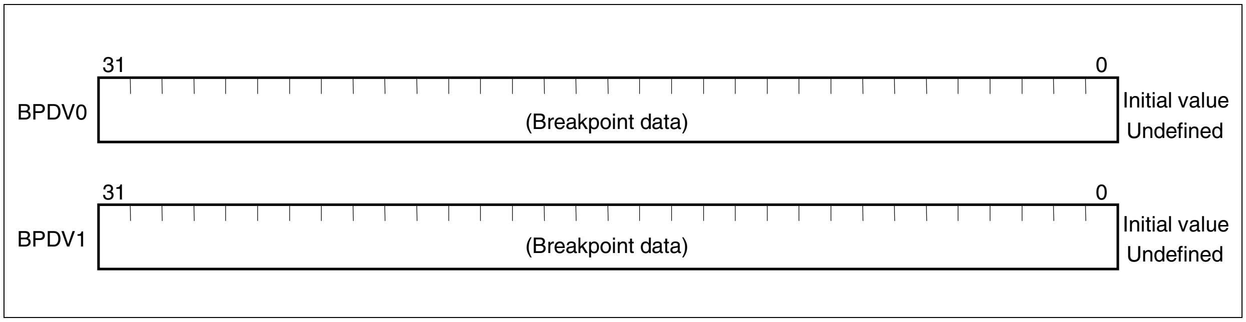
2.2.13 Breakpoint data setting registers 0 and 1 (BPDV0, BPDV1)

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

One or other of these registers is enabled by the setting of the CS bit of DIR register.

★ **Caution Use of breakpoint data setting registers 0 and 1 (BPDV0, BPDV1) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-15. Breakpoint Data Setting Registers 0 and 1 (BPDV0, BPDV1)



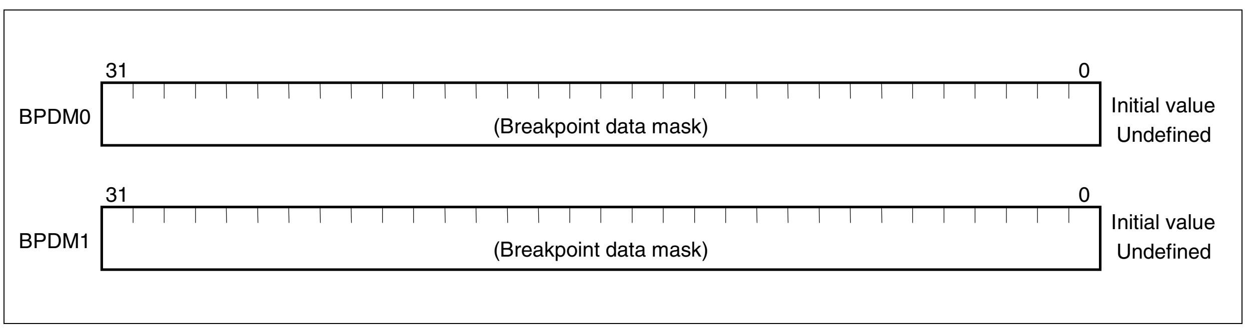
2.2.14 Breakpoint data mask registers 0 and 1 (BPDM0, BPDM1)

These registers set the bit mask for data comparison (masked by 1).

One or other of these registers is enabled by the setting of the CS bit of DIR register.

★ **Caution Use of breakpoint data mask registers 0 and 1 (BPDM0, BPDM1) is possible only for the NU85E or NU85ET, not for the NB85E, NB85ET, V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.**

Figure 2-16. Breakpoint Data Mask Registers 0 and 1 (BPDM0, BPDM1)



CHAPTER 3 DATA TYPE

3.1 Data Format

The following data types are supported (see **3.2 Data Representation**).

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

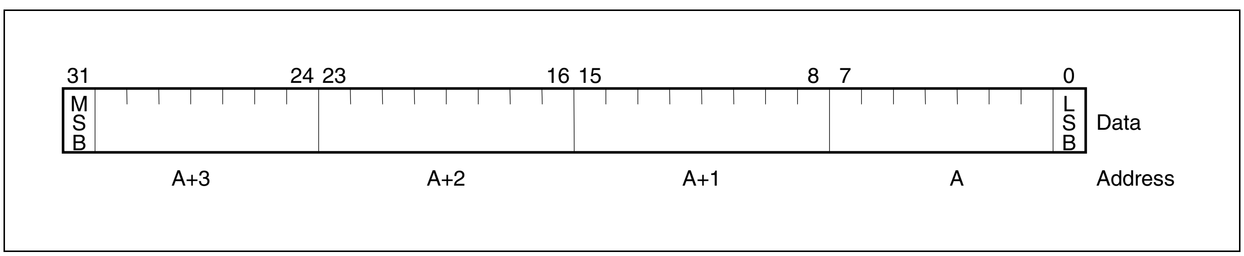
Three types of data lengths: word (32 bits), half-word (16 bits), and byte (8 bits) are supported. Byte 0 of any data is always the least significant byte (this is called little endian) and shown at the rightmost position in figures throughout this manual.

The following paragraphs describe the data format where data of fixed length is in memory.

(1) 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 (Least Significant Bit) is bit 0 and the MSB (Most Significant Bit) is bit 31. A word is specified by its address A (with the 2 lowest bits fixed to 0 when misalign access is disabled[Note]), and occupies 4 bytes, A, A+1, A+2, and A+3.

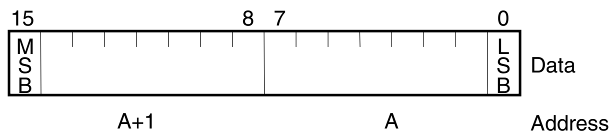
Note When misalign access is enabled, any byte boundary can be accessed whether access is in half-word or word units. See **3.3 Data Alignment**.



(2) 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[Note]), and occupies 2 bytes, A and A+1.

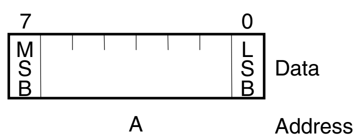
Note When misalign access is enabled, any byte boundary can be accessed whether access is in half-word or word units. See **3.3 Data Alignment**.



(3) 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 is bit 0 and the MSB is bit 7. A byte is specified by its address A.

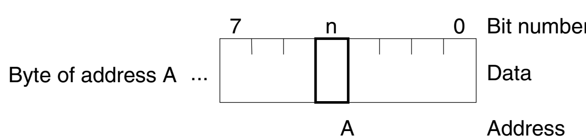
Note When misalign access is enabled, any byte boundary can be accessed whether access is in half-word or word units. See **3.3 Data Alignment**.



(4) 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 When misalign access is enabled, any byte boundary can be accessed whether access is in half-word or word units. See **3.3 Data Alignment**.



3.2 Data Representation

3.2.1 Integer

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

The integer range of each data length is as follows.

- Word (32 bits): –2,147,483,648 to +2,147,483,647
- Half-word (16 bits): –32,768 to +32,767
- Byte (8 bits): –128 to +127

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 32, 16, or 8 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.

The unsigned integer range of each data length is as follows.

- Word (32 bits): 0 to 4,294,967,295
- Half-word (16 bits): 0 to 65,535
- Byte (8 bits): 0 to 255

3.2.3 Bit

1-bit data that can take a value of 0 (cleared) or 1 (set) can be handled as a bit data. Bit manipulation can be performed only to 1-byte data in the memory space in the following four ways:

- Set
- Clear
- Invert
- Test

3.3 Data Alignment

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

When misalign access is enabled, it is possible to place word or half-word data at any address, but if data is not aligned at a boundary, one or more bus cycles is generated, which lowers the bus efficiency.

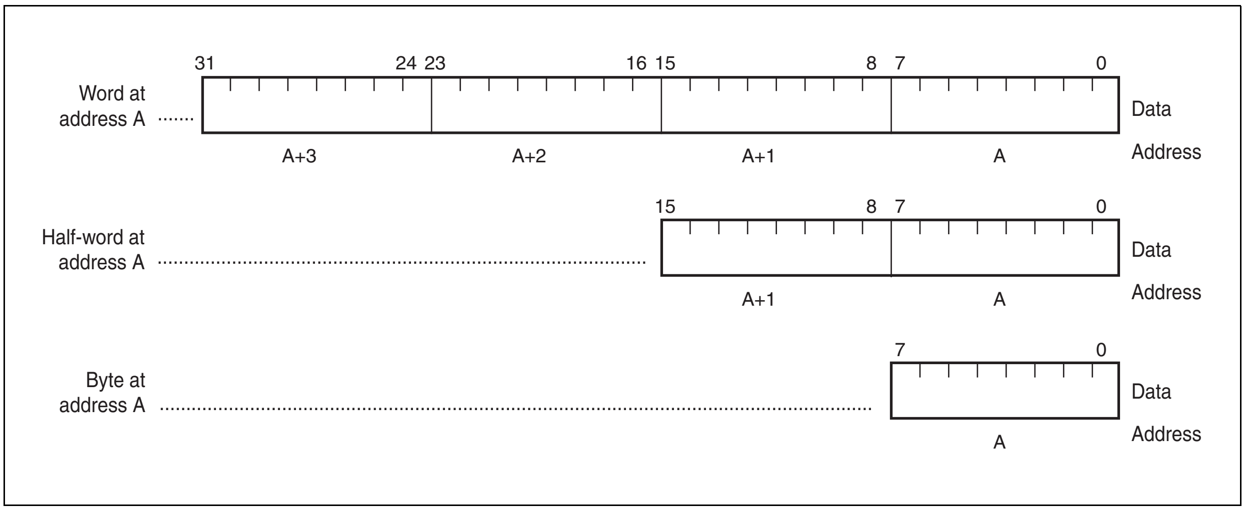
CHAPTER 4 ADDRESS SPACE

The V850E1 CPU supports a 4 GB linear address space. Both memory and I/O are mapped to this address space (memory-mapped I/O). The V850E1 CPU (NB85E) outputs 32-bit addresses to the memory and I/O. The maximum address is $2^{32}-1$.

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 shown below, with the lower address shown on the right and the higher address on the left.



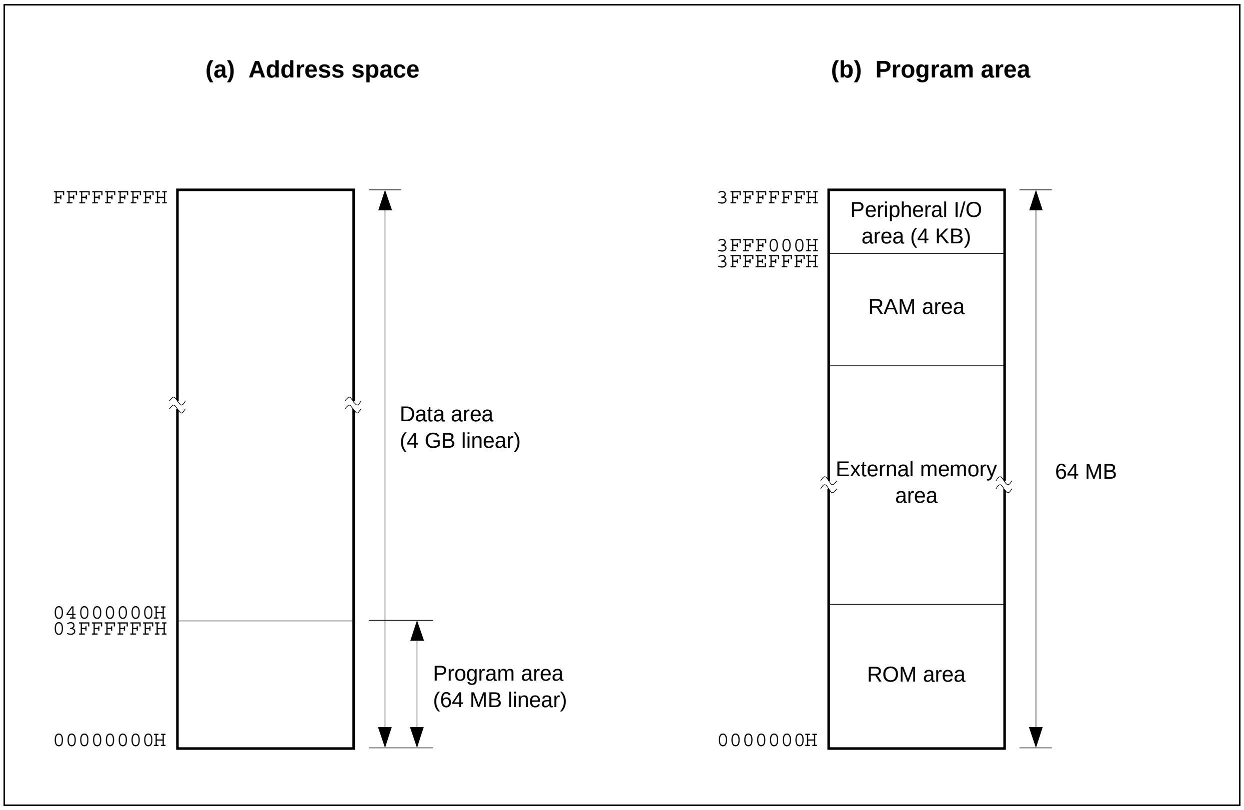
4.1 Memory Map

The V850E1 CPU employs a 32-bit architecture and supports a linear address space (data area) of up to 4 GB for operand addressing (data access).

It supports a linear address space (program area) of up to 64 MB for instruction addressing.

Figure 4-1 shows the memory map.

Figure 4-1. Memory Map



4.2 Addressing Mode

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

4.2.1 Instruction address

An instruction address is determined by the contents of the program counter (PC), and is automatically incremented (+2) according to the number of bytes of an instruction to be fetched each time an instruction has been executed. When a branch instruction is executed, the branch destination address is loaded into the PC using one of the following two addressing modes:

(1) Relative addressing (PC relative)

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

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

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

(a) JARL disp22, reg2 instruction, JR disp22 instruction

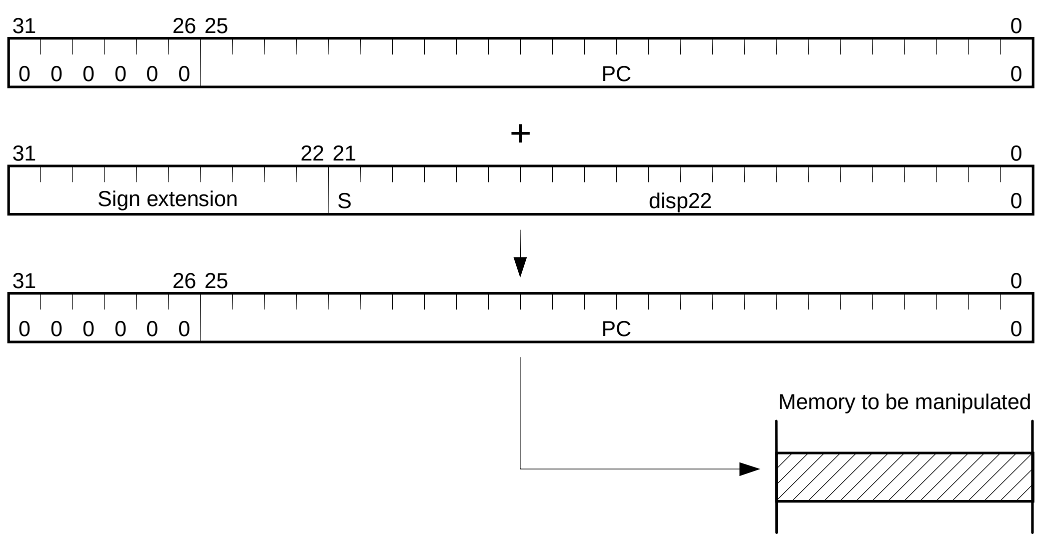
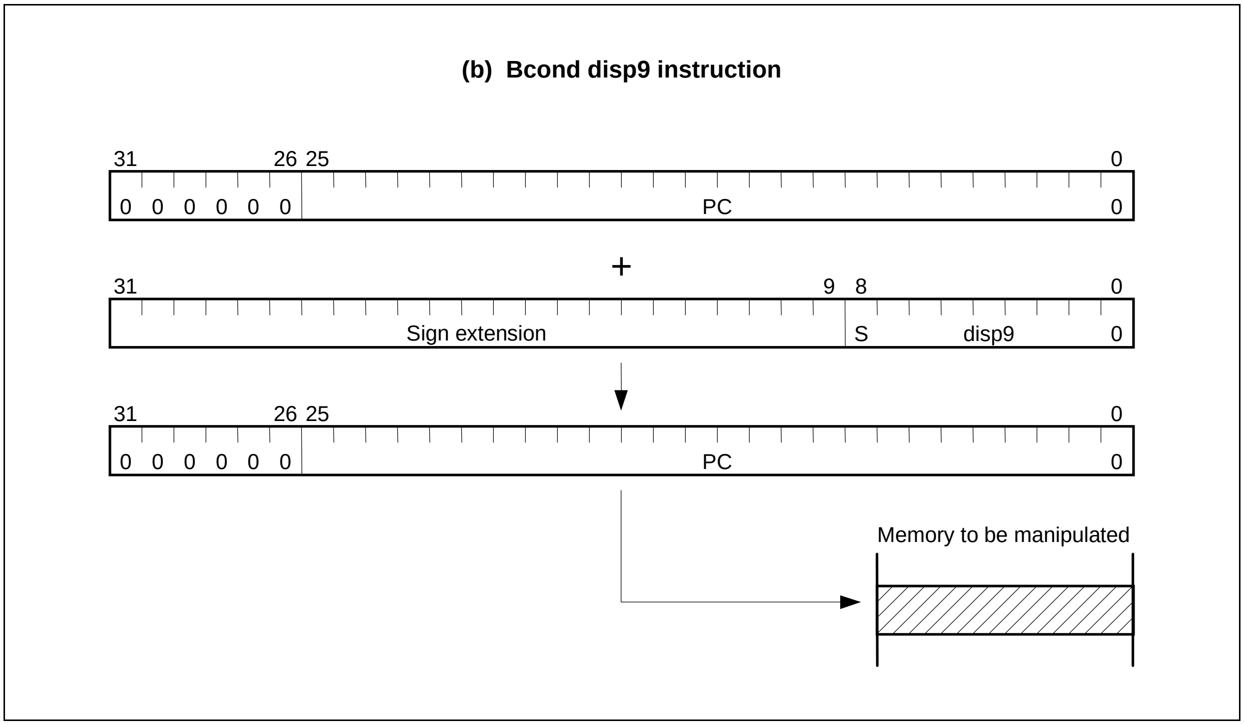


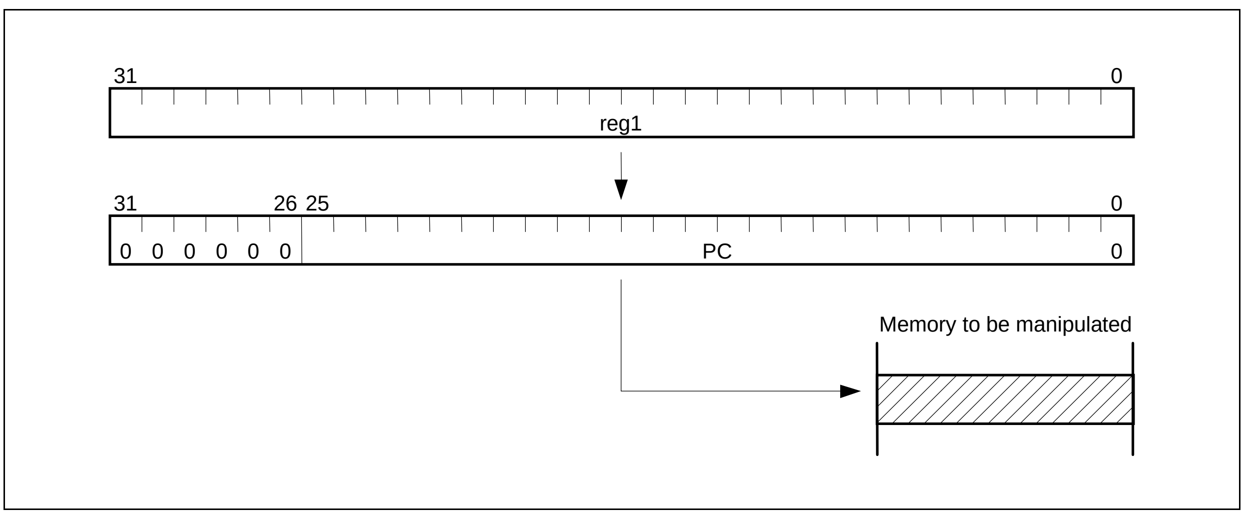
Figure 4-2. Relative Addressing (2/2)



(2) Register addressing (register indirect)

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

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



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 operand.

This addressing mode applies to instructions using the operand format reg1, reg2, reg3, or regID.

(2) Immediate addressing

The 5-bit or 16-bit data for manipulation is contained in the instruction code.

This addressing mode applies to instructions using the operand format imm5, imm16, vector, or cccc.

Remark vector: Operand that is 5-bit immediate data to specify trap vector (00H to 1FH), and is used in TRAP instruction.

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

(3) Based addressing

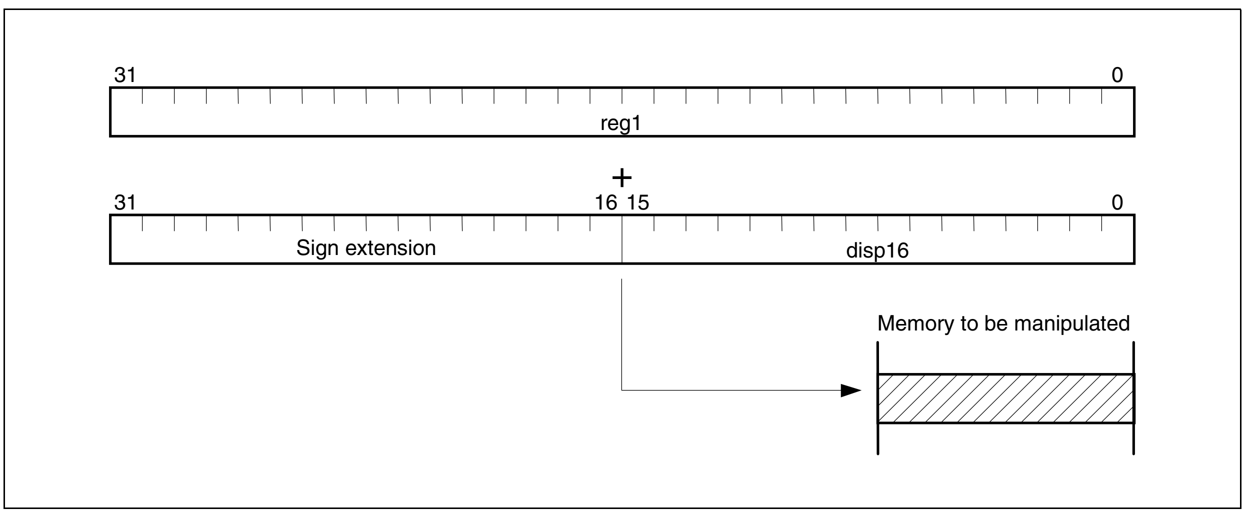
The following two types of based addressing are supported:

(a) Type 1

The address of the data memory location to be accessed is determined by adding the value in the specified general register (reg1) to the 16-bit displacement value (disp16) contained in the instruction code.

This addressing mode applies to instructions using the operand format disp16 [reg1].

Figure 4-4. Based Addressing (Type 1)

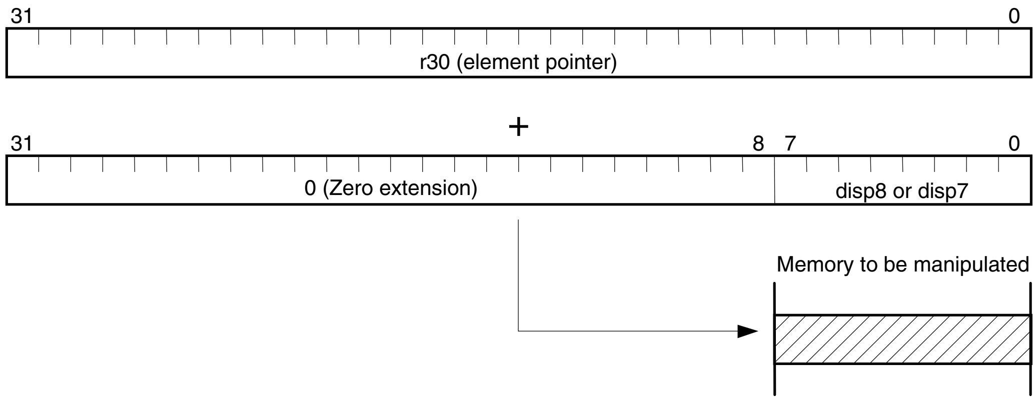


(b) Type 2

The address of the data memory location to be accessed is determined by adding the value in the element pointer (r30) to the 7- or 8-bit displacement value (disp7, disp8).

This addressing mode applies to SLD and SST instructions.

Figure 4-5. Based Addressing (Type 2)



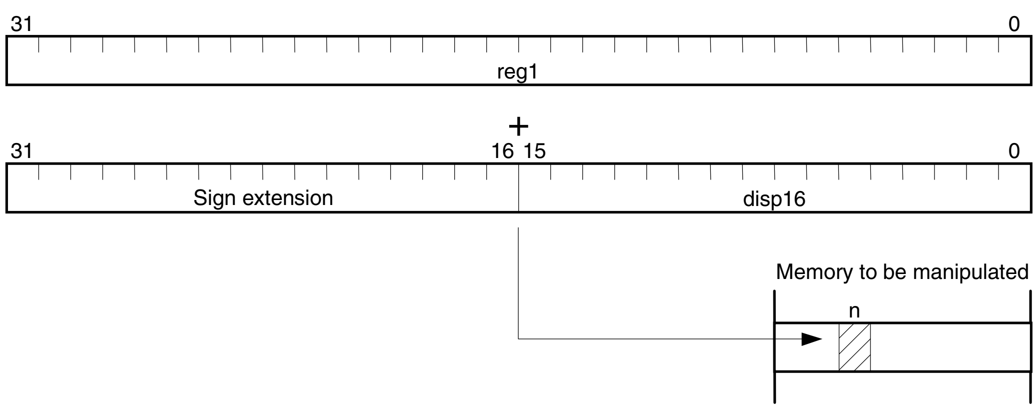
Remark Byte access: disp7
Half-word access and word access: disp8

(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 (reg1) and a 16-bit displacement (disp16) sign-extended to a word length.

This addressing mode applies only to bit manipulate instructions.

Figure 4-6. Bit Addressing



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

5.1 Instruction Format

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

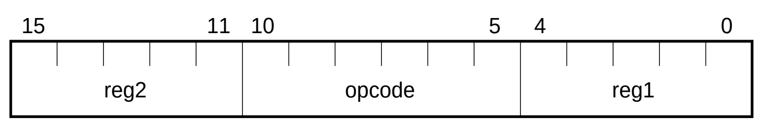
An instruction is actually stored in memory as follows:

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

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

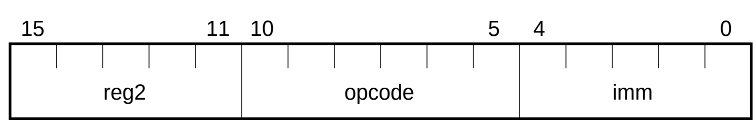
(1) reg-reg instruction (Format I)

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



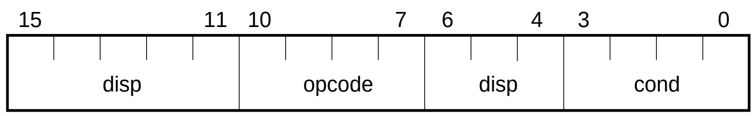
(2) imm-reg instruction (Format II)

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



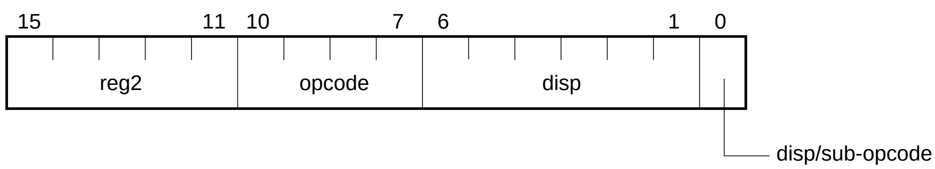
(3) Conditional branch instruction (Format III)

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

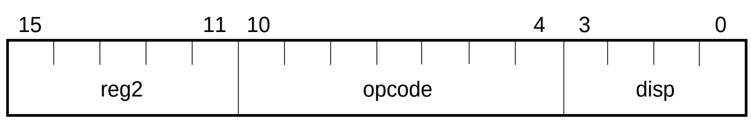


(4) 16-bit load/store instruction (Format IV)

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

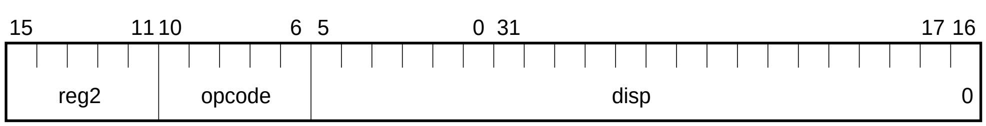


A 16-bit instruction format having a 7-bit opcode field, a general register specification field, and a 4-bit displacement field.



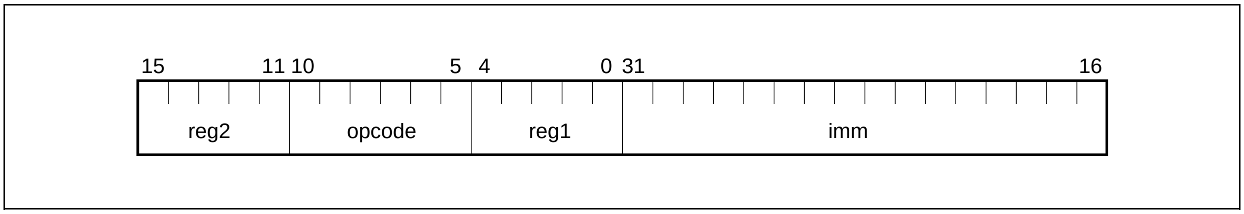
(5) Jump instruction (Format V)

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



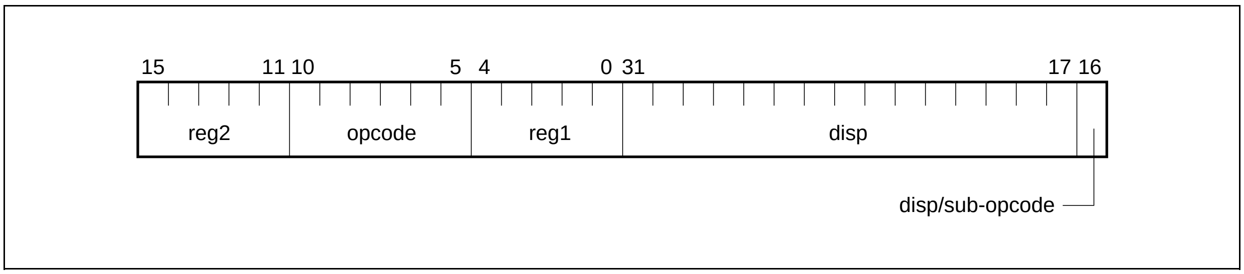
(6) 3-operand instruction (Format VI)

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



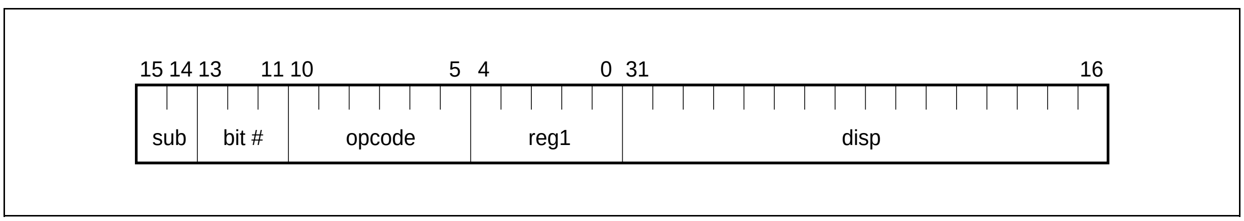
(7) 32-bit load/store instruction (Format VII)

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



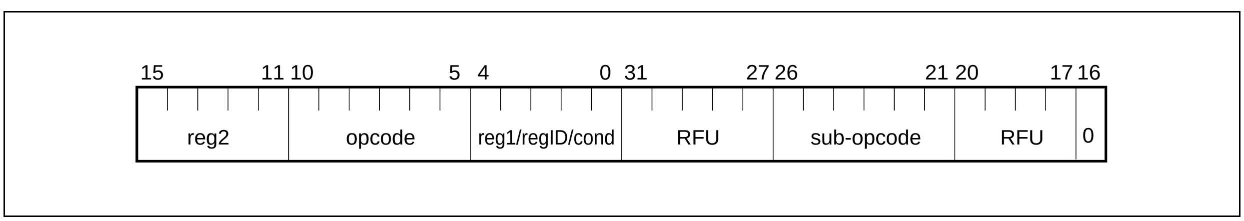
(8) Bit manipulation instruction (Format VIII)

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



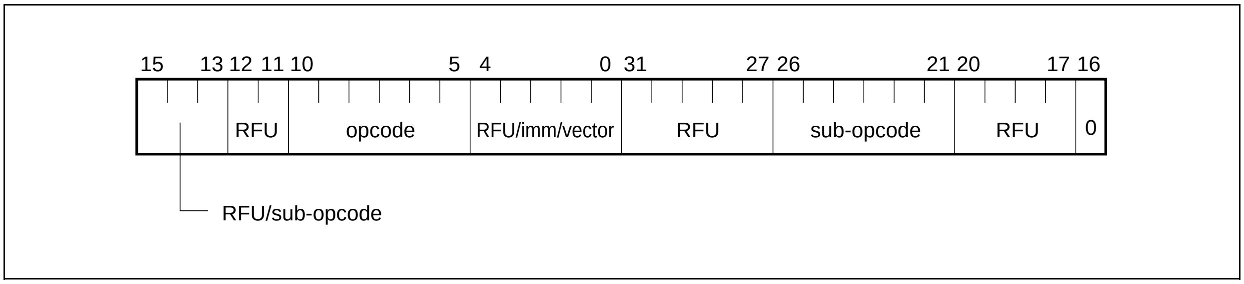
(9) Extended instruction format 1 (Format IX)

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



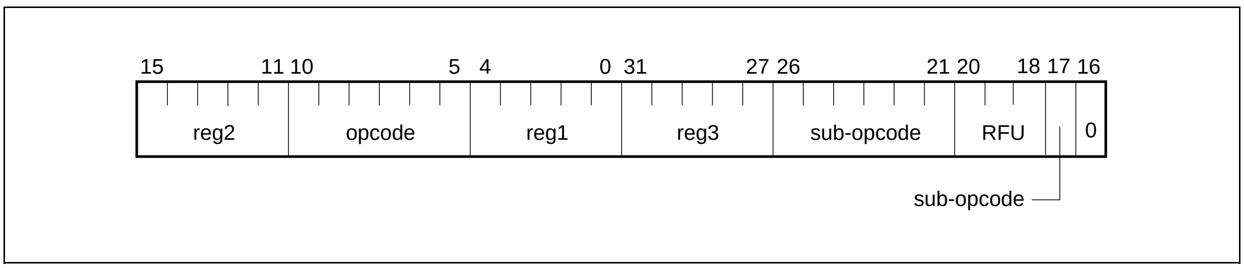
(10) Extended instruction format 2 (Format X)

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



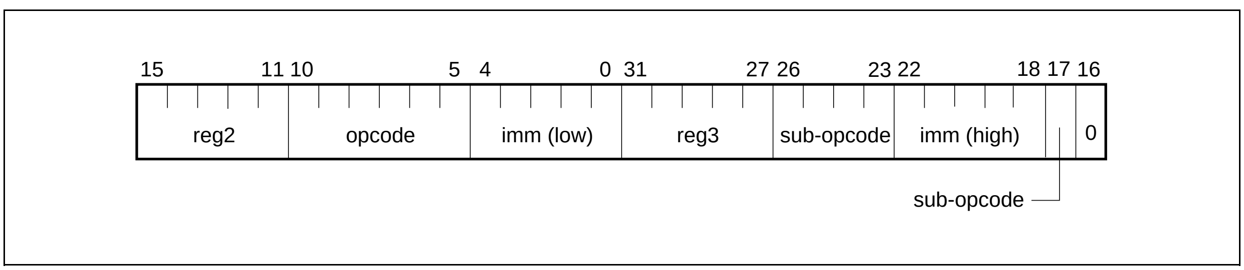
(11) Extended instruction format 3 (Format XI)

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



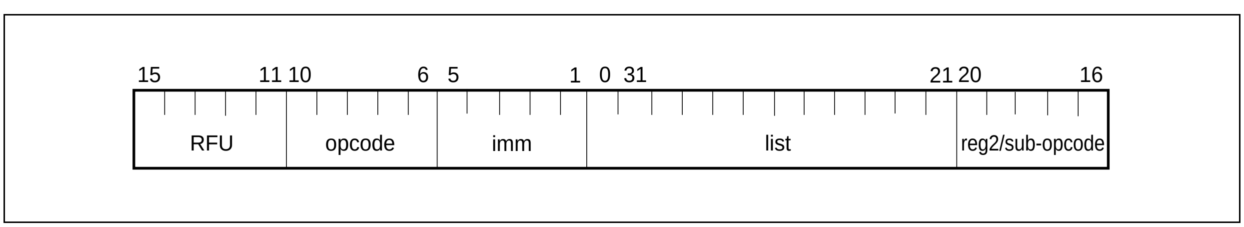
(12) Extended instruction format 4 (Format XII)

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



(13) Stack manipulation instruction 1 (Format XIII)

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



5.2 Outline of Instructions

(1) Load instructions

Transfer data from memory to a register. The following instructions (mnemonics) are provided.

(a) LD instructions

- LD.B: Load byte
- LD.BU: Load byte unsigned
- LD.H: Load half-word
- LD.HU: Load half-word unsigned
- LD.W: Load word

(b) SLD instructions

- SLD.B: Short format load byte
- SLD.BU: Short format load byte unsigned
- SLD.H: Short format load half-word
- SLD.HU: Short format load half-word unsigned
- SLD.W: Short format load word

(2) Store instructions

Transfer data from register to a memory. The following instructions (mnemonics) are provided.

(a) ST instructions

- ST.B: Store byte
- ST.H: Store half-word
- ST.W: Store word

(b) SST instructions

- SST.B: Short format store byte
- SST.H: Short format store half-word
- SST.W: Short format store word

(3) Multiply instructions

Execute multiply processing in 1 to 2 clocks with on-chip hardware multiplier. The following instructions (mnemonics) are provided.

- MUL: Multiply word
- MULH: Multiply half-word
- MULHI: Multiply half-word immediate
- MULU: Multiply word unsigned

(4) Arithmetic operation instructions

Add, subtract, divide, transfer, or compare data between registers. The following instructions (mnemonics) are provided.

- ADD: Add
- ADDI: Add immediate
- CMOV: Conditional move
- CMP: Compare
- DIV: Divide word
- DIVH: Divide half-word
- DIVHU: Divide half-word unsigned
- DIVU: Divide word unsigned
- MOV: Move
- MOVEA: Move effective address
- MOVHI: Move high half-word
- SASF: Shift and set flag condition
- SETF: Set flag condition
- SUB: Subtract
- SUBR: Subtract reverse

(5) Saturated operation instructions

Execute saturation addition and subtraction. If the result of the operation exceeds the maximum positive value (7FFFFFFFH), 7FFFFFFFH is returned. If the result of the operation exceeds the maximum negative value (80000000H), 80000000H is returned. The following instructions (mnemonics) are provided.

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

(6) Logical operation instructions

These instructions include logical operation 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 on-chip barrel shifter. The following instructions (mnemonics) are provided.

- AND: AND
- ANDI: AND immediate
- BSH: Byte swap half-word
- BSW: Byte swap word
- HSW: Half-word swap word
- NOT: NOT
- OR: OR
- ORI: OR immediate
- SAR: Shift arithmetic right
- SHL: Shift logical left
- SHR: Shift logical right
- SXB: Sign extend byte
- SXH: Sign extend half-word

- TST: Test
- XOR: Exclusive OR
- XORI: Exclusive OR immediate
- ZXB: Zero extend byte
- ZXH: Zero extend half-word

(7) Branch instructions

These instructions include unconditional branch instructions (JARL, JMP, JR) and conditional branch instruction (Bcond) which alters the control depending on the status of flags. Program control can be transferred to the address specified by a branch instruction. The following instructions (mnemonics) are provided.

- Bcond (BC, BE, BGE, BGT, BH, BL, BLE, BLT, BN, BNC, BNE, BNH, BNL, BNV, BNZ, BP, BR, BSA, BV, BZ): Branch on condition code
- JARL: Jump and register link
- JMP: Jump register
- JR: Jump relative

(8) Bit manipulation instructions

Execute a logical operation to bit data in memory. Only a specified bit is affected. The following instructions (mnemonics) are provided.

- CLR1: Clear bit
- NOT1: Not bit
- SET1: Set bit
- TST1: Test bit

(9) Special instructions

These instructions are instructions not included in the categories of instructions described above. The following instructions (mnemonics) are provided.

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

(10) Debug function instructions

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

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

★ **Caution The NB85E and NB85ET do not support debug function instructions.**

5.3 Instruction Set

In this section, mnemonic of each instruction is described divided into the following items.

- Instruction format: Indicates the description and operand of the instruction (for symbols, see **Table 5-1**).
- Operation: Indicates the function of the instruction (for symbols, see **Table 5-2**).
- Format: Indicates the instruction format (see **5.1 Instruction Format**).
- Opcode: Indicates the bit field of the instruction opcode (for symbols, see **Table 5-3**).
- Flag: Indicates the operation of the flag which is altered after executing the instruction.
  0 indicates clear (reset), 1 indicates set, and – indicates no change.
- Explanation: Explains the operation of the instruction.
- Remark: Explains the supplementary information of the instruction.
- Caution: Indicates the cautions.

Table 5-1. Conventions of Instruction Format

Symbol	Meaning
reg1	General register (used as source register)
reg2	General register (mainly used as destination register. Some are also used as source registers)
reg3	General register (mainly used as remainder of division results or higher 32 bits of multiply results)
bit#3	3-bit data for specifying bit number
imm×	×-bit immediate data
disp×	×-bit displacement data
regID	System register number
vector	5-bit data for trap vector (00H to1FH) specification
cccc	4-bit data for condition code specification
sp	Stack pointer (r3)
ep	Element pointer (r30)
list×	Lists of registers (× is a maximum number of registers)

Table 5-2. Conventions of Operation (1/2)

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

Table 5-2. Conventions of Operation (2/2)

Symbol	Meaning
saturated (n)	Performs saturation processing of n. If n $\geq$ 7FFFFFFFH as result of calculation, n = 7FFFFFFFH. If n $\leq$ 80000000H as result of calculation, n = 80000000H.
result	Reflects result on flag
Byte	Byte (8 bits)
Half-word	Half-word (16 bits)
Word	Word (32 bits)
+	Add
–	Subtract
||	Bit concatenation
×	Multiply
÷	Divide
%	Remainder of division results
AND	And
OR	Or
XOR	Exclusive Or
NOT	Logical negate
logically shift left by	Logical left shift
logically shift right by	Logical right shift
arithmetically shift right by	Arithmetic right shift

Table 5-3. Conventions of Opcode

Symbol	Meaning
R	1-bit data of code specifying reg1 or regID
r	1-bit data of code specifying reg2
w	1-bit data of code specifying reg3
d	1-bit data of displacement
I	1-bit data of immediate (indicates higher bits of immediate)
i	1-bit data of immediate
cccc	4-bit data for condition code specification
CCCC	4-bit data for condition code specification of Bcond instruction
bbb	3-bit data for bit number specification
L	1-bit data of code specifying program register in register list

<Arithmetic operation instruction>

Add register/immediate

ADD

Add

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

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

Format (1) Format I
(2) Format II

Opcode

(1)
```
15             0
rrrrr001110RRRRR
```

(2)
```
15             0
rrrrr010010iiiii
```

Flag

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

Explanation (1) Adds the word data of general register reg1 to the word data of general register reg2, and stores the result to 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 to general register reg2.

<Arithmetic operation instruction>

Add immediate

ADDI

Add Immediate

Instruction format ADDI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110000RRRRR	iiiiiiiiiiiiiiii

Flag

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

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

<Logical operation instruction>

AND

AND

And

Instruction format AND reg1, reg2

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

Format Format I

Opcode

```
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	–

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

<Logical operation instruction>

AND immediate

ANDI

And Immediate

Instruction format ANDI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110110RRRRR	iiiiiiiiiiiiiiii

Flag

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

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 to general register reg2. The data of general register reg1 is not affected.

<Branch instruction>

Branch on condition code with 9-bit displacement

Bcond

Branch on Condition Code

Instruction format Bcond disp9

Operation if conditions are satisfied
then PC $\leftarrow$ PC + sign-extend (disp9)

Format Format III

Opcode

```
15              0
ddddd1011dddCCCC
```

`dddddddd` is the higher 8 bits of disp9.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation Tests each flag of PSW 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 5-4. Bcond Instructions

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

Caution If executing a conditional branch instruction of a signed integer (BGE, BGT, BLE, or BLT) 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 operations, if an overflow occurs, the S flag is inverted ($0 \rightarrow 1$ or $1 \rightarrow 0$). This is because the result is a negative value if it exceeds the maximum positive value and it is a positive value if it exceeds the maximum negative value. However, when a saturated operation instruction is executed, and if the result exceeds the maximum positive value, the result is saturated with a positive value; if the result exceeds the maximum negative value, the result is saturated with a negative value. Unlike the ordinary operation, therefore, the S flag is not inverted even if an overflow occurs. Hence, the S flag is affected differently when the instruction is a saturate operation, as opposed to an ordinary operation. A branch condition which is an XOR of S and OV flags will therefore have no meaning.

<Logical operation instruction>

BSH	**Byte swap half-word** **Byte Swap Half-word**

Instruction format BSH reg2, reg3

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

Format Format XII

Opcode

15 0	31 16
rrrrr11111100000	wwwww01101000010

Flag

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

Explanation Endian translation.

<Logical operation instruction>

Byte swap word

BSW

Byte Swap Word

Instruction format BSW reg2, reg3

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

Format Format XII

Opcode

15 0	31 16
rrrrr11111100000	wwwww01101000000

Flag

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

Explanation Endian translation.

<Special instruction>

Call with table look up

CALLT

Call with Table Look Up

Instruction format CALLT imm6

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

Format Format II

Opcode

```
15              0
0000001000iiiiii
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation Saves the restore PC and PSW to CTPC and CTPSW. Adds the CTBP and data of imm6, logically shifted left by 1 and zero-extended to word length, to generate a 32-bit table entry address. Then load the half-word entry data, zero-extended to word length, and adds the data and CTBP to generate a 32-bit target address. Then jump to a target address.

Caution If an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. Execution is resumed after returning from the interrupt.

<Bit manipulation instruction>

CLR1

Clear bit

Clear Bit

Instruction format

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

Operation

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

Format

(1) Format VIII
(2) Format IX

Opcode

(1)

15 0	31 16
10bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100100

Flag

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

Explanation

(1) 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. Bit not specified is not affected.
(2) Reads the data of general register reg1 to generate a 32-bit address. Then clears the bit, specified by the data of lower 3 bits of reg2, of the byte data referenced by the generated address. Bit not specified 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.

<Arithmetic operation instruction>

Conditional move

CMOV

Conditional Move

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

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

Format (1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww011001cccc0

(2)

15 0	31 16
rrrrr111111iiiii	wwwww011000cccc0

Flag
CY –
OV –
S –
Z –
SAT –

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

Remark See SETF instruction.

<Arithmetic operation instruction>

CMP	**Compare register/immediate (5-bit)** **Compare**

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

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

Format (1) Format I
(2) Format II

Opcode

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

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

Flag

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

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 flags of PSW. 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 are 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 flags of PSW. 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.

<Special instruction>

Return from CALLT

CTRET

Return from CALLT

Instruction format CTRET

Operation PC ← CTPC
PSW ← CTPSW

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101000100

Flag

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

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

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

<Debug function instruction>

Return from debug trap

DBRET

Return from debug trap

Instruction format DBRET

Operation PC ← DBPC
PSW ← DBPSW

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101000110

Flag

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

Explanation Fetches the restore PC and PSW from the appropriate system register and returns from debug mode.

Caution (1) Because the DBRET instruction is for debugging, it is essentially used by debug tools. When a debug tool is using this instruction, therefore, use of it in the application may cause a malfunction.

★ (2) The NB85E and NB85ET do not support the DBRET instruction.

<Debug function instruction>

Debug trap

DBTRAP

Debug trap

Instruction format DBTRAP

Operation

DBPC ← PC + 2 (restore PC)
DBPSW ← PSW
PSW.NP ← 1
PSW.EP ← 1
PSW.ID ← 1
PC ← 00000060H

Format Format I

Opcode

```
15              0
1111100001000000
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation

Saves the contents of the restore PC (address of the instruction following the DBTRAP instruction) and the PSW to the DBPC and DBPSW, respectively, and sets the NP, EP, and ID flags of PSW to 1.
Next, the handler address (00000060H) of the exception trap is set to the PC, and control shifts to the PC. PSW flags other than NP, EP, and ID flags are unaffected.
Note that the value saved to the DBPC is the address of the instruction following the DBTRAP instruction.

Caution

(1) Because the DBTRAP instruction is for debugging, it is essentially used by debug tools. When a debug tool is using this instruction, therefore, use of it in the application may cause a malfunction.
★ (2) The NB85E and NB85ET do not support the DBTRAP instruction.

<Special instruction>

Disable interrupt

DI

Disable Interrupt

Instruction format DI

Operation PSW.ID ← 1 (Disables maskable interrupt)

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101100000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–
ID	1

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

Remark Interrupts are not sampled during execution of this instruction. The PSW 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.

<Special instruction>

Function dispose

DISPOSE

Function Dispose

Instruction format

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

Operation

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

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

Format

Format XIII

Opcode

(1)

15 — 0	31 — 16
`0000011001iiiiiL`	`LLLLLLLLLLL00000`

(2)

15 — 0	31 — 16
`0000011001iiiiiL`	`LLLLLLLLLLLRRRRR`

`RRRRR` must not be 00000.

`LLLLLLLLLLLL` shows a register in list12. Bit assignment of list12 is below.

```
15                               0   31   2827   2423  21        16
- - - - - - - - - - - - - - - - 3   2 2 2 2 2 2 2 2 2 2 3 - - - - -
- - - - - - - - - - - - - - - - 0   4 5 6 7 0 1 2 3 8 9 1 - - - - -
```

Flag

CY –
OV –
S –
Z –
SAT –

Explanation

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

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

Remark General registers in list12 are loaded in the downward direction. (r31, r30, ... r20)

The 5-bit immediate imm5 is used to restore a stack frame for auto variables and temporary data.

The lower 2-bit of address specified by sp is always masked to 0 even if misaligned access is enabled.

If an interrupt occurs before updating the sp, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction (sp will retain their original values prior to the start of execution).

Caution If an interrupt is generated during instruction execution, due to manipulation of the stack, the execution of that instruction may stop after the read/write cycle and register value rewriting are complete. Execution is resumed after returning from the interrupt.

<Arithmetic operation instruction>

DIV	**Divide word**
	Divide Word

Instruction format DIV reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01011000000

Flag

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

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

Remark 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, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction. Also, general registers reg1 and reg2 will retain their original values prior to the start of execution.

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

<Arithmetic operation instruction>

DIVH

Divide half-word

Divide Half-word

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

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

Format (1) Format I
(2) Format XI

Opcode

(1)

15 0
rrrrr000010RRRRR

(2)

15 0	31 16
rrrrr111111RRRRR	wwwww01010000000

Flag

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

Explanation

(1) Divides the word data of general register reg2 by the lower half-word data of general register reg1, and stores the quotient to 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.

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

Remark

(1) 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, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction. Also, general registers reg1 and reg2 will retain their original values prior to the start of execution.
Do not specify r0 as the destination register reg2.
The higher 16 bits of general register reg1 are ignored when division is executed.

(2) 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, execution is aborted, and the interrupt is processed. Upon returning from the interrupt, the execution is restarted from the beginning, with the return address being the address of this instruction. 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.
If the address of reg2 is the same as the address of reg3, the remainder is stored to reg2 (= reg3).

<Arithmetic operation instruction>

Divide half-word unsigned

DIVHU

Divide Half-word Unsigned

Instruction format DIVHU reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01010000010

Flag

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

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

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

<Arithmetic operation instruction>

Divide word unsigned

DIVU

Divide Word Unsigned

Instruction format DIVU reg1, reg2, reg3

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

Format Format XI

Opcode

15 0	31 16
rrrrr111111RRRRR	wwwww01011000010

Flag

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

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

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

<Special instruction>

EI	**Enable interrupt** **Enable Interrupt**

Instruction format EI

Operation PSW.ID ← 0 (enables maskable interrupt)

Format Format X

Opcode

```
15              0  31              16
1000011111100000 | 0000000101100000
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–
ID	0

Explanation Clears 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.

<Special instruction>

Halt

HALT

Halt

Instruction format HALT

Operation Halts

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000100100000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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
- Non-maskable interrupt request (NMI input)
- Maskable interrupt request (when ID of PSW = 0)

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

<Logical operation instruction>

Half-word swap word

HSW

Half-word Swap Word

Instruction format HSW reg2, reg3

Operation GR [reg3] ← GR [reg2] (15:0) || GR [reg2] (31:16)

Format Format XII

Opcode

15 0	31 16
rrrrr11111100000	wwwww01101000100

Flag

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

Explanation Endian translation.

<Branch instruction>

JARL	**Jump and register link**
	Jump and Register Link

Instruction format JARL disp22, reg2

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

Format Format V

Opcode

15 0	31 16
`rrrrr11110dddddd`	`ddddddddddddddd0`

`ddddddddddddddddddddd` is the higher 21 bits of disp22.

Flag

CY –
OV –
S –
Z –
SAT –

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 restore PC address to general register reg2. The JMP instruction, which is equivalent to a subroutine-return instruction, can be used to specify as reg1 the general register containing the return address saved during the JARL subroutine-call instruction, to restore the program counter.

<Branch instruction>

JMP	**Jump register** **Jump Register**

Instruction format JMP [reg1]

Operation PC ← GR [reg1]

Format Format I

Opcode

```
15            0
0000000000011RRRRR
```

Flag

CY –
OV –
S –
Z –
SAT –

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.

<Branch instruction>

Jump relative

JR

Jump Relative

Instruction format JR disp22

Operation PC ← PC + sign-extend (disp22)

Format Format V

Opcode

15 0	31 16
0000011110dddddd	ddddddddddddddd0

`ddddddddddddddddddddd` is the higher 21 bits of disp22.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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.

<Load instruction>

LD.B	**Load byte**
	Load

Instruction format LD.B disp16 [reg1], reg2

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

Format Format VII

Opcode

15 0	31 16
rrrrr111000RRRRR	dddddddddddddddd

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 stored to general register reg2.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Load instruction>

Load byte unsigned

LD.BU

Load

Instruction format LD.BU disp16 [reg1], reg2

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

Format Format VII

Opcode

15 — 0	31 — 16
`rrrrr11110bRRRRR`	`ddddddddddddddd1`

`ddddddddddddddd` is the higher 15 bits of disp16. `b` is the bit 0 of disp16.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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, zero-extended to word length, and stored to general register reg2.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Load instruction>

LD.H	**Load half-word**
	Load

Instruction format LD.H disp16 [reg1], reg2

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

Format Format VII

Opcode

15 0	31 16
`rrrrr111001RRRRR`	`ddddddddddddddd0`

`ddddddddddddddd` is the higher 15 bits of disp16.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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 to general register reg2.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Load instruction>

Load half-word unsigned

LD.HU

Load

Instruction format LD.HU disp16 [reg1], reg2

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

Format Format VII

Opcode

15 0	31 16
rrrrr111111RRRRR	ddddddddddddddd1

ddddddddddddddd is the higher 15 bits of disp16.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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, zero-extended to word length, and stored to general register reg2.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Load instruction>

LD.W	**Load word**
	Load

Instruction format LD.W disp16 [reg1], reg2

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

Format Format VII

Opcode

15 0	31 16
`rrrrr111001RRRRR`	`ddddddddddddddd1`

`ddddddddddddddd` is the higher 15 bits of disp16.

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to general register reg2.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Special instruction>

LDSR	**Load to system register**
	Load to System Register

Instruction format LDSR reg2, regID

Operation SR [regID] ← GR [reg2]

Format Format IX

Opcode

15 ... 0	31 ... 16
rrrrr111111RRRRR	0000000000100000

Caution **The source register in this instruction is represented by reg2 for convenience of describing its mnemonic . In the opcode, however, the reg1 field is used for the source register. Unlike other instructions therefore, the register specified in the mnemonic description has a different meaning in the opcode.**

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.)

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. 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.

<Arithmetic operation instruction>

Move register/immediate (5-bit)/immediate (32-bit)

MOV

Move

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

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

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

Opcode

(1)

15 … 0
`rrrrr000000RRRRR`

(2)

15 … 0
`rrrrr010000iiiii`

(3)

15 … 0	31 … 16	47 … 32
`00000110001RRRRR`	`iiiiiiiiiiiiiiii`	`IIIIIIIIIIIIIIII`

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

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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. Do not specify r0 as the destination register reg2.
(3) Transfers the value of a 32-bit immediate data to general register reg1.

<Arithmetic operation instruction>

Move effective address

MOVEA

Move Effective Address

Instruction format MOVEA imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110001RRRRR	iiiiiiiiiiiiiiii

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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

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

<Arithmetic operation instruction>

Move high half-word

MOVHI

Move High Half-word

Instruction format MOVHI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110010RRRRR	iiiiiiiiiiiiiiii

Flag

CY –
OV –
S –
Z –
SAT –

Explanation Adds a word data, 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.
Do not specify r0 as the destination register reg2.

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

<Multiply instruction>

Multiply word by register/immediate (9-bit)

MUL

Multiply Word

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

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

Format
(1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	wwwww01000100000

(2)

15 0	31 16
rrrrr111111iiiii	wwwww01001IIII00

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

Flag
CY –
OV –
S –
Z –
SAT –

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

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

<Multiply instruction>

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

MULH

Multiply Half-word

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

Operation (1) GR [reg2] (32) $\leftarrow$ GR [reg2] (16) $\times$ GR [reg1] (16)
(2) GR [reg2] $\leftarrow$ GR [reg2] $\times$ sign-extend (imm5)

Format (1) Format I
(2) Format II

Opcode

(1)
```
15             0
rrrrr000111RRRRR
```

(2)
```
15             0
rrrrr010111iiiii
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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 to general register reg2 as word data.
The data of general register reg1 is not affected.
Do not specify r0 as the destination register reg2.
(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 to general register reg2.
Do not specify r0 as the destination register reg2.

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

<Multiply instruction>

Multiply half-word by immediate (16-bit)

MULHI

Multiply Half-word Immediate

Instruction format MULHI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110111RRRRR	iiiiiiiiiiiiiiii

Flag

CY –
OV –
S –
Z –
SAT –

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

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

<Multiply instruction>

Multiply word by register/immediate (9-bit)

MULU

Multiply Word Unsigned

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

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

Format (1) Format XI
(2) Format XII

Opcode

(1)

15 0	31 16
`rrrrr111111RRRRR`	`wwwww01000100010`

(2)

15 0	31 16
`rrrrr111111iiiii`	`wwwww01001IIII10`

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

Flag

CY –
OV –
S –
Z –
SAT –

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

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

<Special instruction>

NOP	**No operation** **No Operation**

Instruction format NOP

Operation Executes nothing and consumes at least one clock.

Format Format I

Opcode

```
15              0
0000000000000000
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation Executes nothing and consumes at least one clock cycle.

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

<Logical operation instruction>

NOT

NOT

Not

Instruction format NOT reg1, reg2

Operation GR [reg2] ← NOT (GR [reg1])

Format Format I

Opcode

```
15               0
rrrrr000001RRRRR
```

Flag

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

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

<Bit manipulation instruction>

NOT bit

NOT1

Not Bit

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

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

Format (1) Format VIII
(2) Format IX

Opcode

(1)

15 0	31 16
01bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100010

Flag

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

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. The bit, specified by the 3-bit bit number, is inverted (0 → 1 or 1 → 0) at the byte data location referenced by the generated address. The bits other than the specified bit are not affected.
(2) Reads the data of general register reg1 to generate a 32-bit address. The bit, specified by the data of lower 3 bits of reg2 is inverted (0 → 1 or 1 → 0) 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.

<Logical operation instruction>

OR

OR

Or

Instruction format OR reg1, reg2

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

Format Format I

Opcode

```
15               0
rrrrr001000RRRRR
```

Flag

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

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

<Logical operation instruction>

OR immediate (16-bit)

ORI

Or Immediate

Instruction format ORI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110100RRRRR	iiiiiiiiiiiiiiii

Flag

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

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 to general register reg2. The data of general register reg1 is not affected.

<Special instruction>

PREPARE	**Function prepare** **Function Prepare**

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

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

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

Format Format XIII

Opcode

(1)

15 … 0	31 … 16
0000011110iiiiiL	LLLLLLLLLLL00001

(2)

15 … 0	31 … 16	Optional (47 to 32 or 63 to 32)
0000011110iiiiiL	LLLLLLLLLLLff011	imm16 / imm32

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

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

LLLLLLLLLLLL shows a register in list12. Bit assignment of list12 is below.

15 … 0	31 28 27 24 23 21 16
- - - - - - - - - - - - - - - 3	2 2 2 2 2 2 2 2 2 2 3 - - - - -
- - - - - - - - - - - - - - - 0	4 5 6 7 0 1 2 3 8 9 1 - - - - -

Flag
CY –
OV –
S –
Z –
SAT –

Explanation

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

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

Remark

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

Caution

If an interrupt is generated during instruction execution, due to manipulation of the stack, the execution of that instruction may stop after the read/write cycle and register value rewriting are complete.

<Special instruction>

RETI	**Return from trap or interrupt**
	Return from Trap or Interrupt

Instruction format RETI

Operation

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

Format Format X

Opcode

15 0	31 16
0000011111100000	0000000101000000

Flag

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

Explanation This instruction reads the restore PC and PSW from the appropriate system register, and operation returns from a software 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 restore PC and PSW values are set to the PC and PSW, the operation returns to the address immediately before the trap or interrupt occurred.

Caution When returning from a non-maskable interrupt or software exception routine using the RETI instruction, the NP and EP flags of PSW must be set accordingly to restore the PC and PSW:

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

<Logical operation instruction>

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

SAR

Shift Arithmetic Right

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

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

Format (1) Format IX
(2) Format II

Opcode

(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	–

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.

<Logical operation instruction>

SASF	**Shift and set flag condition**
	Shift and Set Flag Condition

Instruction format SASF cccc, reg2

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

Format Format IX

Opcode

15 0	31 16
rrrrr1111110cccc	0000001000000000

Flag

CY –
OV –
S –
Z –
SAT –

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

Remark See SETF instruction.

<Saturated operation instruction>

Saturated add register/immediate (5-bit)

SATADD

Saturated Add

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

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

Format (1) Format I
(2) Format II

Opcode

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

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

Flag

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

Explanation

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

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

Remark The SAT flag is a cumulative flag. Once the result of the saturated operation instruction has been saturated, this flag is set to 1 and is not cleared 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 clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

<Saturated operation instruction>

	Saturated subtract
SATSUB	
	Saturated Subtract

Instruction format SATSUB reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr000101RRRRR
```

Flag

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

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

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not cleared 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 clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

<Saturated operation instruction>

Saturated subtract immediate

SATSUBI

Saturated Subtract Immediate

Instruction format SATSUBI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110011RRRRR	iiiiiiiiiiiiiiii

Flag

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

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

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not cleared 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 clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

<Saturated operation instruction>

Saturated subtract reverse

SATSUBR

Saturated Subtract Reverse

Instruction format SATSUBR reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr000100RRRRR
```

Flag

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

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

Remark The SAT flag is a cumulative flag. Once the result of the operation of the saturated operation instruction has been saturated, this flag is set to 1 and is not cleared 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 clear the SAT flag to 0, load data to the PSW by using the LDSR instruction.

<Bit manipulation instruction>

SET1

Set bit

Set Bit

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

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

Format (1) Format VIII
(2) Format IX

Opcode

(1)

15 0	31 16
00bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100000

Flag

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

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. The bit, specified by the 3-bit bit number, is set at the byte data location referenced by the generated address. The bits other than the specified bit are not affected.
(2) Reads the data of general register reg1 to generate a 32-bit address. The bit, specified by the data of lower 3 bits of reg2, 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.

<Arithmetic operation instruction>

SETF	**Set flag condition** **Set Flag Condition**

Instruction format SETF cccc, reg2

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

Format Format IX

Opcode

15 ... 0	31 ... 16
rrrrr1111110cccc	0000000000000000

Flag
CY –
OV –
S –
Z –
SAT –

Explanation The general register reg2 is set to 1 if a condition specified by condition code "cccc" is satisfied; otherwise, 0 are stored to the register. One of the codes shown in **Table 5-5 Condition Codes** should be specified as the condition code "cccc".

Remark Here are some examples of using this instruction:

(1) Translation of two or more condition clauses
If A of statement if (A) in C language consists of two or more condition clauses (a_1, a_2, a_3, and so on), it is usually translated to a sequence of if (a_1) then, if (a_2) then. The object code executes "conditional branch" by checking the result of evaluation equivalent to a_n. Since a pipeline processor takes more time to execute "condition judgment" + "branch" than to execute an ordinary operation, the result of evaluating each condition clause if (a_n) is stored 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 5-5. 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 (unconditional)
1101	SA	SAT = 1
0110	LT	(S xor OV) = 1
1110	GE	(S xor OV) = 0
0111	LE	((S xor OV) or Z) = 1
1111	GT	((S xor OV) or Z) = 0

<Logical operation instruction>

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

SHL

Shift Logical Left

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

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

Format
(1) Format IX
(2) Format II

Opcode

(1)

15 ... 0	31 ... 16
rrrrr111111RRRRR	0000000011000000

(2)

15 ... 0
rrrrr010110iiiii

Flag

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

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.

<Logical operation instruction>

SHR	**Shift logical right by register/immediate (5-bit)** **Shift Logical Right**

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

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

Format (1) Format IX
(2) Format II

Opcode

(1)

15 0	31 16
rrrrr111111RRRRR	0000000010000000

(2)

15 0
rrrrr010100iiiii

Flag

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

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.

<Load instruction>

SLD.B	**Short format load byte**
	Load

Instruction format SLD.B disp7 [ep], reg2

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

Format Format IV

Opcode

```
15              0
rrrrr0110ddddddd
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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 to reg2.

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

For details on misaligned access, see **3.3 Data Alignment**.
Also, if an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. In this case, the instruction is re-executed after returning from the interrupt. Therefore, except in cases when clearly no interrupt is generated, the LD instruction should be used for accessing I/O, FIFO types, or other resources whose status is changed by the read cycle (the bus cycle is not re-executed even if an interrupt is generated while the LD or store instruction is being executed).

<Load instruction>

SLD.BU	**Short format load byte unsigned**
	Load

Instruction format SLD.BU disp4 [ep], reg2

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

Format Format IV

Opcode

```
 15             0
 rrrrr0000110dddd
```

`rrrrr` must not be 00000.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation Adds the 4-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored to reg2.

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

For details on misaligned access, see **3.3 Data Alignment**.
Also, if an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. In this case, the instruction is re-executed after returning from the interrupt. Therefore, except in cases when clearly no interrupt is generated, the LD instruction should be used for accessing I/O, FIFO types, or other resources whose status is changed by the read cycle (the bus cycle is not re-executed even if an interrupt is generated while the LD or store instruction is being executed).

<Load instruction>

Short format load half-word

SLD.H

Load

Instruction format SLD.H disp8 [ep], reg2

Operation adr ← ep + zero-extend (disp8)
GR [reg2] ← sign-extend (Load-memory (adr, Half-word))

Format Format IV

Opcode

```
15              0
rrrrr1000ddddddd
```

`ddddddd` is the higher 7 bits of disp8.

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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 to reg2.

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

For details on misaligned access, see **3.3 Data Alignment**.
Also, if an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. In this case, the instruction is re-executed after returning from the interrupt. Therefore, except in cases when clearly no interrupt is generated, the LD instruction should be used for accessing I/O, FIFO types, or other resources whose status is changed by the read cycle (the bus cycle is not re-executed even if an interrupt is generated while the LD or store instruction is being executed).

<Load instruction>

Short format load half-word unsigned

SLD.HU

Load

Instruction format SLD.HU disp5 [ep], reg2

Operation adr ← ep + zero-extend (disp5)
GR [reg2] ← zero-extend (Load-memory (adr, Half-word))

Format Format IV

Opcode

```
15             0
rrrrr0000111dddd
```

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

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation Adds the 5-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Half-word data is read from this 32-bit address with bit 0 masked to 0, zero-extended to word length, and stored to reg2.

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

For details on misaligned access, see **3.3 Data Alignment**.
Also, if an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. In this case, the instruction is re-executed after returning from the interrupt. Therefore, except in cases when clearly no interrupt is generated, the LD instruction should be used for accessing I/O, FIFO types, or other resources whose status is changed by the read cycle (the bus cycle is not re-executed even if an interrupt is generated while the LD or store instruction is being executed).

<Load instruction>

# SLD.W	**Short format load word**
	Load

Instruction format SLD.W disp8 [ep], reg2

Operation adr ← ep + zero-extend (disp8)
GR [reg2] ← Load-memory (adr, Word)

Format Format IV

Opcode

```
15              0
rrrrr1010dddddd0
```

`dddddd` is the higher 6 bits of disp8.

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to reg2.

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

For details on misaligned access, see **3.3 Data Alignment**.
Also, if an interrupt is generated during instruction execution, the execution of that instruction may stop after the end of the read/write cycle. In this case, the instruction is re-executed after returning from the interrupt. Therefore, except in cases when clearly no interrupt is generated, the LD instruction should be used for accessing I/O, FIFO types, or other resources whose status is changed by the read cycle (the bus cycle is not re-executed even if an interrupt is generated while the LD or store instruction is being executed).

<Store instruction>

SST.B	**Short format store byte**
	Store

Instruction format SST.B reg2, disp7 [ep]

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

Format Format IV

Opcode

```
15              0
rrrrr0111ddddddd
```

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to the generated address.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Store instruction>

Short format store half-word

SST.H

Store

Instruction format SST.H reg2, disp8 [ep]

Operation adr ← ep + zero-extend (disp8)
Store-memory (adr, GR [reg2], Half-word)

Format Format IV

Opcode

```
15              0
rrrrr1001ddddddd
```

`ddddddd` is the higher 7 bits of disp8.

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to the generated 32-bit address with bit 0 masked to 0.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Store instruction>

Short format store word

SST.W

Store

Instruction format SST.W reg2, disp8 [ep]

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

Format Format IV

Opcode

```
15             0
rrrrr1010dddddd1
```

`dddddd` is the higher 6 bits of disp8.

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to the generated 32-bit address with bits 0 and 1 masked to 0.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Store instruction>

ST.B	**Store byte**
	Store

Instruction format ST.B reg2, disp16 [reg1]

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

Format Format VII

Opcode

15 0	31 16
rrrrr111010RRRRR	dddddddddddddddd

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

Explanation 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 to the generated address.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
(when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Store instruction>

ST.H	**Store half-word**
	Store

Instruction format ST.H reg2, disp16 [reg1]

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

Format Format VII

Opcode

15 　　　　　　　0	31 　　　　　　　16
`rrrrr111011RRRRR`	`ddddddddddddddd0`

`ddddddddddddddd` is the higher 15 bits of disp16.

Flag
CY –
OV –
S –
Z –
SAT –

Explanation 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 to the generated 32-bit address with bit 0 masked to 0. Therefore, stored data is automatically aligned on a half-word boundary.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
  (when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Store instruction>

Store word

ST.W

Store

Instruction format ST.W reg2, disp16 [reg1]

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

Format Format VII

Opcode

15 0	31 16
`rrrrr111011RRRRR`	`ddddddddddddddd1`

`ddddddddddddddd` is the higher 15 bits of disp16.

Flag

CY –
OV –
S –
Z –
SAT –

Explanation 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 to the generated 32-bit address with bits 0 and 1 masked to 0. Therefore, stored data is automatically aligned on a word boundary.

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

- Lower bits are masked to 0 and address is generated (when misaligned access is disabled)
- Lower bits are not masked and address is generated (when misaligned access is enabled)
(when misaligned access is enabled for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2)

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

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

[For V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2]
Depending on the resource to be accessed (internal ROM, internal RAM, on-chip peripheral I/O, external memory), the bus cycle may be switched (this will not occur if the same resource is accessed).

[For NB85E, NB85ET, NU85E, or NU85ET]
The bus cycle sequence for accessing the different resources connected to each bus (VFB, VDB, VSB, NPB, instruction cache bus, data cache bus) may be switched (this will not occur if the same bus is accessed).

<Special instruction>

Store contents of system register

STSR

Store Contents of System Register

Instruction format STSR regID, reg2

Operation GR [reg2] ← SR [regID]

Format Format IX

Opcode

15 0	31 16
rrrrr111111RRRRR	0000000001000000

Flag

CY –
OV –
S –
Z –
SAT –

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

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

<Logical operation instruction>

SUB	**Subtract** **Subtract**

Instruction format SUB reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr001101RRRRR
```

Flag

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

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

<Logical operation instruction>

Subtract reverse

SUBR

Subtract Reverse

Instruction format SUBR reg1, reg2

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

Format Format I

Opcode

```
15             0
rrrrr001100RRRRR
```

Flag

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

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

<Special instruction>

Jump with table look up

SWITCH

Jump with Table Look Up

Instruction format SWITCH reg1

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

Format Format I

Opcode

15 0
00000000010RRRRR

Flag

CY –
OV –
S –
Z –
SAT –

Explanation

<1> Adds the table entry address (address following SWITCH instruction) and data of general register reg1 logically shifted left by 1, and generates 32-bit table entry address.
<2> Loads half-word data pointed by address generated in <1>.
<3> Sign-extends the loaded half-word data to word length, and adds the table entry address after logically shifts it left by 1 bit (next address following SWITCH instruction) to generate a 32-bit target address.
<4> Then jumps to the target address generated in <3>.

<Logical operation instruction>

SXB	**Sign extend byte** **Sign Extend Byte**

Instruction format SXB reg1

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

Format Format I

Opcode

```
15               0
0000000000101RRRRR
```

Flag

CY –
OV –
S –
Z –
SAT –

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

<Logical operation instruction>

Sign extend half-word

SXH

Sign Extend Half-word

Instruction format SXH reg1

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

Format Format I

Opcode

```
15               0
0000000000111RRRRR
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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

<Special instruction>

TRAP

Trap

Trap

Instruction format TRAP vector

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

Format Format X

Opcode

15 0	31 16
00000111111iiiii	0000000100000000

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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 (sets EP and ID flags to 1); jumps to the handler address corresponding to the trap vector (00H to 1FH) specified by vector, and starts exception processing.
The flags of PSW other than EP and ID flags are not affected.
The restore PC is the address of the instruction following the TRAP instruction.

<Logical operation instruction>

TST

Test

Test

Instruction format TST reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr001011RRRRR
```

Flag

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

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.

<Bit manipulation instruction>

TST1

Test bit

Test Bit

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

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

Format (1) Format VIII
(2) Format IX

Opcode

(1)

15 0	31 16
11bbb111110RRRRR	dddddddddddddddd

(2)

15 0	31 16
rrrrr111111RRRRR	0000000011100110

Flag

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

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. Performs the test on the bit, specified by the 3-bit bit number, at the byte data location referenced by the generated address. If the specified bit is 0, the Z flag of PSW is set to 1; if the bit is 1, the Z flag is cleared to 0. The byte data, including the specified bit, is not affected.

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

<Logical operation instruction>

Exclusive OR

XOR

Exclusive Or

Instruction format XOR reg1, reg2

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

Format Format I

Opcode

```
15              0
rrrrr001001RRRRR
```

Flag

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

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

<Logical operation instruction>

Exclusive OR immediate (16-bit)

XORI

Exclusive Or Immediate

Instruction format XORI imm16, reg1, reg2

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

Format Format VI

Opcode

15 0	31 16
rrrrr110101RRRRR	iiiiiiiiiiiiiiii

Flag

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

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 to general register reg2. The data of general register reg1 is not affected.

<Logical operation instruction>

ZXB

Zero extend byte

Zero Extend Byte

Instruction format ZXB reg1

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

Format Format I

Opcode

```
15              0
0000000000100RRRRR
```

Flag

CY	–
OV	–
S	–
Z	–
SAT	–

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

<Logical operation instruction>

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

Instruction format ZXH reg1

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

Format Format I

Opcode

```
15                0
0000000000110RRRRR
```

Flag

CY –
OV –
S –
Z –
SAT –

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

5.4 Number of Instruction Execution Clock Cycles

A list of the number of instruction execution clocks when the internal ROM or internal RAM is used is shown below. The number of instruction execution clock cycles differ depending on the combination of instructions. For details, see **CHAPTER 8 PIPELINE**.

The number of instruction execution clock cycles differ in the case of the NB85E, NB85ET, NU85E, NU85ET and the V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2. Table 5-6 shows the number of instruction execution clock cycles in the case of the NB85E, NB85ET, NU85E, NU85ET and Table 5-7 shows the case of the V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2.

Table 5-6. List of Number of Instruction Execution Clock Cycles (NB85E, NB85ET, NU85E, and NU85ET) (1/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Load instructions	LD.B	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.H	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.W	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.BU	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.HU	disp16 [reg1] , reg2	4	1	1	**Note 1**
	SLD.B	disp7 [ep] , reg2	2	1	1	**Note 2**
	SLD.BU	disp4 [ep] , reg2	2	1	1	**Note 2**
	SLD.H	disp8 [ep] , reg2	2	1	1	**Note 2**
	SLD.HU	disp5 [ep] , reg2	2	1	1	**Note 2**
	SLD.W	disp8 [ep] , reg2	2	1	1	**Note 2**
Store instructions	ST.B	reg2, disp16 [reg1]	4	1	1	1
	ST.H	reg2, disp16 [reg1]	4	1	1	1
	ST.W	reg2, disp16 [reg1]	4	1	1	1
	SST.B	reg2, disp7 [ep]	2	1	1	1
	SST.H	reg2, disp8 [ep]	2	1	1	1
	SST.W	reg2, disp8 [ep]	2	1	1	1
Multiply instructions	MUL	reg1, reg2, reg3	4	1	2[Note 3]	2
	MUL	imm9, reg2, reg3	4	1	2[Note 3]	2
	MULH	reg1, reg2	2	1	1	2
	MULH	imm5, reg2	2	1	1	2
	MULHI	imm16, reg1, reg2	4	1	1	2
	MULU	reg1, reg2, reg3	4	1	2[Note 3]	2
	MULU	imm9, reg2, reg3	4	1	2[Note 3]	2
Arithmetic operation instructions	ADD	reg1, reg2	2	1	1	1
	ADD	imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
	CMOV	cccc, imm5, reg2, reg3	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
	CMP	imm5, reg2	2	1	1	1

Table 5-6. List of Number of Instruction Execution Clock Cycles (NB85E, NB85ET, NU85E, and NU85ET) (2/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Arithmetic operation instructions	DIV	reg1, reg2, reg3	4	35	35	35
	DIVH	reg1, reg2	2	35	35	35
	DIVH	reg1, reg2, reg3	4	35	35	35
	DIVHU	reg1, reg2, reg3	4	34	34	34
	DIVU	reg1, reg2, reg3	4	34	34	34
	MOV	reg1, reg2	2	1	1	1
	MOV	imm5, reg2	2	1	1	1
	MOV	imm32, reg1	6	2	2	2
	MOVEA	imm16, reg1, reg2	4	1	1	1
	MOVHI	imm16, reg1, reg2	4	1	1	1
	SASF	cccc, reg2	4	1	1	1
	SETF	cccc, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
Saturated operation instructions	SATADD	reg1, reg2	2	1	1	1
	SATADD	imm5, reg2	2	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
	SATSUBR	reg1, reg2	2	1	1	1
Logical operation instructions	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	BSH	reg2, reg3	4	1	1	1
	BSW	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
	SAR	imm5, reg2	2	1	1	1
	SHL	reg1, reg2	4	1	1	1
	SHL	imm5, reg2	2	1	1	1
	SHR	reg1, reg2	4	1	1	1
	SHR	imm5, reg2	2	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
	XORI	imm16, reg1, reg2	4	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1

Table 5-6. List of Number of Instruction Execution Clock Cycles (NB85E, NB85ET, NU85E, and NU85ET) (3/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Branch instructions	Bcond	disp9 (When condition is satisfied)	2	2[Note 4]	2[Note 4]	2[Note 4]
		disp9 (When condition is not satisfied)	2	1	1	1
	JARL	disp22, reg2	4	2	2	2
	JMP	[reg1]	2	3	3	3
	JR	disp22	4	2	2	2
Bit manipulation instructions	CLR1	bit#3, disp16 [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	CLR1	reg2, [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	NOT1	bit#3, disp16 [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	NOT1	reg2, [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	SET1	bit#3, disp16 [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	SET1	reg2, [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	TST1	bit#3, disp16 [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
	TST1	reg2, [reg1]	4	3[Note 5]	3[Note 5]	3[Note 5]
Special instructions	CALLT	imm6	2	4	4	4
	CTRET	–	4	3	3	3
	DI	–	4	1	1	1
	DISPOSE	imm5, list12	4	n+1[Note 6]	n+1[Note 6]	n+1[Note 6]
	DISPOSE	imm5, list12, [reg1]	4	n+3[Note 6]	n+3[Note 6]	n+3[Note 6]
	EI	–	4	1	1	1
	HALT	–	4	1	1	1
	LDSR	reg2, regID	4	1	1	1
	NOP	–	2	1	1	1
	PREPARE	list12, imm5	4	n+1[Note 6]	n+1[Note 6]	n+1[Note 6]
	PREPARE	list12, imm5, sp	4	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]
	PREPARE	list12, imm5, imm16	6	n+2[Note 6]	n+2[Note 6]	n+2[Note 6]
	PREPARE	list12, imm5, imm32	8	n+3[Note 6]	n+3[Note 6]	n+3[Note 6]
	RETI	–	4	3	3	3
	STSR	regID, reg2	4	1	1	1
	SWITCH	reg1	2	5	5	5
	TRAP	vector	4	3	3	3
Debug function instructions[Note 7]	DBRET	–	4	3	3	3
	DBTRAP	–	2	3	3	3
Undefined instruction code			4	3	3	3

Notes 1. Depends on the number of wait states (2 if no wait states).

2. Depends on the number of wait states (1 if no wait states).

3. 1 clock shortened if reg2 = reg3 (lower 32 bits of results are not written to register) or reg3 = r0 (higher 32 bits of results are not written to register).

★ **4.** 2 if there is an instruction rewriting the PSW contents immediately before.

5. In case of no wait states (3 + number of read access wait states).

★ **6.** n is the total number of cycles to load registers in list12 (Depends on the number of wait states, n is the number of registers in list12 if no wait states. The operation when n = 0 is the same as when n = 1).

★ **7.** The NB85E and NB85ET do not support instructions for the debug function.

Remarks 1. Operand convention

Symbol	Meaning
reg1	General register (used as source register)
reg2	General register (mainly used as destination register. Some are also used as source registers.)
reg3	General register (mainly used as remainder of division results or higher 32 bits of multiply results)
bit#3	3-bit data for bit number specification
imm×	×-bit immediate data
disp×	×-bit displacement data
regID	System register number
vector	5-bit data for trap vector (00H to 1FH) specification
cccc	4-bit data condition code specification
sp	Stack pointer (r3)
ep	Element pointer (r30)
list×	List of registers (× is a maximum number of registers)

2. Execution clock convention

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

Table 5-7. List of Number of Instruction Execution Clock Cycles (V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2) (1/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Load instructions	LD.B	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.H	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.W	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.BU	disp16 [reg1] , reg2	4	1	1	**Note 1**
	LD.HU	disp16 [reg1] , reg2	4	1	1	**Note 1**
	SLD.B	disp7 [ep] , reg2	2	1	1	**Note 2**
	SLD.BU	disp4 [ep] , reg2	2	1	1	**Note 2**
	SLD.H	disp8 [ep] , reg2	2	1	1	**Note 2**
	SLD.HU	disp5 [ep] , reg2	2	1	1	**Note 2**
	SLD.W	disp8 [ep] , reg2	2	1	1	**Note 2**
Store instructions	ST.B	reg2, disp16 [reg1]	4	1	1	1
	ST.H	reg2, disp16 [reg1]	4	1	1	1
	ST.W	reg2, disp16 [reg1]	4	1	1	1
	SST.B	reg2, disp7 [ep]	2	1	1	1
	SST.H	reg2, disp8 [ep]	2	1	1	1
	SST.W	reg2, disp8 [ep]	2	1	1	1
Multiply instructions	MUL	reg1, reg2, reg3	4	1	2[Note 3]	2
	MUL	imm9, reg2, reg3	4	1	2[Note 3]	2
	MULH	reg1, reg2	2	1	1	2
	MULH	imm5, reg2	2	1	1	2
	MULHI	imm16, reg1, reg2	4	1	1	2
	MULU	reg1, reg2, reg3	4	1	2[Note 3]	2
	MULU	imm9, reg2, reg3	4	1	2[Note 3]	2
Arithmetic operation instructions	ADD	reg1, reg2	2	1	1	1
	ADD	imm5, reg2	2	1	1	1
	ADDI	imm16, reg1, reg2	4	1	1	1
	CMOV	cccc, reg1, reg2, reg3	4	1	1	1
	CMOV	cccc, imm5, reg2, reg3	4	1	1	1
	CMP	reg1, reg2	2	1	1	1
	CMP	imm5, reg2	2	1	1	1
	DIV	reg1, reg2, reg3	4	35	35	35
	DIVH	reg1, reg2	2	35	35	35
	DIVH	reg1, reg2, reg3	4	35	35	35
	DIVHU	reg1, reg2, reg3	4	34	34	34
	DIVU	reg1, reg2, reg3	4	34	34	34
	MOV	reg1, reg2	2	1	1	1
	MOV	imm5, reg2	2	1	1	1
	MOV	imm32, reg1	6	2	2	2
	MOVEA	imm16, reg1, reg2	4	1	1	1

Table 5-7. List of Number of Instruction Execution Clock Cycles (V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2) (2/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Arithmetic operation instructions	MOVHI	imm16, reg1, reg2	4	1	1	1
	SASF	cccc, reg2	4	1	1	1
	SETF	cccc, reg2	4	1	1	1
	SUB	reg1, reg2	2	1	1	1
	SUBR	reg1, reg2	2	1	1	1
Saturated operation instructions	SATADD	reg1, reg2	2	1	1	1
	SATADD	imm5, reg2	2	1	1	1
	SATSUB	reg1, reg2	2	1	1	1
	SATSUBI	imm16, reg1, reg2	4	1	1	1
	SATSUBR	reg1, reg2	2	1	1	1
Logical operation instructions	AND	reg1, reg2	2	1	1	1
	ANDI	imm16, reg1, reg2	4	1	1	1
	BSH	reg2, reg3	4	1	1	1
	BSW	reg2, reg3	4	1	1	1
	HSW	reg2, reg3	4	1	1	1
	NOT	reg1, reg2	2	1	1	1
	OR	reg1, reg2	2	1	1	1
	ORI	imm16, reg1, reg2	4	1	1	1
	SAR	reg1, reg2	4	1	1	1
	SAR	imm5, reg2	2	1	1	1
	SHL	reg1, reg2	4	1	1	1
	SHL	imm5, reg2	2	1	1	1
	SHR	reg1, reg2	4	1	1	1
	SHR	imm5, reg2	2	1	1	1
	SXB	reg1	2	1	1	1
	SXH	reg1	2	1	1	1
	TST	reg1, reg2	2	1	1	1
	XOR	reg1, reg2	2	1	1	1
	XORI	imm16, reg1, reg2	4	1	1	1
	ZXB	reg1	2	1	1	1
	ZXH	reg1	2	1	1	1
Branch instructions	Bcond	disp9 (When condition is satisfied)	2	3 Note 4	3 Note 4	3 Note 4
		disp9 (When condition is not satisfied)	2	1	1	1
	JARL	disp22, reg2	4	3	3	3
	JMP	[reg1]	2	4	4	4
	JR	disp22	4	3	3	3

Table 5-7. List of Number of Instruction Execution Clock Cycles (V850E/MA1, V850E/MA2, V850E/IA1, and V850E/IA2) (3/3)

Type of Instruction	Mnemonic	Operand	Byte	Number of Execution Clocks		
				i	r	l
Bit manipulation instructions	CLR1	bit#3, disp16 [reg1]	4	3Note 5	3Note 5	3Note 5
	CLR1	reg2, [reg1]	4	3Note 5	3Note 5	3Note 5
	NOT1	bit#3, disp16 [reg1]	4	3Note 5	3Note 5	3Note 5
	NOT1	reg2, [reg1]	4	3Note 5	3Note 5	3Note 5
	SET1	bit#3, disp16 [reg1]	4	3Note 5	3Note 5	3Note 5
	SET1	reg2, [reg1]	4	3Note 5	3Note 5	3Note 5
	TST1	bit#3, disp16 [reg1]	4	3Note 5	3Note 5	3Note 5
	TST1	reg2, [reg1]	4	3Note 5	3Note 5	3Note 5
Special instructions	CALLT	imm6	2	5	5	5
	CTRET	–	4	4	4	4
	DI	–	4	1	1	1
	DISPOSE	imm5, list12	4	n+1Note 6	n+1Note 6	n+1Note 6
	DISPOSE	imm5, list12, [reg1]	4	n+3Note 6	n+3Note 6	n+3Note 6
	EI	–	4	1	1	1
	HALT	–	4	1	1	1
	LDSR	reg2, regID	4	1	1	1
	NOP	–	2	1	1	1
	PREPARE	list12, imm5	4	n+1Note 6	n+1Note 6	n+1Note 6
	PREPARE	list12, imm5, sp	4	n+2Note 6	n+2Note 6	n+2Note 6
	PREPARE	list12, imm5, imm16	6	n+2Note 6	n+2Note 6	n+2Note 6
	PREPARE	list12, imm5, imm32	8	n+3Note 6	n+3Note 6	n+3Note 6
	RETI	–	4	4	4	4
	STSR	regID, reg2	4	1	1	1
	SWITCH	reg1	2	5	5	5
	TRAP	vector	4	4	4	4
Debug function instructions	DBRET	–	4	4	4	4
	DBTRAP	–	2	4	4	4
Undefined instruction code			4	3	3	3

Notes 1. Depends on the number of wait states (2 if no wait states).

2. Depends on the number of wait states (1 if no wait states).

3. 1 clock shortened if reg2 = reg3 (lower 32 bits of results are not written to register) or reg3 = r0 (higher 32 bits of results are not written to register).

★ **4.** 2 if there is an instruction rewriting the PSW contents immediately before.

5. In case of no wait states (3 + number of read access wait states).

6. n is the total number of cycles to load registers in list12 (Depends on the number of wait states, n is ★ the number of registers in list12 if no wait states. The operation when n = 0 is the same as when n = 1).

Remarks 1. Operand convention

Symbol	Meaning
reg1	General register (used as source register)
reg2	General register (mainly used as destination register. Some are also used as source registers.)
reg3	General register (mainly used as remainder of division results or higher 32 bits of multiply results)
bit#3	3-bit data for bit number specification
imm×	×-bit immediate data
disp×	×-bit displacement data
regID	System register number
vector	5-bit data for trap vector (00H to 1FH) specification
cccc	4-bit data condition code specification
sp	Stack pointer (r3)
ep	Element pointer (r30)
list×	List of registers (× is a maximum number of registers)

2. Execution clock convention

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

Interrupts are events that occur independently of the program execution and are divided into two types: maskable interrupts and non-maskable interrupts (NMI). In contrast, exceptions are events whose occurrence is dependent on the program execution and are divided into three types: software exception, exception trap, and debug trap.

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

To restore execution from interrupt or software exception processing, use the RETI instruction. To restore execution from exception trap or debug trap, use the DBRET instruction. Read the restore PC and restore PSW from the status saving register, and transfer control to the restore PC.

Table 6-1. Interrupt/Exception Codes

Interrupt/Exception Source			Classification	Exception Code	Handler Address	Restore PC
Name		Trigger				
Non-maskable interrupt (NMI)[Note 1]		NMI0 input	Interrupt	0010H	00000010H	next PC[Note 2]
		NMI1 input	Interrupt	0020H	00000020H	next PC[Notes 2, 3]
		NMI2 input[Note 4]	Interrupt	0030H	00000030H	next PC[Notes 2, 3]
Maskable interrupt		**Note 5**	Interrupt	**Note 5**	**Note 6**	next PC[Note 2]
Software exception	TRAP0n (n = 0 to FH)	TRAP instruction	Exception	004nH	00000040H	next PC
	TRAP1n (n = 0 to FH)	TRAP instruction	Exception	005nH	00000050H	next PC
Exception trap (ILGOP)		Illegal instruction code	Exception	0060H	00000060H	next PC[Note 7]
Debug trap[Note 8]		DBTRAP instruction[Note 8]	Exception	0060H	00000060H	next PC

Notes 1. NMI0 input is the only generation source for the V850E/MA1, V850E/MA2, V850E/IA1, or V850E/IA2.

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

- Load instructions (SLD.B, SLD.BU, SLD.H, SLD.HU, SLD.W), divide instructions (DIV, DIVH, DIVU, DIVHU)
- PREPARE, DISPOSE instruction (only if an interrupt is generated before the stack pointer is updated)

3. The PC cannot be restored by the RETI instruction. Perform a system reset after interrupt servicing.

4. Acknowledged even if the NP flag of PSW is set to 1.

5. Differs depending on the type of the interrupts.

6. Higher 16 bits are 0000H and lower 16 bits are the same value as the exception code.

7. The execution address of the illegal instruction is obtained by “Restore PC – 4”.

★ **8.** Not supported in the NB85E and NB85ET

Remark Restore PC: PC value saved to the EIPC or FEPC when interrupt/exception processing is started
next PC: PC value that starts processing after interrupt/exception processing

6.1 Interrupt Servicing

6.1.1 Maskable interrupt

The maskable interrupt can be masked by the interrupt control register of the interrupt controller (INTC).

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 interrupt request input (INT input), the CPU performs the following steps, and transfers control to the handler routine.

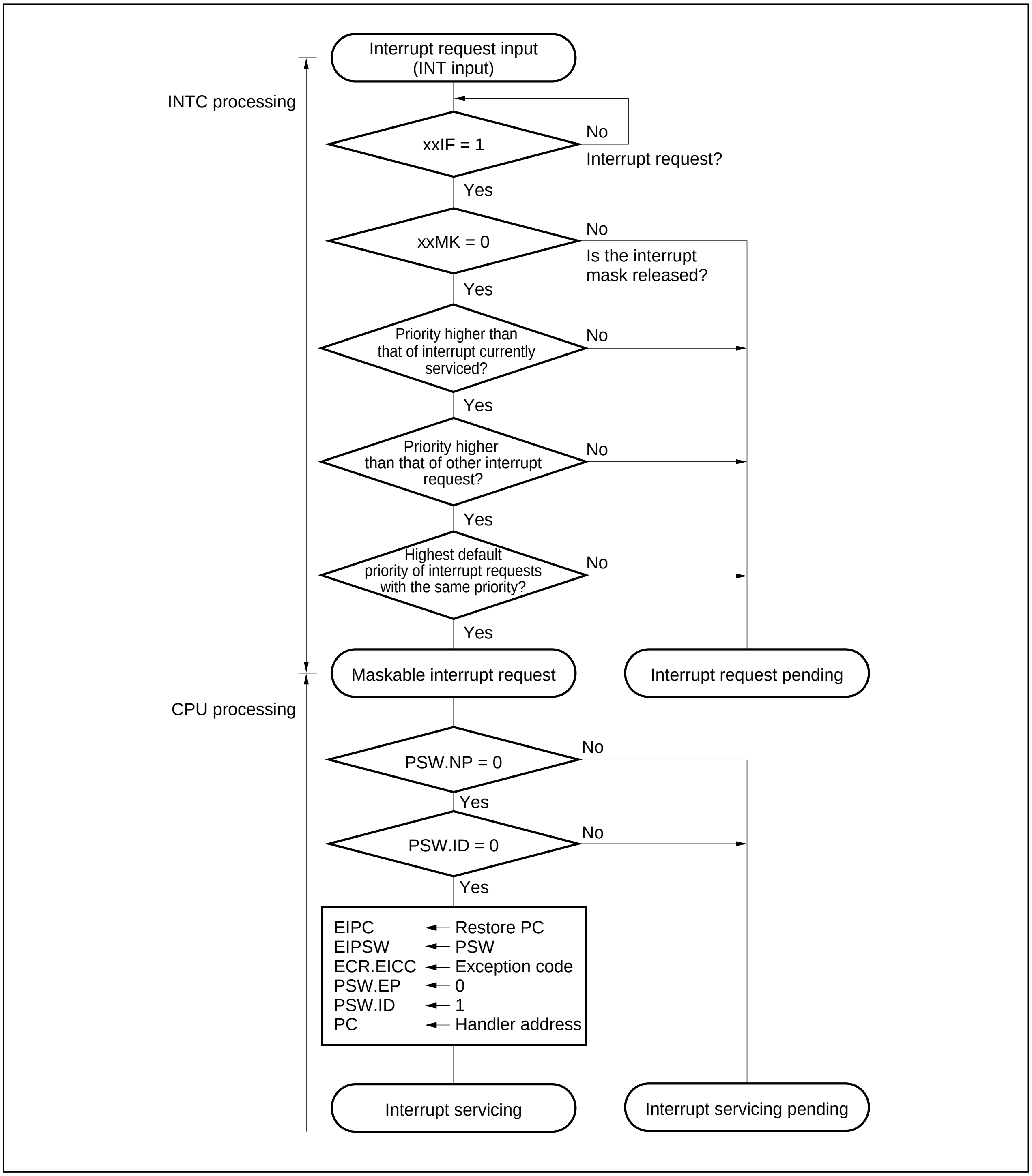
(1) Saves restore PC to EIPC.
(2) Saves current PSW to EIPSW.
(3) Writes exception code to lower half-word of ECR (EICC).
(4) Sets ID flag of PSW to 1 and clears EP flag to 0.
(5) Sets handler address for each interrupt to PC and transfers control.

The EIPC and EIPSW are used as the status saving registers. INT inputs are held pending in the interrupt controller (INTC) when one of the following two conditions occur: when the INT input is masked by its interrupt controller, or when an interrupt service routine is currently being executed (when the NP flag of the PSW is 1 or when the ID flag of the PSW is 1). Interrupts are enabled by clearing the mask condition or by setting the NP and ID flags of the PSW to 0 with the LDSR instruction, which will be enabling new maskable interrupt servicing by a pending INT input.

The EIPC and EIPSW registers must be saved by program to enable nesting of interrupts because there is only one set of EIPC and EIPSW is provided.

Maskable interrupt servicing format is shown below.

Figure 6-1. Maskable Interrupt Servicing Format



6.1.2 Non-maskable interrupt

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

When the non-maskable interrupt is generated, the CPU performs the following steps, and transfers control to the handler routine.

(1) Saves restore PC to FEPC.
(2) Saves current PSW to FEPSW.
(3) Writes exception code (0010H) to higher half-word of ECR (FECC).
(4) Sets NP and ID flags of PSW to 1 and clears EP flag to 0.
(5) Sets handler address for the non-maskable interrupt to PC and transfers control.

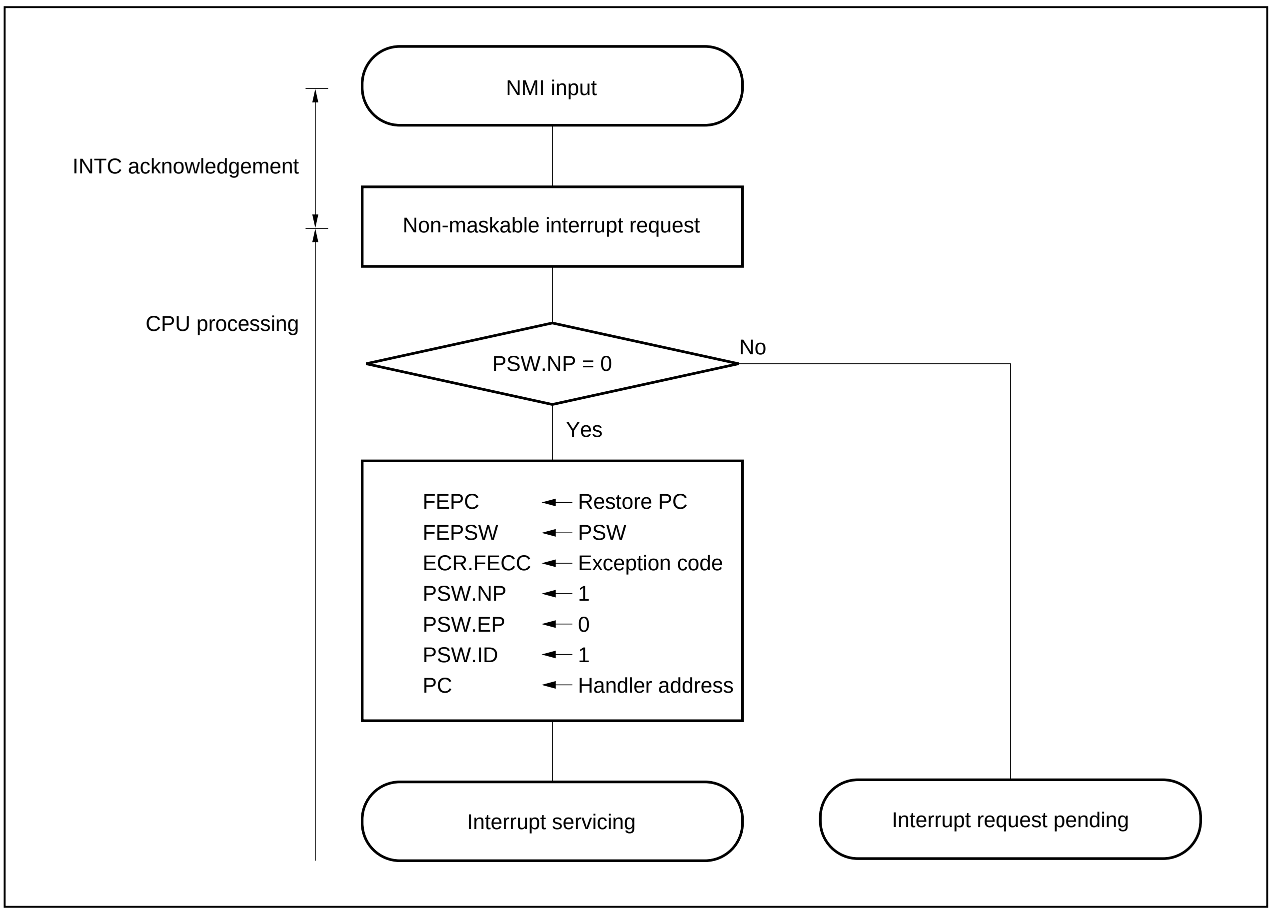
The FEPC and FEPSW are used as the status saving registers.

Non-maskable interrupts are held pending in the interrupt controller when another non-maskable interrupt is currently being executed (when the NP flag of the PSW is 1). Non-maskable interrupts are enabled by setting the NP flag of the PSW to 0 with the RETI and LDSR instructions, which will be enabling new non-maskable interrupt servicing by a pending non-maskable interrupt request.

In the case of NB85E, NB85ET, NU85E, or NU85ET, only when the NMI2 is generated during the interrupt servicing of NMI0 and NMI1, NMI2 servicing is executed regardless of the value of NP flag.

Non-maskable interrupt servicing format is shown below.

Figure 6-2. Non-Maskable Interrupt Servicing Format



6.2 Exception Processing

6.2.1 Software exception

A software exception is generated when the TRAP instruction is executed 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.

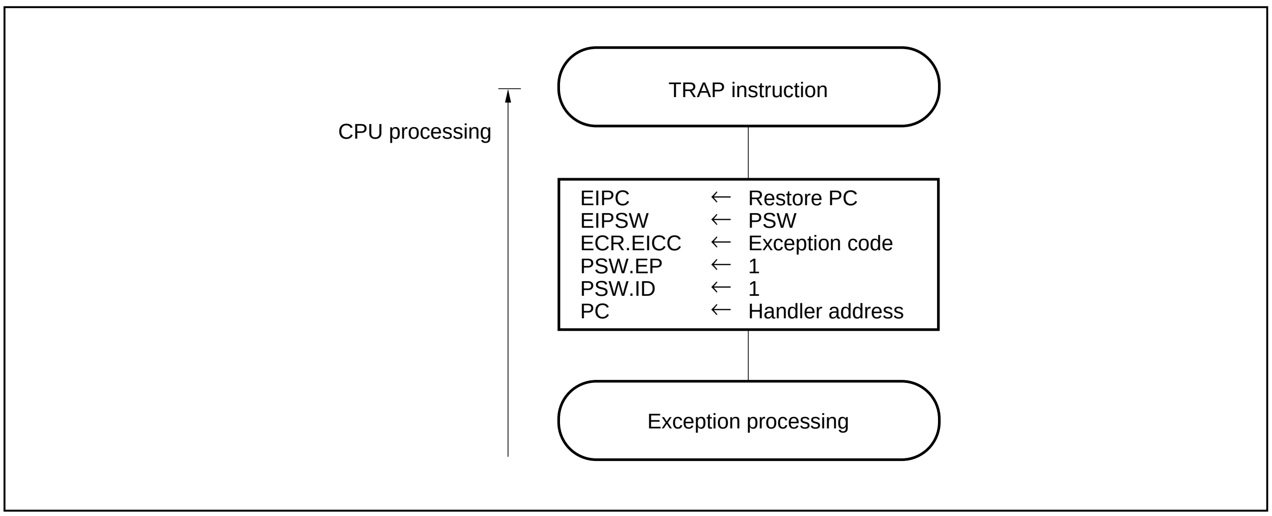
(3) Writes exception code to lower 16 bits (EICC) of ECR (interrupt source).

(4) Sets EP and ID flags of PSW to 1.

(5) Sets handler address (00000040H or 00000050H) for software exception to PC and transfers control.

Software exception processing format is shown below.

Figure 6-3. Software Exception Processing Format

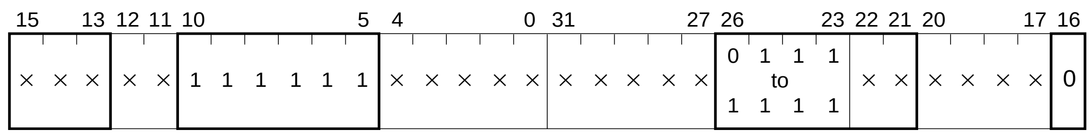


6.2.2 Exception trap

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

An illegal opcode instruction has an instruction code with an opcode (bits 10 through 5) of 111111B and a sub-opcode (bits 26 through 23) of 0111B through 1111B and a sub-opcode (bit 16) of 0B. When this kind of an illegal opcode instruction is executed, an exception trap occurs.

Figure 6-4. Illegal Instruction Code



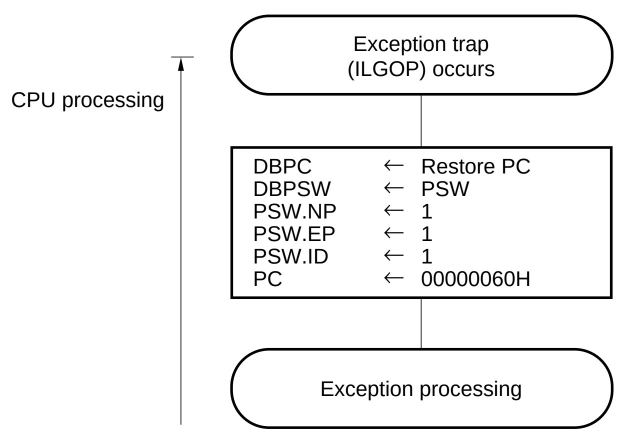
Remark ×: don't care, □: opcode/sub-opcode

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

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

Exception trap processing format is shown below.

Figure 6-5. Exception Trap Processing Format



Caution The operation when executing the instruction not defined as an instruction or illegal instruction is not guaranteed.

Remark The execution address of the illegal instruction is obtained by "Restore PC – 4".

6.2.3 Debug trap

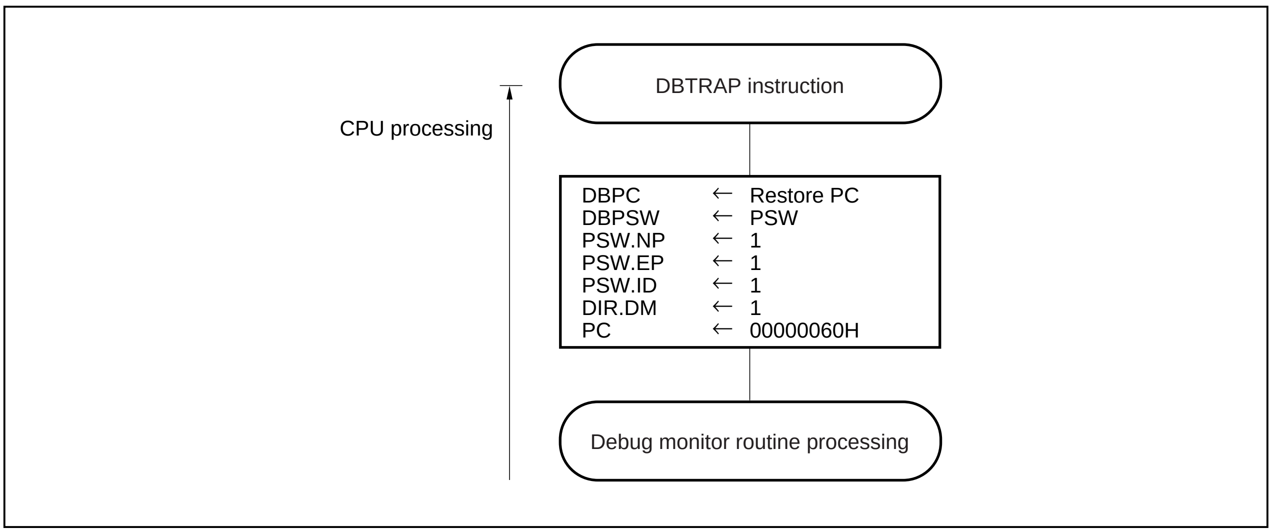
A debug trap is generated when the DBTRAP instruction is executed and is always acknowledged.
If a debug trap occurs, the CPU performs the following steps.

(1) Saves restore PC to DBPC.
(2) Saves current PSW to DBPSW.
(3) Sets NP, EP, and ID flags of PSW to 1.
(4) Sets DM flag of DIR to 1 when a debug trap occurs.
(5) Sets handler address (00000060H) for debug trap to PC and transfers control.

★ **Caution The NB85E and NB85ET do not support debug trap.**

Debug trap processing format is shown below.

Figure 6-6. Debug Trap Processing Format



6.3 Restoring from Interrupt/Exception Processing

6.3.1 Restoring from interrupt and software exception

All restoration from interrupt servicing and software exception is executed by the RETI instruction.

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

(1) 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. 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.

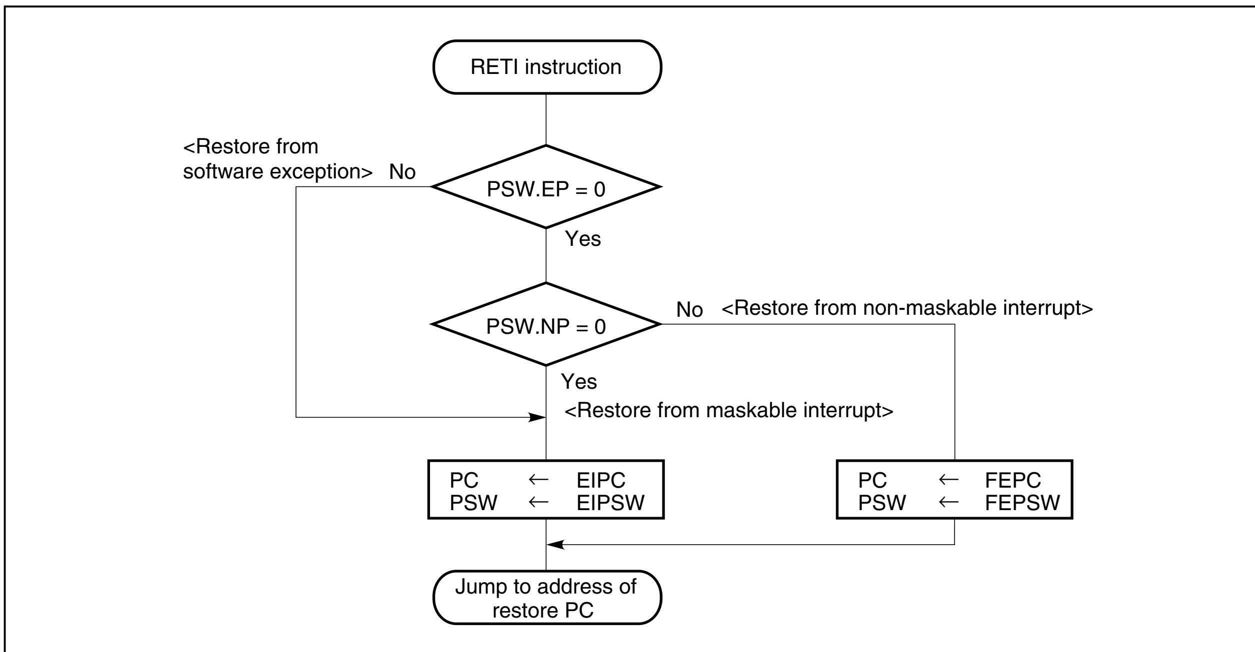
When execution has returned from each interrupt servicing, the NP and EP flags of the PSW must be set to the following values by using the LDSR instruction immediately before the RETI instruction, in order to restore the PC and PSW normally:

- To restore from non-maskable interrupt servicing[Note]: NP flag of PSW = 1, EP flag = 0
- To restore from maskable interrupt servicing: NP flag of PSW = 0, EP flag = 0
- To restore from exception processing: EP flag of PSW = 1

Note In the case of NB85E, NB85ET, NU85E, or NU85ET, NMI1 and NMI2 cannot be restored by the RETI instruction. Execute the system reset after the interrupt servicing. NMI2 can be acknowledged even if the NP flag of PSW is set to 1.

Restoration from interrupt/exception processing format is shown below.

Figure 6-7. Restoration from Interrupt/Software Exception Processing Format



6.3.2 Restoring from exception trap and debug trap

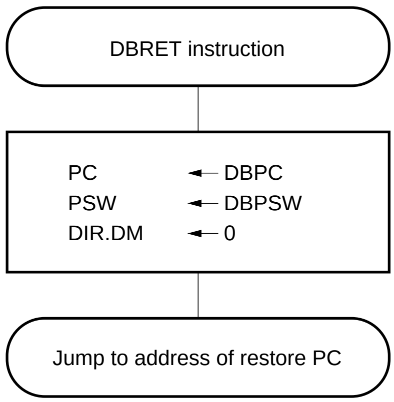
Restoration from exception trap and debug trap is executed by the DBRET instruction.

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

(1) The restore PC and PSW are read from the DBPC and DBPSW.
(2) Control is transferred to the address of the restored PC and PSW.
(3) If restored from exception trap or debug trap, DM flag of DIR is cleared to 0.

Restoration from exception trap/debug trap processing format is shown below.

Figure 6-8. Restoration from Exception Trap/Debug Trap Processing Format



CHAPTER 7 RESET

7.1 Register Status after Reset

When a low-level signal is input to the reset pin, the system is reset, and program registers and system registers are set in the status shown in Table 7-1. When the reset signal goes high, the reset status is cleared, and program execution begins. If necessary, initialize the contents of each register by program control.

Table 7-1. Register Status after Reset

Register		Status after Reset (Initial Value)
Program registers	General register (r0)	00000000H (Fixed)
	General register (r1 to r31)	Undefined
	Program counter (PC)	00000000H
System registers	Interrupt status saving register (EIPC)	0xxxxxxxH
	Interrupt status saving register (EIPSW)	00000xxxH
	NMI status saving register (FEPC)	0xxxxxxxH
	NMI status saving register (FEPSW)	00000xxxH
	Exception cause register (ECR)	00000000H
	Program status word (PSW)	00000020H
	CALLT caller status saving register (CTPC)	0xxxxxxxH
	CALLT caller status saving register (CTPSW)	00000xxxH
	Exception/debug trap status saving register (DBPC)	0xxxxxxxH
	Exception/debug trap status saving register (DBPSW)	00000xxxH
	CALLT base pointer (CTBP)	0xxxxxxxH
	Debug interface register (DIR)	00000040H
	Breakpoint control register 0 (BPC0)	00xxxxx0H
	Breakpoint control register 1 (BPC1)	00xxxxx0H
	Program ID register (ASID)	000000xxH
	Breakpoint address setting register 0 (BPAV0)	0xxxxxxxH
	Breakpoint address setting register 1 (BPAV1)	0xxxxxxxH
	Breakpoint address mask register 0 (BPAM0)	0xxxxxxxH
	Breakpoint address mask register 1 (BPAM1)	0xxxxxxxH
	Breakpoint data setting register 0 (BPDV0)	Undefined
	Breakpoint data setting register 1 (BPDV1)	Undefined
	Breakpoint data mask register 0 (BPDM0)	Undefined
	Breakpoint data mask register 1 (BPDM1)	Undefined

Remark x: Undefined

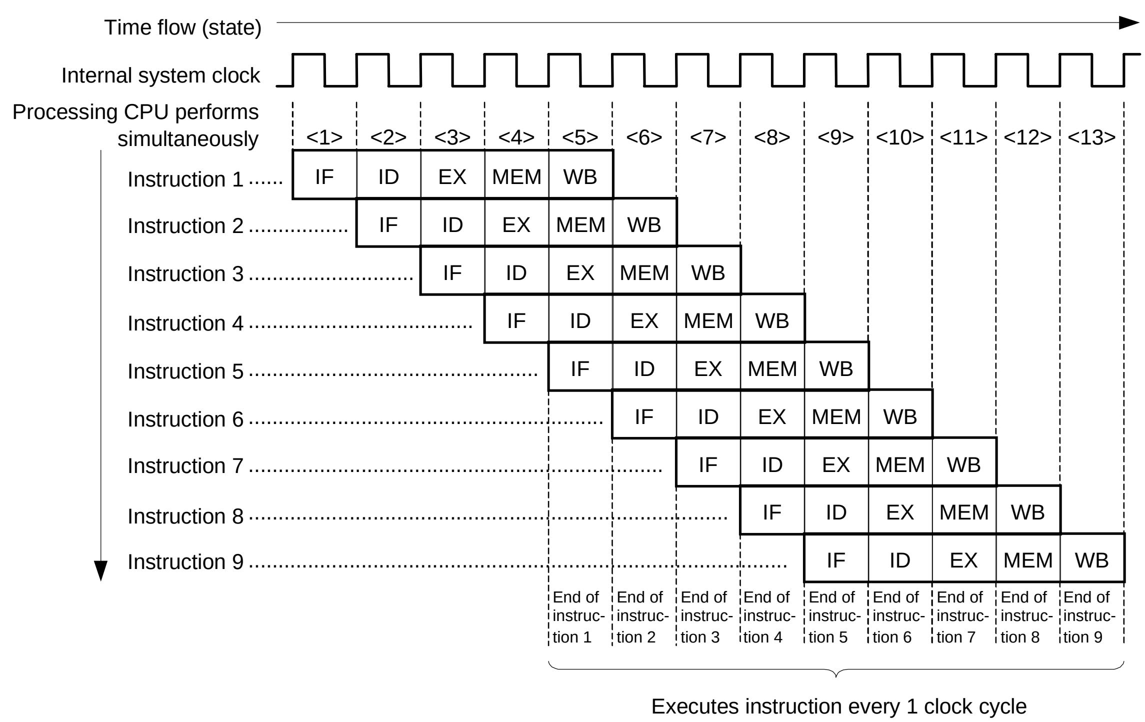
7.2 Starting Up

The CPU begins program execution from address 00000000H after it has been reset.

After reset, no immediate interrupt requests are acknowledged. To enable interrupts by program, clear the ID flag of the PSW to 0.

The V850E1 CPU is based on the RISC architecture and executes almost all the instructions in one clock cycle under control of a 5-stage pipeline. The instruction execution sequence usually consists of five stages including fetch (IF) to write back (WB) 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 9 standard instructions are executed in succession.

Figure 8-1. Example of Executing Nine Standard Instructions



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

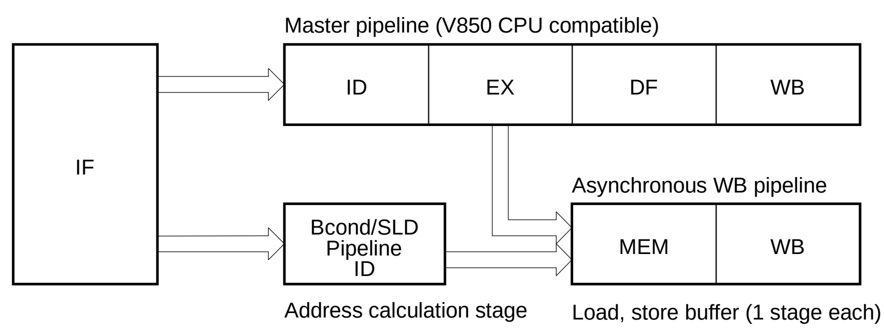
<1> through <13> in the figure above indicate the states of the CPU. In each state, write back (WB) of instruction n, memory access (MEM) of instruction n+1, execution (EX) of instruction n+2, decoding (ID) of instruction n+3, and fetching (IF) of instruction n+4 are simultaneously performed. It takes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

8.1 Features

The V850E1 CPU, by optimizing the pipeline, improves the CPI (Cycle per instruction) rate over the previous V850 CPU.

The pipeline configuration of the V850E1 CPU is shown in Figure 8-2.

Figure 8-2. Pipeline Configuration



Remark DF (data fetch): Execution data is transferred to the WB stage.

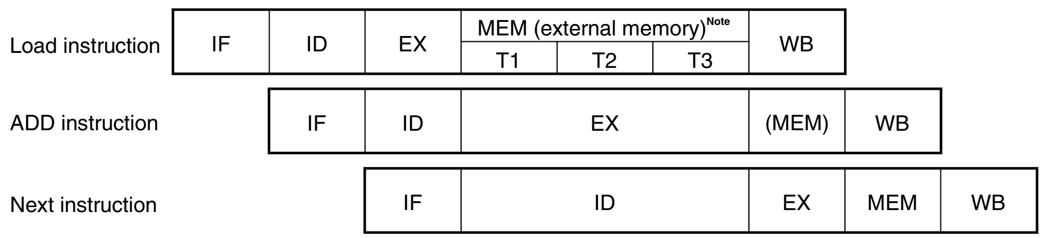
8.1.1 Non-blocking load/store

As the pipeline does not stop during external memory access, efficient processing is possible.

For example, Figure 8-3 shows a comparison of pipeline operations between the V850 CPU and the V850E1 CPU when an ADD instruction is executed after the execution of a load instruction for external memory.

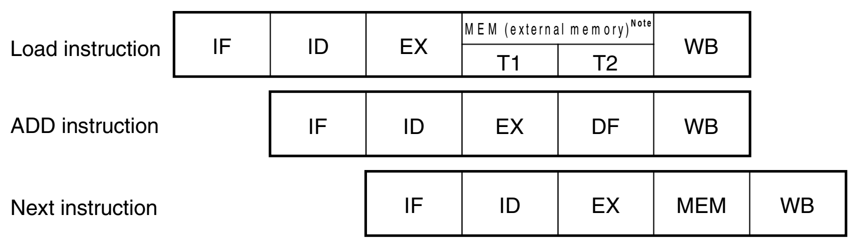
Figure 8-3. Non-Blocking Load/Store

(a) Previous version (V850 CPU): Pipeline is stopped until MEM stage is complete



Note The basic bus cycle for the external memory is 3 clocks.

(b) V850E1 CPU: Efficient pipeline processing through use of asynchronous WB pipeline



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

(1) V850 CPU

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

(2) V850E1 CPU

An asynchronous WB pipeline for the instructions that are necessary for the MEM stage is provided in addition to the master pipeline. The MEM stage of the load instruction is therefore processed on this asynchronous WB pipeline. Because the ADD instruction is processed on the master pipeline, a wait time is not generated, making it possible to execute instructions efficiently as shown in Figure 8-3.

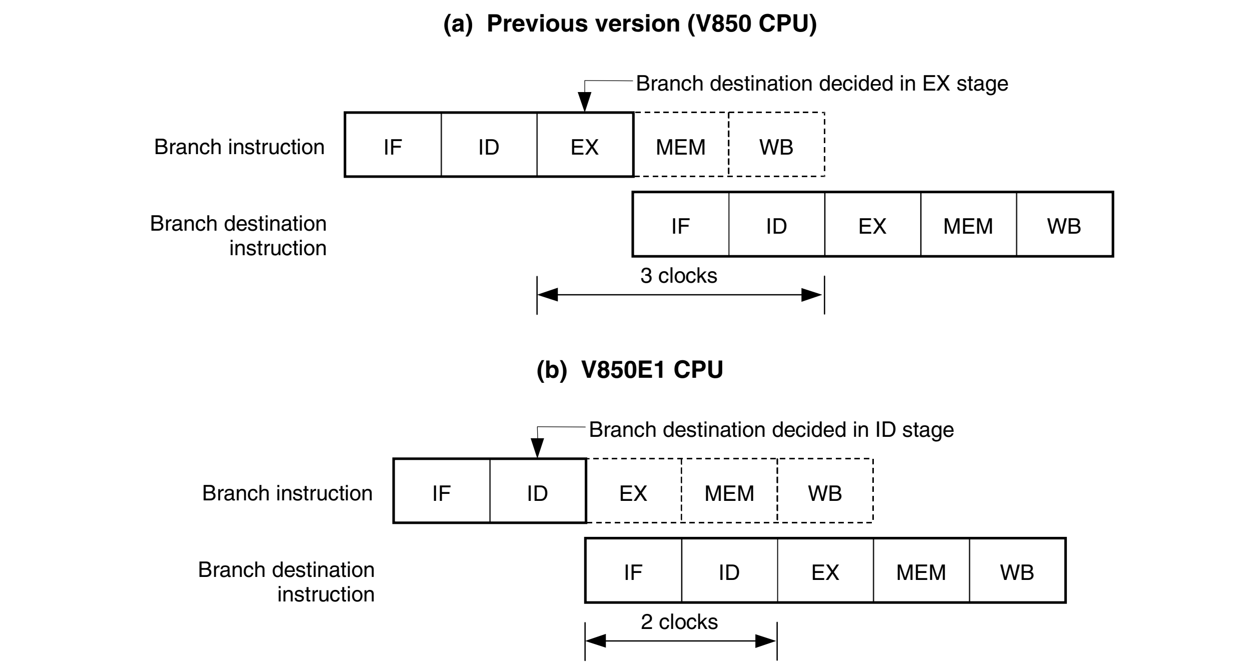
8.1.2 2-clock branch

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

In the case of the conventional V850 CPU, the branch destination of when the branch instruction is executed was decided after execution of the EX stage, but in the case of the V850E1 CPU, due to the addition of a address calculation stage for branch/SLD instruction, the branch destination is decided in the ID stage. Therefore, it is possible to fetch the branch destination instruction 1 clock faster than in the conventional V850 CPU.

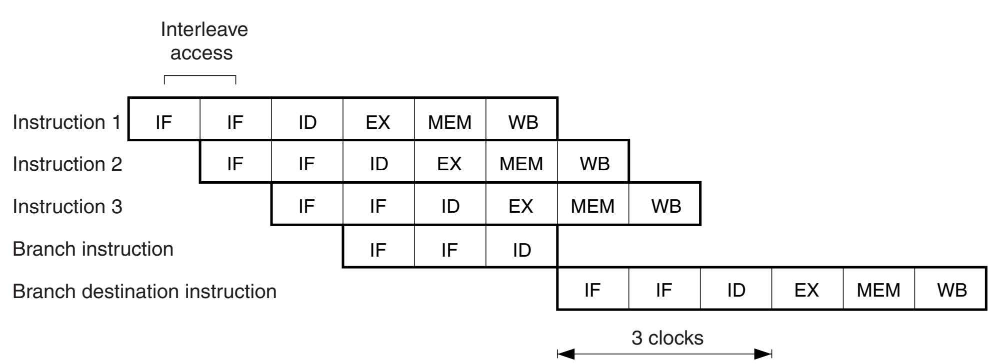
Figure 8-4 shows a comparison between the V850 CPU and the V850E1 CPU of pipeline operations with branch instructions.

Figure 8-4. Pipeline Operations with Branch Instructions



★ **Remark** The V850E/MA1, V850E/IA1, and V850E/IA2 executes interleave access to the internal flash memory or internal mask ROM. Therefore, it takes two clocks to an instruction fetch immediately after an interrupt has occurred or after a branch destination instruction has been executed. Consequently, it takes three clocks to execute the ID stage of the branch destination instruction.

★ **Example**

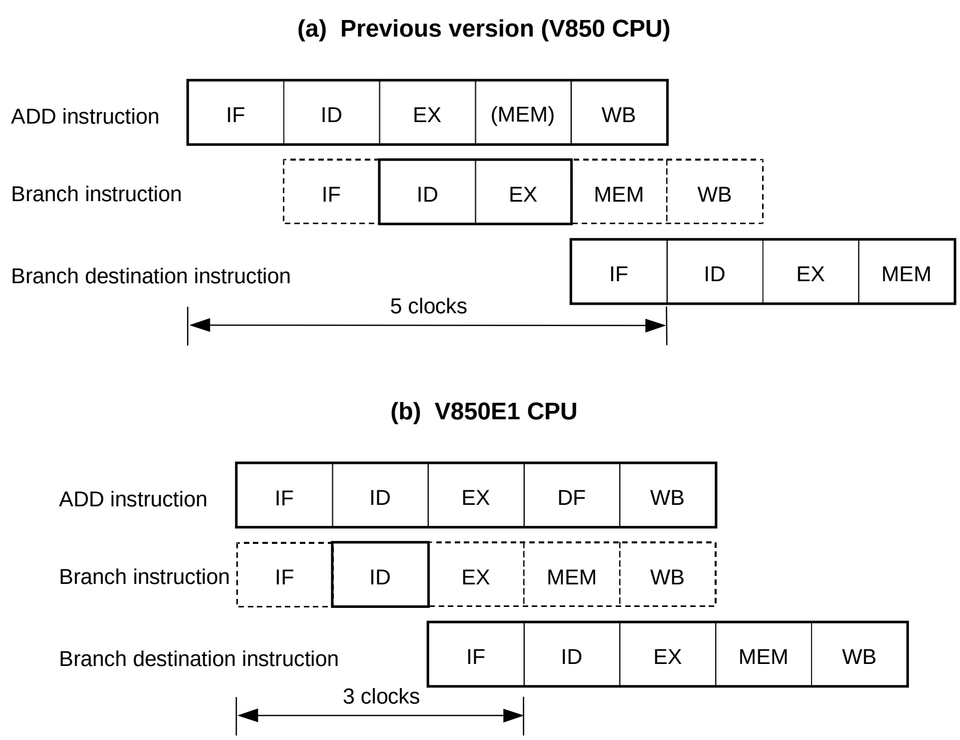


8.1.3 Efficient pipeline processing

Because the V850E1 CPU has an ID stage for branch/SLD instructions in addition to the ID stage on the master pipeline, it is possible to perform efficient pipeline processing.

Figure 8-5 shows an example of a pipeline operation where the next branch instruction was fetched in the IF stage of the ADD instruction (Instruction fetch from the ROM directly connected to the dedicated bus is performed in 32-bit units. Both ADD instructions and branch instructions in Figure 8-5 use a 16-bit format instruction).

Figure 8-5. Parallel Execution of Branch Instructions



(1) V850 CPU

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

(2) V850E1 CPU

Because V850E1 CPU has an ID stage for branch/SLD instructions in addition to the ID stage on the master pipeline, the parallel execution of the ID stage of the ADD instruction and the ID stage of the branch instruction within the same clock is possible. Therefore, it takes only 3 clocks from the branch instruction fetch start to the branch destination instruction completion.

★ **Caution Be aware that the SLD and Bcond instructions are sometimes executed at the same time as other 16-bit format instructions. For example, if the SLD and NOP instructions are executed simultaneously, the NOP instruction may keep the delay time from being generated.**

8.2 Pipeline Flow During Execution of Instructions

This section explains the pipeline flow during the execution of instructions.

In pipeline processing, the CPU is already processing the next instruction when the memory or I/O write cycle is generated. As a result, I/O manipulations and interrupt request masking will be reflected later than next instructions are issued (ID stage).

(1) NB85E, NB85ET, NU85E, NU85ET

When a dedicated interrupt controller (INTC) is connected to the NPB (NEC peripheral bus), maskable interrupt acknowledgement is disabled from the next instruction because the CPU detects access to the INTC and performs interrupt request mask processing.

(2) V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2

★ When an interrupt mask manipulation is performed, maskable interrupt acknowledged is disabled from the instruction immediately after an instruction because the CPU detects access to the internal INTC (ID stage) and performs interrupt request mask processing.

8.2.1 Load instructions

Caution Due to non-blocking control, there is no guarantee that the bus cycle is complete between the MEM stages. However, when accessing the peripheral I/O area, blocking control is effected, making it possible to wait for the end of the bus cycle at the MEM stage.

For the NB85E, NB85ET, NU85E, or NU85ET, non-blocking control is used for access to the programmable peripheral I/O area.

(1) LD instructions

[Instructions] LD.B, LD.BU, LD.H, LD.HU, LD.W

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>
LD 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 LD instruction, data wait time occurs.

(2) SLD instructions

[Instructions] SLD.B, SLD.BU, SLD.H, SLD.HU, SLD.W

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>
SLD instruction	IF	ID	MEM	WB		
Next instruction		IF	ID	EX	MEM	WB

★ [Description] The pipeline consists of 4 stages, IF, ID, MEM, and WB. If an instruction using the execution result is placed immediately after the SLD instruction, data wait time occurs.

8.2.2 Store instructions

Caution Due to non-blocking control, there is no guarantee that the bus cycle is complete between the MEM stages. However, when accessing the peripheral I/O area, blocking control is effected, making it possible to wait for the end of the bus cycle at the MEM stage.
For the NB85E, NB85ET, NU85E, or NU85ET, non-blocking control is used for access to the programmable peripheral I/O area.

[Instructions] ST.B, ST.H, ST.W, SST.B, SST.H, SST.W

[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 Multiply instructions

[Instructions] MUL, MULH, MULHI, MULU

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

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

(b) When next instruction is 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. The EX stage takes 2 clocks because it is executed by a multiplier. EX1 and EX2 stages (different from the normal EX stage) can operate independently. Therefore, the number of clocks for instruction execution is always 1 clock, even if several multiply instructions are executed in a row. However, if an instruction using the execution result is placed immediately after a multiply instruction, data wait time occurs.

8.2.4 Arithmetic operation instructions

(1) Instructions other than divide/move word instructions

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

[Pipeline]

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

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

(2) Move word instruction

[Instructions] MOV imm32

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>
Arithmetic operation instruction	IF	ID	EX1	EX2	DF	WB	
Next instruction		IF	–	ID	EX	MEM	WB

–: Idle inserted for wait

[Description] The pipeline consists of 6 stages, IF, ID, EX1, EX2 (normal EX stage), DF, and WB.

(3) Divide instructions

[Instructions] DIV, DIVH, DIVHU, DIVU

[Pipeline] (a) DIV, DIVH instructions

	<1>	<2>	<3>	<4>		<35>	<36>	<37>	<38>	<39>	<40>	<41>
Divide instruction	IF	ID	EX1	EX2		EX33	EX34	EX35	DF	WB		
Next instruction		IF	–	–		–	–	ID	EX	MEM	WB	
Next to next instruction								IF	ID	EX	MEM	WB

–: Idle inserted for wait

(b) DIVHU, DIVU instructions

	<1>	<2>	<3>	<4>		<35>	<36>	<37>	<38>	<39>	<40>
Divide instruction	IF	ID	EX1	EX2		EX33	EX34	DF	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 39 stages, IF, ID, EX1 to EX35 (normal EX stage), DF, and WB for DIV and DIVH instructions. The pipeline consists of 38 stages, IF, ID, EX1 to EX34 (normal EX stage), DF, and WB for DIVHU and DIVU instructions.

★ [Remark] If an interrupt occurs while a division instruction is executed, execution of the instruction is stopped, and the interrupt is processed, assuming that the return address is the first address of that instruction. After interrupt processing has been completed, the division instruction is executed again. In this case, general-purpose registers reg1 and reg2 hold the value before the instruction is executed.

8.2.5 Saturated operation instructions

[Instructions] SATADD, SATSUB, SATSUBI, SATSUBR

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>
Saturated operation instruction	IF	ID	EX	DF	WB	
Next instruction		IF	ID	EX	MEM	WB

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

8.2.6 Logical operation instructions

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

[Pipeline]

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

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

8.2.7 Branch instructions

(1) Conditional branch instructions (except BR instruction)

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

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

	<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 satisfied

	<1>	<2>	<3>	<4>	<5>	<6>	<7>
Conditional 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

[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 the branch destination is decided in the ID stage.

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

(b) When the condition is satisfied
The number of execution clocks for the branch instruction is 2. IF stage of the next instruction of the branch instruction is not executed.
If an instruction overwriting the contents of PSW occurs immediately before, the number of execution clocks is 3 because of flag hazard occurrence.

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

[Instructions] BR, JARL, JR

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>
BR instruction, 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 JR and BR instructions 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 EX, MEM, and WB stages, because the branch destination is decided in the ID stage. However, in the case of the JARL instruction, data is written to the restore PC in the WB stage. Also, the IF stage of the next instruction of the branch instruction is not executed.

(3) JMP instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>
JMP instruction	IF	ID	EX	MEM	WB		
Next instruction		(IF)					
Branch destination instruction			IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the EX, MEM, and WB stages, because the branch destination is decided in the ID stage.

8.2.8 Bit manipulation instructions

(1) CLR1, NOT1, SET1 instructions

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>	<9>
Bit manipulation instruction	IF	ID	EX1	MEM	EX2	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 7 stages, IF, ID, EX1, MEM, EX2 (normal stage), 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, the EX stage requires a total of 2 clocks, and the MEM stage requires a total of 2 cycles.

(2) TST1 instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>	<9>
Bit manipulation instruction	IF	ID	EX1	MEM	EX2	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 7 stages, IF, ID, EX1, MEM, EX2 (normal stage), 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 all, this instruction requires 2 clocks.

8.2.9 Special instructions

(1) CALLT instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>	<9>
CALLT instruction	IF	ID	MEM	EX	MEM	WB			
Next instruction		(IF)							
Branch destination instruction					IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed

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

(2) CTRET instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>
CTRET instruction	IF	ID	EX	MEM	WB		
Next instruction		(IF)					
Branch destination instruction			IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed

[Description] The pipeline consists of 5 stages, IF, ID, EX, MEM, and WB. However, no operation is performed in the EX, MEM, and WB stages, because the branch destination is decided in the ID stage.

(3) DI, EI instructions

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>
DI, EI 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.

★ [Remark] Both the DI and EI instructions do not sample an interrupt request. An interrupt is sampled as follows while these instructions are executed.

Instruction immediately before	IF	ID	EX	MEM	WB		
DI, EI instruction		IF	ID	EX	MEM	WB	
Instruction immediately after			IF	ID	EX	MEM	WB

Last sampling of interrupt before execution of EI or DI instruction

First sampling of interrupt after execution of EI or DI instruction

(4) DISPOSE instruction

[Pipeline] (a) When branch is not executed

	<1>	<2>	<3>	<4>	...	<n+2>	<n+3>	<n+4>	<n+5>	<n+6>	<n+7>
DISPOSE instruction	IF	ID	EX	MEM		MEM	MEM	MEM	WB		
Next instruction		IF	–	–		–	ID	EX	MEM	WB	
Next to next instruction							IF	ID	EX	MEM	WB

–: Idle inserted for wait

(b) When branch is executed

	<1>	<2>	<3>	<4>	...	<n+2>	<n+3>	<n+4>	<n+5>	<n+6>	<n+7>
DISPOSE instruction	IF	ID	EX	MEM		MEM	MEM	MEM	WB		
Next instruction		(IF)									
Branch destination instruction									IF	ID	EX

(IF): Instruction fetch that is not executed
–: Idle inserted for wait

[Description] The pipeline consists of n + 5 stages (n: register list number), IF, ID, EX, n + 1 times MEM, and WB. The MEM stage requires n + 1 cycles.

(5) HALT instruction

[Pipeline]

HALT mode release

	<1>	<2>	<3>	<4>	<5>	<6>	...						
HALT instruction	IF	ID	EX	MEM	WB								
Next instruction		IF	–	–	–	–		–	ID	EX	MEM	WB	
Next to 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 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 mode is released.

(6) LDSR, STSR instructions

[Pipeline]

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

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

(7) NOP instruction

[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.

★ **Caution Be aware that the SLD and Bcond instructions are sometimes executed at the same time as other 16-bit format instructions. For example, if the SLD and NOP instructions are executed simultaneously, the NOP instruction may keep the delay time from being generated.**

(8) PREPARE instruction

[Pipeline]

	<1>	<2>	<3>	<4>		<n+2>	<n+3>	<n+4>	<n+5>	<n+6>	<n+7>
PREPARE instruction	IF	ID	EX	MEM		MEM	MEM	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 n + 5 stages (n: register list number), IF, ID, EX, n + 1 times MEM, and WB. The MEM stage requires n + 1 cycles.

(9) RETI instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
RETI instruction	IF	ID1	ID2	EX	MEM	WB		
Next instruction		(IF)						
Next to next instruction			(IF)					
Jump destination instruction				IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed

ID1: Register selection

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 stages of the next instruction and next to next instruction are not executed.

(10) SWITCH instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>	<9>	<10>
SWITCH instruction	IF	ID	EX1	MEM	EX2	MEM	WB			
Next instruction		(IF)								
Branch destination instruction						IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed

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

(11) TRAP instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
TRAP instruction	IF	ID1	ID2	EX	DF	WB		
Next instruction		(IF)						
Next to next instruction			(IF)					
Jump destination instruction				IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed
ID1: Exception code (004nH, 005nH) detection (n = 0 to FH)
ID2: Address generation

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

8.2.10 Debug function instructions

(1) DBRET instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
DBRET instruction	IF	ID1	ID2	EX	MEM	WB		
Next instruction		(IF)						
Next to next instruction			(IF)					
Jump destination instruction				IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed
ID1: Register selection
ID2: Read DBPC

[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 the memory is not accessed and no data is written to registers. The ID stage requires 2 clocks. Also, the IF stages of the next instruction and next to next instruction are not executed.

(2) DBTRAP instruction

[Pipeline]

	<1>	<2>	<3>	<4>	<5>	<6>	<7>	<8>
DBTRAP instruction	IF	ID1	ID2	EX	DF	WB		
Next instruction		(IF)						
Next to next instruction			(IF)					
Jump destination instruction				IF	ID	EX	MEM	WB

(IF): Instruction fetch that is not executed
ID1: Exception code (0060H) detection
ID2: Address generation

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

★ 8.3 Pipeline Disorder

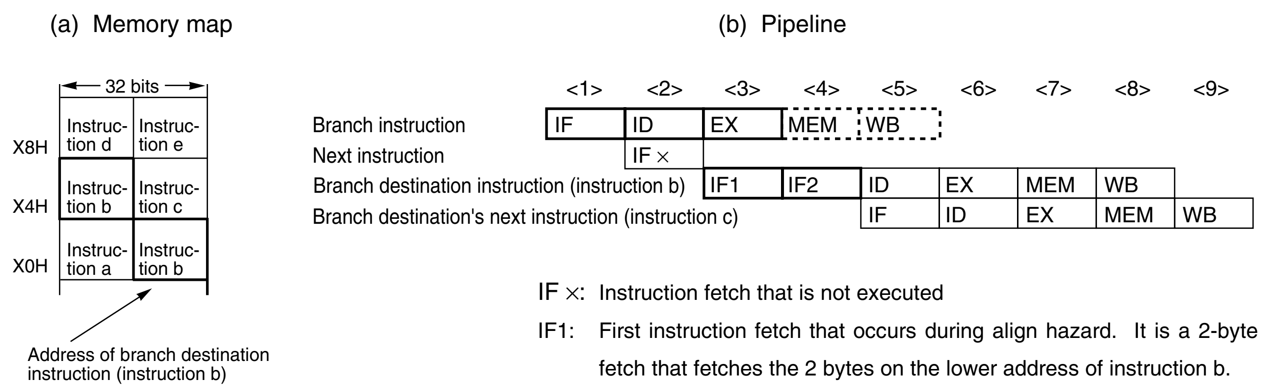
The pipeline consists of 5 stages from IF (Instruction Fetch) to WB (Write Back). Each stage basically requires 1 clock for processing, but the pipeline may become disordered, causing the number of execution clocks to increase. This section describes the main causes of pipeline disorder.

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=A0=0), and is not word aligned (A1=0, A0=0). Therefore, when this instruction b becomes the branch destination instruction, an align hazard occurs. When an align hazard occurs, the number of execution clocks of the branch instruction becomes 4.

Figure 8-6. Align Hazard Example



IF ×: Instruction fetch that is not executed

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

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

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

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

8.3.2 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 V850E1 CPU has an interlock function to automatically handle this hazard by delaying the ID stage of the next instruction.

The V850E1 CPU also has a short path that allows the data read during the MEM stage to be used in the ID stage of the next instruction. This short path allows data to be read with the load instruction during the MEM stage and the use of this data in the ID stage of the next instruction with the same timing.

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

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

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

8.3.3 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 V850E1 CPU'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 8-8. 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 shown in Figure 8-8, when an instruction placed immediately after a multiply instruction uses its execution result, a data wait time occurs due to the interlock function, and the execution speed is lowered. This drop in execution speed can be avoided by placing instructions that use the execution result of a multiply instruction at least 2 instructions after the multiply instruction.

8.3.4 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 V850E1 CPU'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 8-9. Example of Referencing 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 shown in Figure 8-9, when an STSR instruction is placed immediately after an LDSR instruction that uses the operand EIPC or FEPC, and that STSR instruction uses the LDSR instruction execution result, the interlock function causes a data wait time to occur, and the execution speed is lowered. This drop in execution speed can be avoided by placing STSR instructions that reference the execution result of the preceding LDSR instruction at least 3 instructions after the LDSR instruction.

8.3.5 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 V850E1 CPU 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) V850E1 CPU (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 V850E1 CPU (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 V850E1 CPU 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

- V850E1 CPU (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 V850E1 CPU (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

- V850E1 CPU (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 V850E1 CPU (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 LIST

The instruction function list in alphabetical order is shown in Table A-1, and instruction list in format order is shown in Table A-2.

Table A-1. Instruction Function List (in Alphabetical Order) (1/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
ADD	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Add. Adds the word data of reg1 to the word data of reg2, and stores the result to reg2.
ADD	imm5, reg2	II	0/1	0/1	0/1	0/1	–	Add. Adds the 5-bit immediate data, sign-extended to word length, to the word data of reg2, and stores the result to reg2.
ADDI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	–	Add Immediate. Adds the 16-bit immediate data, sign-extended to word length, to the word data of reg1, and stores the result to reg2.
AND	reg1, reg2	I	–	0	0/1	0/1	–	And. ANDs the word data of reg2 with the word data of reg1, and stores the result to reg2.
ANDI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	And. ANDs the word data of reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result to reg2.
Bcond	disp9	III	–	–	–	–	–	Branch on Condition Code. 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.
BSH	reg2, reg3	XII	0/1	0	0/1	0/1	–	Byte Swap Half-word. Performs endian conversion.
BSW	reg2, reg3	XII	0/1	0	0/1	0/1	–	Byte Swap Word. Performs endian conversion.
CALLT	imm6	II	–	–	–	–	–	Call with Table Look Up. Based on CTBP contents, updates PC value and transfers control.
CLR1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Clear Bit. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Then clears the bit, specified by the instruction bit field, of the byte data referenced by the generated address.

Table A-1. Instruction Function List (in Alphabetical Order) (2/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
CLR1	reg2 [reg1]	IX	–	–	–	0/1	–	Clear Bit. First, reads the data of reg1 to generate a 32-bit address. Then clears the bit, specified by the data of lower 3 bits of reg2 of the byte data referenced by the generated address.
CMOV	cccc, reg1, reg2, reg3	XI	–	–	–	–	–	Conditional Move. reg3 is set to reg1 if a condition specified by condition code “cccc” is satisfied; otherwise, set to the data of reg2.
CMOV	cccc, imm5, reg2, reg3	XII	–	–	–	–	–	Conditional Move. reg3 is set to the data of 5-immediate, sign-extended to word length, if a condition specified by condition code “cccc” is satisfied; otherwise, set to the data of reg2.
CMP	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Compare. Compares the word data of reg2 with the word data of reg1, and indicates the result by using the PSW flags. To compare, the contents of reg1 are subtracted from the word data of reg2.
CMP	imm5, reg2	II	0/1	0/1	0/1	0/1	–	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 PSW flags. To compare, the contents of the sign-extended immediate data are subtracted from the word data of reg2.
CTRET	(None)	X	0/1	0/1	0/1	0/1	0/1	Restore from CALLT. Restores the restore PC and PSW from the appropriate system register and restores from a routine called by CALLT.
DBRET[Note]	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from debug trap. Restores the restore PC and PSW from the appropriate system register and restores from a debug monitor routine.
DBTRAP[Note]	(None)	I	–	–	–	–	–	Debug trap. Saves the restore PC and PSW to the appropriate system register and transfers control by setting the PC to handler address (00000060H).
DI	(None)	X	–	–	–	–	–	Disables 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.
DISPOSE	imm5, list12	XIII	–	–	–	–	–	Function Dispose. Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pop (load data from the address specified by sp and adds 4 to sp) general registers listed in list12.

★ **Note** Not supported in the NB85E and NB85ET

Table A-1. Instruction Function List (in Alphabetical Order) (3/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
DISPOSE	imm5, list12, [reg1]	XIII	–	–	–	–	–	Function Dispose. Adds the data of 5-bit immediate imm5, logically shifted left by 2 and zero-extended to word length, to sp. Then pop (load data from the address specified by sp and adds 4 to sp) general registers listed in list12, transfers control to the address specified by reg1.
DIV	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Divide Word. Divides the word data of reg2 by the word data of reg1, and stores the quotient to reg2 and the remainder to reg3.
DIVH	reg1, reg2	I	–	0/1	0/1	0/1	–	Divide Half-word. Divides the word data of reg2 by the lower half-word data of reg1, and stores the quotient to reg2.
DIVH	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Divide Half-word. Divides word data of reg2 by lower half-word data of reg1, and stores the quotient to reg2 and the remainder to reg3.
DIVHU	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Divide Half-word Unsigned. Divides word data of reg2 by lower half-word data of reg1, and stores the quotient to reg2 and the remainder to reg3.
DIVU	reg1, reg2, reg3	XI	–	0/1	0/1	0/1	–	Divide Word Unsigned. Divides the word data of reg2 by the word data of reg1, and stores the quotient to reg2 and the remainder to reg3.
EI	(None)	X	–	–	–	–	–	Enable Interrupt. Clears the ID flag of the PSW to 0 and enables the acknowledgement of maskable interrupts at the beginning of next instruction.
HALT	(None)	X	–	–	–	–	–	Halt. Stops the operating clock of the CPU and places the CPU in the HALT mode.
HSW	reg2, reg3	XII	0/1	0	0/1	0/1	–	Half-word Swap Word. Performs endian conversion.
JARL	disp22, reg2	V	–	–	–	–	–	Jump and Register Link. Saves the current PC value plus 4 to general register reg2, adds a 22-bit displacement, sign-extended to word length, to the current PC value, and transfers control to the PC. Bit 0 of the 22-bit displacement is masked to 0.
JMP	[reg1]	I	–	–	–	–	–	Jump Register. Transfers control to the address specified by reg1. Bit 0 of the address is masked to 0.
JR	disp22	V	–	–	–	–	–	Jump Relative. Adds a 22-bit displacement, sign-extended to word length, to the current PC value, and transfers control to the PC. Bit 0 of the 22-bit displacement is masked to 0.

Table A-1. Instruction Function List (in Alphabetical Order) (4/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
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 to reg2.
LD.BU	disp16 [reg1], reg2	VII	–	–	–	–	–	Unsigned Byte Load. Adds the data of reg1 and the 16-bit displacement sign-extended to word length, and generates a 32-bit address. Then reads the byte data from the generated address, zero-extends it to word length, and stores it to 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 bit 0 masked to 0, sign-extended to word length, and stored to reg2.
LD.HU	disp16 [reg1], reg2	VII	–	–	–	–	–	Unsigned Half-word Load. Adds the data of reg1 and the 16-bit displacement sign-extended to word length to generate a 32-bit address. Reads the half-word data from the address masking bit 0 of this 32-bit address to 0, zero-extends it to word length, and stores it to reg2.
LD.W	disp16 [reg1], reg2	VII	–	–	–	–	–	Word Load. Adds the data of reg1 to a 16-bit displacement, sign-extended to word length, to generate a 32-bit address. Word data is read from this 32-bit address with bits 0 and 1 masked to 0, and stored to reg2.
LDSR	reg2, regID	IX	–	–	–	–	–	Load to System Register. Set the word data of reg2 to a system register specified by regID. If regID is PSW, the values of the corresponding bits of reg2 are set to the respective flags of the PSW.
MOV	reg1, reg2	I	–	–	–	–	–	Move. Transfers the word data of reg1 to reg2.
MOV	imm5, reg2	II	–	–	–	–	–	Move. Transfers the value of a 5-bit immediate data, sign-extended to word length, to reg2.
MOV	imm32, reg1	VI	–	–	–	–	–	Move. Transfers the 32-bit immediate data to reg1.
MOVEA	imm16, reg1, reg2	VI	–	–	–	–	–	Move Effective Address. Adds a 16-bit immediate data, sign-extended to word length, to the word data of reg1, and stores the result to reg2.

Table A-1. Instruction Function List (in Alphabetical Order) (5/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
MOVHI	imm16, reg1, reg2	VI	–	–	–	–	–	Move High 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 to reg2.
MUL	reg1, reg2, reg3	XI	–	–	–	–	–	Multiply Word. Multiplies the word data of reg2 by the word data of reg1, and stores the result to reg2 and reg3 as double-word data.
MUL	imm9, reg2, reg3	XII	–	–	–	–	–	Multiply Word. Multiplies the word data of reg2 by the 9-bit immediate data sign-extended to word length, and stores the result to reg2 and reg3.
MULH	reg1, reg2	I	–	–	–	–	–	Multiply Half-word. Multiplies the lower half-word data of reg2 by the lower half-word data of reg1, and stores the result to reg2 as word data.
MULH	imm5, reg2	II	–	–	–	–	–	Multiply Half-word. Multiplies the lower half-word data of reg2 by a 5-bit immediate data, sign-extended to half-word length, and stores the result to reg2 as word data.
MULHI	imm16, reg1, reg2	VI	–	–	–	–	–	Multiply Half-word Immediate. Multiplies the lower half-word data of reg1 by a 16-bit immediate data, and stores the result to reg2.
MULU	reg1, reg2, reg3	XI	–	–	–	–	–	Multiply Word Unsigned. Multiplies the word data of reg2 by the word data of reg1, and stores the result to reg2 and reg3 as double-word data. reg1 is not affected.
MULU	imm9, reg2, reg3	XII	–	–	–	–	–	Multiply Word Unsigned. Multiplies the word data of reg2 by the 9-bit immediate data sign-extended to word length, and store the result to reg2 and reg3.
NOP	(None)	I	–	–	–	–	–	No Operation.
NOT	reg1, reg2	I	–	0	0/1	0/1	–	Not. Logically negates (takes 1’s complement of) the word data of reg1, and stores the result to reg2.
NOT1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Not Bit. 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 bit number is inverted at the byte data location referenced by the generated address.
NOT1	reg2, [reg1]	IX	–	–	–	0/1	–	Not Bit. First, reads reg1 to generate a 32-bit address. The bit specified by the lower 3 bits of reg2 of the byte data of the generated address is inverted.

Table A-1. Instruction Function List (in Alphabetical Order) (6/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
OR	reg1, reg2	I	–	0	0/1	0/1	–	Or. ORs the word data of reg2 with the word data of reg1, and stores the result to reg2.
ORI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	Or Immediate. ORs the word data of reg1 with the 16-bit immediate data, zero-extended to word length, and stores the result to reg2.
PREPARE	list12, imm5	XIII	–	–	–	–	–	Function Prepare. The general register displayed in list12 is saved (4 is subtracted from sp, and the data is stored to that address). Next, the data is logically shifted 2 bits to the left, and the 5-bit immediate data zero-extended to word length is subtracted from sp.
PREPARE	list12, imm5, sp/imm	XIII	–	–	–	–	–	Function Prepare. The general register displayed in list12 is saved (4 is subtracted from sp, and the data is stored to that address). Next, the data is logically shifted 2 bits to the left, and the 5-bit immediate data zero-extended to word length is subtracted from sp. Then, the data specified by the third operand is loaded to ep.
RETI	(None)	X	0/1	0/1	0/1	0/1	0/1	Return from Trap or Interrupt. Reads the restore PC and PSW from the appropriate system register, and restores from interrupt or exception processing routine.
SAR	reg1, reg2	IX	0/1	0	0/1	0/1	–	Shift Arithmetic Right. Arithmetically shifts the word data of reg2 to the right by ‘n’ positions, where ‘n’ is specified by the lower 5 bits of reg1 (the MSB prior to shift execution is copied and set as the new MSB), and then writes the result to reg2.
SAR	imm5, reg2	II	0/1	0	0/1	0/1	–	Shift Arithmetic Right. Arithmetically shifts the word data of reg2 to the right by ‘n’ positions specified by the lower 5-bit immediate data, zero-extended to word length (the MSB prior to shift execution is copied and set as the new MSB), and then writes the result to reg2.
SASF	cccc, reg2	IX	–	–	–	–	–	Shift and Set Flag Condition. reg2 is logically shifted left by 1, and its LSB is set to 1 in a condition specified by condition code “cccc” is satisfied; otherwise, LSB is set to 0.

Table A-1. Instruction Function List (in Alphabetical Order) (7/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SATADD	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated Add. Adds the word data of reg1 to the word data of reg2, and stores the result to reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored to reg2; if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. The SAT flag is set to 1.
SATADD	imm5, reg2	II	0/1	0/1	0/1	0/1	0/1	Saturated Add. Adds the 5-bit immediate data, sign-extended to word length, to the word data of reg2, and stores the result to reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored to reg2; if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. The SAT flag is set to 1.
SATSUB	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated Subtract. Subtracts the word data of reg1 from the word data of reg2, and stores the result to reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored to reg2; if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. The SAT flag is set to 1.
SATSUBI	imm16, reg1, reg2	VI	0/1	0/1	0/1	0/1	0/1	Saturated Subtract Immediate. Subtracts a 16-bit immediate data, sign-extended to word length, from the word data of reg1, and stores the result to reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored to reg2; if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. The SAT flag is set to 1.
SATSUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	0/1	Saturated Subtract Reverse. Subtracts the word data of reg2 from the word data of reg1, and stores the result to reg2. However, if the result exceeds the maximum positive value, the maximum positive value is stored to reg2; if the result exceeds the maximum negative value, the maximum negative value is stored to reg2. The SAT flag is set to 1.
SET1	bit#3, disp16 [reg1]	VIII	–	–	–	0/1	–	Set Bit. First, adds a 16-bit displacement, sign-extended to word length, to the data of reg1 to generate a 32-bit address. The bits, specified by the 3-bit bit number, are set at the byte data location specified by the generated address.

Table A-1. Instruction Function List (in Alphabetical Order) (8/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SET1	reg2, [reg1]	IX	–	–	–	0/1	–	Set Bit. First, reads the data of general register reg1 to generate a 32-bit address. The bit, specified by the data of lower 3 bits of reg2, is set at the byte data location referenced by the generated address.
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 to reg2.
SHL	reg1, reg2	IX	0/1	0	0/1	0/1	–	Shift Logical Left. Logically shifts the word data of reg2 to the left by ‘n’ positions (0 is shifted to the LSB side), where ‘n’ is specified by the lower 5 bits of reg1, and then writes the result to reg2.
SHL	imm5, reg2	II	0/1	0	0/1	0/1	–	Shift Logical Left. Logically shifts the word data of reg2 to the left by ‘n’ positions (0 is shifted to the LSB side), where ‘n’ is specified by a 5-bit immediate data, zero-extended to word length, and then writes the result to reg2.
SHR	reg1, reg2	IX	0/1	0	0/1	0/1	–	Shift Logical Right. Logically shifts the word data of reg2 to the right by ‘n’ positions (0 is shifted to the MSB side), where ‘n’ is specified by the lower 5 bits of reg1, and then writes the result to reg2.
SHR	imm5, reg2	II	0/1	0	0/1	0/1	–	Shift Logical Right. Logically shifts the word data of reg2 to the right by ‘n’ positions (0 is shifted to the MSB side), where ‘n’ is specified by a 5-bit immediate data, zero-extended to word length, and then writes the result to reg2.
SLD.B	disp7 [ep], reg2	IV	–	–	–	–	–	Byte Load. Adds the 7-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, sign-extended to word length, and then stored to reg2.
SLD.BU	disp4 [ep], reg2	IV	–	–	–	–	–	Unsigned Byte Load. Adds the 4-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Byte data is read from the generated address, zero-extended to word length, and stored to 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 to reg2.

Table A-1. Instruction Function List (in Alphabetical Order) (9/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SLD.HU	disp5 [ep], reg2	IV	–	–	–	–	–	Unsigned Half-word Load. Adds the 5-bit displacement, zero-extended to word length, to the element pointer to generate a 32-bit address. Half-word data is read from this 32-bit address with bit 0 masked to 0, zero-extended to word length, and stored to 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 to 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 to 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 to 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 to 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 to 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 to the generated 32-bit address with bit 0 masked to 0.
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 to the generated 32-bit address with bits 0 and 1 masked to 0.
STSR	regID, reg2	IX	–	–	–	–	–	Store Contents of System Register. Stores the contents of a system register specified by regID to reg2.

Table A-1. Instruction Function List (in Alphabetical Order) (10/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
SUB	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Subtract. Subtracts the word data of reg1 from the word data of reg2, and stores the result to reg2.
SUBR	reg1, reg2	I	0/1	0/1	0/1	0/1	–	Subtract Reverse. Subtracts the word data of reg2 from the word data of reg1, and stores the result to reg2.
SWITCH	reg1	I	–	–	–	–	–	Jump with Table Look Up. Adds the table entry address (address following SWITCH instruction) and data of reg1 logically shifted to the left by 1 bit, and loads the half-word entry data specified by the table entry address. Next, logically shifts to the left by 1 bit the loaded data, and after sign-extending it to word length, branches to the target address added to the table entry address (instruction following SWITCH instruction).
SXB	reg1	I	–	–	–	–	–	Sign Extend Byte. Sign-extends the lowermost byte of reg1 to word length.
SXH	reg1	I	–	–	–	–	–	Sign Extend Half-word. Sign-extends lower half-word of reg1 to word length.
TRAP	vector	X	–	–	–	–	–	Trap. Saves the restore PC and PSW; 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, and starts exception processing.
TST	reg1, reg2	I	–	0	0/1	0/1	–	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	–	–	–	0/1	–	Test Bit. 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 bit number, 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 cleared to 0.
TST1	reg2, [reg1]	IX	–	–	–	0/1	–	Test Bit. First, reads the data of reg1 to generate a 32-bit address. If the bits indicated by the lower 3 bits of reg2 of the byte data of the generated address are 0, the Z flag is set to 1, and if they are 1, the Z flag is cleared to 0.
XOR	reg1, reg2	I	–	0	0/1	0/1	–	Exclusive Or. Exclusively ORs the word data of reg2 with the word data of reg1, and stores the result to reg2.

Table A-1. Instruction Function List (in Alphabetical Order) (11/11)

Mnemonic	Operand	Format	Flag					Instruction Function
			CY	OV	S	Z	SAT	
XORI	imm16, reg1, reg2	VI	–	0	0/1	0/1	–	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 to reg2.
ZXB	reg1	I	–	–	–	–	–	Zero Extend Byte. Zero-extends to word length the lowest byte of reg1.
ZXH	reg1	I	–	–	–	–	–	Zero Extend Half-word. Zero-extends to word length the lower half-word of reg1.

Table A-2. Instruction List (in Format Order) (1/3)

Format	Opcode		Mnemonic	Operand
	15 0	31 16		
I	0000000000000000	–	NOP	–
	rrrrr000000RRRRR	–	MOV	reg1, reg2
	rrrrr000001RRRRR	–	NOT	reg1, reg2
	rrrrr000010RRRRR	–	DIVH	reg1, reg2
	00000000010RRRRR	–	SWITCH	reg1
	00000000011RRRRR	–	JMP	[reg1]
	rrrrr000100RRRRR	–	SATSUBR	reg1, reg2
	rrrrr000101RRRRR	–	SATSUB	reg1, reg2
	rrrrr000110RRRRR	–	SATADD	reg1, reg2
	rrrrr000111RRRRR	–	MULH	reg1, reg2
	00000000100RRRRR	–	ZXB	reg1
	00000000101RRRRR	–	SXB	reg1
	00000000110RRRRR	–	ZXH	reg1
	00000000111RRRRR	–	SXH	reg1
	rrrrr001000RRRRR	–	OR	reg1, reg2
	rrrrr001001RRRRR	–	XOR	reg1, reg2
	rrrrr001010RRRRR	–	AND	reg1, reg2
	rrrrr001011RRRRR	–	TST	reg1, reg2
	rrrrr001100RRRRR	–	SUBR	reg1, reg2
	rrrrr001101RRRRR	–	SUB	reg1, reg2
	rrrrr001110RRRRR	–	ADD	reg1, reg2
	rrrrr001111RRRRR	–	CMP	reg1, reg2
	1111100001000000	–	DBTRAP[Note]	–
II	rrrrr010000iiiii	–	MOV	imm5, reg2
	rrrrr010001iiiii	–	SATADD	imm5, reg2
	rrrrr010010iiiii	–	ADD	imm5, reg2
	rrrrr010011iiiii	–	CMP	imm5, reg2
	0000001000iiiiii	–	CALLT	imm6
	rrrrr010100iiiii	–	SHR	imm5, reg2
	rrrrr010101iiiii	–	SAR	imm5, reg2
	rrrrr010110iiiii	–	SHL	imm5, reg2
	rrrrr010111iiiii	–	MULH	imm5, reg2
III	ddddd1011dddCCCC	–	Bcond	disp9

★ **Note** Not supported in the NB85E and NB85ET

Table A-2. Instruction List (in Format Order) (2/3)

Format	Opcode		Mnemonic	Operand
	15 0	31 16		
IV	rrrrr0000110dddd	–	SLD.BU	disp4 [ep], reg2
	rrrrr0000111dddd	–	SLD.HU	disp5 [ep], reg2
	rrrrr0110ddddddd	–	SLD.B	disp7 [ep], reg2
	rrrrr0111ddddddd	–	SST.B	reg2, disp7 [ep]
	rrrrr1000ddddddd	–	SLD.H	disp8 [ep], reg2
	rrrrr1001ddddddd	–	SST.H	reg2, disp8 [ep]
	rrrrr1010dddddd0	–	SLD.W	disp8 [ep], reg2
	rrrrr1010dddddd1	–	SST.W	reg2, disp8 [ep]
V	rrrrr11110dddddd	ddddddddddddddd0	JARL	disp22, reg2
	0000011110dddddd	ddddddddddddddd0	JR	disp22
VI	rrrrr110000RRRRR	iiiiiiiiiiiiiiii	ADDI	imm16, reg1, reg2
	rrrrr110001RRRRR	iiiiiiiiiiiiiiii	MOVEA	imm16, reg1, reg2
	rrrrr110010RRRRR	iiiiiiiiiiiiiiii	MOVHI	imm16, reg1, reg2
	rrrrr110011RRRRR	iiiiiiiiiiiiiiii	SATSUBI	imm16, reg1, reg2
	00000110001RRRRR	**Note**	MOV	imm32, reg1
	rrrrr110100RRRRR	iiiiiiiiiiiiiiii	ORI	imm16, reg1, reg2
	rrrrr110101RRRRR	iiiiiiiiiiiiiiii	XORI	imm16, reg1, reg2
	rrrrr110110RRRRR	iiiiiiiiiiiiiiii	ANDI	imm16, reg1, reg2
	rrrrr110111RRRRR	iiiiiiiiiiiiiiii	MULHI	imm16, reg1, reg2
VII	rrrrr111000RRRRR	dddddddddddddddd	LD.B	disp16 [reg1], reg2
	rrrrr111001RRRRR	ddddddddddddddd0	LD.H	disp16 [reg1], reg2
	rrrrr111001RRRRR	ddddddddddddddd1	LD.W	disp16 [reg1], reg2
	rrrrr111010RRRRR	dddddddddddddddd	ST.B	reg2, disp16 [reg1]
	rrrrr111011RRRRR	ddddddddddddddd0	ST.H	reg2, disp16 [reg1]
	rrrrr111011RRRRR	ddddddddddddddd1	ST.W	reg2, disp16 [reg1]
	rrrrr11110bRRRRR	ddddddddddddddd1	LD.BU	disp16 [reg1], reg2
	rrrrr111111RRRRR	ddddddddddddddd1	LD.HU	disp16 [reg1], reg2
VIII	00bbb111110RRRRR	dddddddddddddddd	SET1	bit#3, disp16 [reg1]
	01bbb111110RRRRR	dddddddddddddddd	NOT1	bit#3, disp16 [reg1]
	10bbb111110RRRRR	dddddddddddddddd	CLR1	bit#3, disp16 [reg1]
	11bbb111110RRRRR	dddddddddddddddd	TST1	bit#3, disp16 [reg1]

Note 32-bit immediate data. The higher 32 bits (bits 16 to 47) are as follows.

31 16	47 32
iiiiiiiiiiiiiiii	IIIIIIIIIIIIIIII

Table A-2. Instruction List (in Format Order) (3/3)

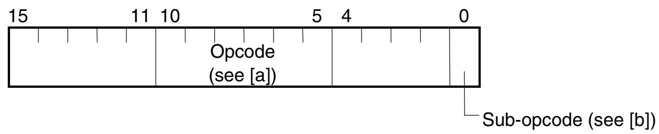
Format	Opcode		Mnemonic	Operand
	15 0	31 16		
IX	rrrrr1111110cccc	0000000000000000	SETF	cccc, reg2
	rrrrr111111RRRRR	0000000000100000	LDSR	reg2, regID
	rrrrr111111RRRRR	0000000001000000	STSR	regID, reg2
	rrrrr111111RRRRR	0000000010000000	SHR	reg1, reg2
	rrrrr111111RRRRR	0000000010100000	SAR	reg1, reg2
	rrrrr111111RRRRR	0000000011000000	SHL	reg1, reg2
	rrrrr111111RRRRR	0000000011100000	SET1	reg2, [reg1]
	rrrrr111111RRRRR	0000000011100010	NOT1	reg2, [reg1]
	rrrrr111111RRRRR	0000000011100100	CLR1	reg2, [reg1]
	rrrrr111111RRRRR	0000000011100110	TST1	reg2, [reg1]
	rrrrr1111110cccc	0000001000000000	SASF	cccc, reg2
X	00000111111iiiii	0000000100000000	TRAP	vector
	0000011111100000	0000000100100000	HALT	–
	0000011111100000	0000000101000000	RETI	–
	0000011111100000	0000000101000100	CTRET	–
	0000011111100000	0000000101000110	DBRET[Note]	–
	0000011111100000	0000000101100000	DI	–
	1000011111100000	0000000101100000	EI	–
XI	rrrrr111111RRRRR	wwwww01000100000	MUL	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww01000100010	MULU	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww01010000000	DIVH	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww01010000010	DIVHU	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww01011000000	DIV	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww01011000010	DIVU	reg1, reg2, reg3
	rrrrr111111RRRRR	wwwww011001cccc0	CMOV	cccc, reg1, reg2, reg3
XII	rrrrr111111iiiii	wwwww01001IIII00	MUL	imm9, reg2, reg3
	rrrrr111111iiiii	wwwww01001IIII10	MULU	imm9, reg2, reg3
	rrrrr111111iiiii	wwwww011000cccc0	CMOV	cccc, imm5, reg2, reg3
	rrrrr11111100000	wwwww01101000000	BSW	reg2, reg3
	rrrrr11111100000	wwwww01101000010	BSH	reg2, reg3
	rrrrr11111100000	wwwww01101000100	HSW	reg2, reg3
XIII	0000011001iiiiiL	LLLLLLLLLLLRRRRR	DISPOSE	imm5, list12, [reg1]
	0000011001iiiiiL	LLLLLLLLLLL00000	DISPOSE	imm5, list12
	0000011110iiiiiL	LLLLLLLLLLL00001	PREPARE	list12, imm5
	0000011110iiiiiL	LLLLLLLLLLLff011	PREPARE	list12, imm5, sp/imm

★ **Note** Not supported in the NB85E and NB85ET

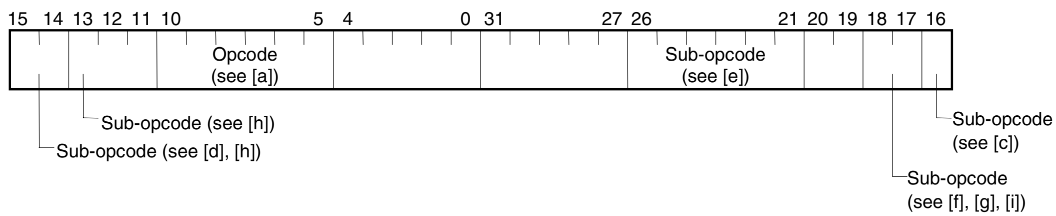
APPENDIX B INSTRUCTION OPCODE MAP

This chapter shows the opcode map for the instruction code shown below.

(1) 16-bit format instruction



★ **(2) 32-bit format instruction**



Remark Operand convention

Symbol	Meaning
R	reg1: General register (used as source register)
r	reg2: General register (mainly used as destination register. Some are also used as source registers.)
w	reg3: General register (mainly used as remainder of division results or higher 32 bits of multiply results)
bit#3	3-bit data for bit number specification
imm×	×-bit immediate data
disp×	×-bit displacement data
cccc	4-bit data condition code specification

[a] Opcode

Bit 10	Bit 9	Bit 8	Bit 7	Bits 6, 5: 0,0	Bits 6, 5: 0,1	Bits 6, 5: 1,0	Bits 6, 5: 1,1	Format
0	0	0	0	MOV R, r NOP[Note 1]	NOT	DIVH SWITCH[Note 2] DBTRAP Undefined[Note 3]	JMP[Note 4] SLD.BU[Note 5] SLD.HU[Note 6]	I, IV
0	0	0	1	SATSUBR ZXB[Note 4]	SATSUB SXB[Note 4]	SATADD R, r ZXH[Note 4]	MULH SXH[Note 4]	I
0	0	1	0	OR	XOR	AND	TST	
0	0	1	1	SUBR	SUB	ADD R, r	CMP R, r	
0	1	0	0	MOV imm5, r CALLT[Note 4]	SATADD imm5, r	ADD imm5, r	CMP imm5, r	II
0	1	0	1	SHR imm5, r	SAR imm5, r	SHL imm5, r	MULH imm5, r Undefined[Note 4]	
0	1	1	0	SLD.B				IV
0	1	1	1	SST.B				
1	0	0	0	SLD.H				
1	0	0	1	SST.H				
1	0	1	0	SLD.W[Note 7] SST.W[Note 7]				
1	0	1	1	Bcond				III
1	1	0	0	ADDI	MOVEA MOV imm32, R[Note 4]	MOVHI DISPOSE[Note 4]	SATSUBI	VI, XIII
1	1	0	1	ORI	XORI	ANDI	MULHI Undefined[Note 4]	VI
1	1	1	0	LD.B	LD.H[Note 8] LD.W[Note 8]	ST.B	ST.H[Note 8] ST.W[Note 8]	VII
1	1	1	1	JR JARL LD.BU[Note 10] PREPARE[Note 11]		Bit manipulation 1[Note 9]	LD.HU[Note 10] Undefined[Note 11] Expansion 1[Note 12]	V, VII, VIII, XIII

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

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

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

4. If R (reg2) = r0 (instruction without reg2)

5. If bit 4 = 0 and r (reg2) ≠ r0 (instruction with reg2)

6. If bit 4 = 1 and r (reg2) ≠ r0 (instruction with reg2)

7. See **[b]**

8. See **[c]**

9. See **[d]**

10. If bit 16 = 1 and r (reg2) ≠ r0 (instruction with reg2)

11. If bit 16 = 1 and r (reg2) = r0 (instruction without reg2)

12. See **[e]**

★ **Remark** The NB85E and NB85ET do not support the DBTRAP instruction.

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

Bit 10	Bit 9	Bit 8	Bit 7	Bit 0	
				0	1
0	1	1	0	SLD.B	
0	1	1	1	SST.B	
1	0	0	0	SLD.H	
1	0	0	1	SST.H	
1	0	1	0	SLD.W	SST.W

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

Bit 6	Bit 5	Bit 16	
		0	1
0	0	LD.B	
0	1	LD.H	LD.W
1	0	ST.B	
1	1	ST.H	ST.W

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

Bit 15	Bit 14	
	0	1
0	SET1 bit#3, disp16 [R]	NOT1 bit#3, disp16 [R]
1	CLR1 bit#3, disp16 [R]	TST1 bit#3, disp16 [R]

[e] Expansion 1 (sub-opcode)

Bit 26	Bit 25	Bit 24	Bit 23	Bits 22, 21				Format
				0,0	0,1	1,0	1,1	
0	0	0	0	SETF	LDSR	STSR	Undefined	IX
0	0	0	1	SHR	SAR	SHL	Bit manipulation 2[Note 1]	
0	0	1	0	TRAP	HALT	RETI[Note 2] CTRET[Note 2] DBRET[Note 2] Undefined	EI[Note 3] DI[Note 3] Undefined	X
0	0	1	1	Undefined		Undefined		–
0	1	0	0	SASF	MUL R, r, w MULU R, r, w[Note 4]	MUL imm9, r, w MULU imm9, r, w[Note 4]		IX, XI, XII
0	1	0	1	DIVH DIVHU[Note 4]		DIV DIVU[Note 4]		XI
0	1	1	0	CMOV cccc, imm5, r, w	CMOV cccc, R, r, w	BSW[Note 5] BSH[Note 5] HSW[Note 5]	Undefined	XI, XII
0	1	1	1	Illegal instruction				–
1	x	x	x					

Notes 1. See **[f]**

2. See **[g]**

3. See **[h]**

4. If bit 17 = 1

5. See **[i]**

★ **Remark** The NB85E and NB85ET do not support the DBRET instruction.

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

Bit 18	Bit 17	
	0	1
0	SET1 r, [R]	NOT1 r, [R]
1	CLR1 r, [R]	TST1 r, [R]

[g] Return instruction (sub-opcode)

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

[h] PSW operation instruction (sub-opcode)

Bit 15	Bit 14	Bits 13, 12, 11							
		0,0,0	0,0,1	0,1,0	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
0	0	DI	Undefined						
0	1	Undefined							
1	0	EI	Undefined						
1	1	Undefined							

[i] Endian conversion instruction (sub-opcode)

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

APPENDIX C DIFFERENCES WITH ARCHITECTURE OF V850 CPU

(1/2)

Item		V850E1 CPU	V850 CPU
Instructions (including operand)	BSH reg2, reg3	Provided	Not provided
	BSW reg2, reg3		
	CALLT imm6		
	CLR1 reg2, [reg1]		
	CMOV cccc, imm5, reg2, reg3		
	CMOV cccc, reg1, reg2, reg3		
	CTRET		
	DBRET[Note]		
	DBTRAP[Note]		
	DISPOSE imm5, list12		
	DISPOSE imm5, list12 [reg1]		
	DIV reg1, reg2, reg3		
	DIVH reg1, reg2, reg3		
	DIVHU reg1, reg2, reg3		
	DIVU reg1, reg2, reg3		
	HSW reg2, reg3		
	LD.BU disp16 [reg1], reg2		
	LD.HU disp16 [reg1], reg2		
	MOV imm32, reg1		
	MUL imm9, reg2, reg3		
	MUL reg1, reg2, reg3		
	MULU reg1, reg2, reg3		
	MULU imm9, reg2, reg3		
	NOT1 reg2, [reg1]		
	PREPARE list12, imm5		
	PREPARE list12, imm5, sp/imm		
	SASF cccc, reg2		
	SET1 reg2, [reg1]		
	SLD.BU disp4 [ep], reg2		
	SLD.HU disp5 [ep], reg2		
	SWITCH reg1		
	SXB reg1		
	SXH reg1		
	TST1 reg2, [reg1]		
	ZXB reg1		
	ZXH reg1		

★ **Note** Not supported in the NB85E and NB85ET

(2/2)

Item		V850E1 CPU	V850 CPU
Instruction format	Format IV	Format is different for some instructions.	
	Format XI	Provided	Not provided
	Format XII		
	Format XIII		
Instruction execution clocks		Value differs for some instructions.	
Program space		64 MB linear	16 MB linear
Valid bits of program counter (PC)		Lower 26 bits	Lower 24 bits
System register	CALLT execution status saving registers (CTPC, CTPSW)	Provided	Not provided
	Exception/debug trap status saving registers (DBPC, DBPSW)		
	CALLT base pointer (CTBP)		
	Debug interface register (DIR)[Note 1]		
	Breakpoint control registers 0 and 1 (BPC0, BPC1)[Note 1]		
	Program ID register (ASID)[Note 1]		
	Breakpoint address setting registers 0 and 1 (BPAV0, BPAV1)[Note 1]		
	Breakpoint address mask registers 0 and 1 (BPAM0, BPAM1)[Note 1]		
	Breakpoint data setting registers 0 and 1 (BPDV0, BPDV1)[Note 1]		
	Breakpoint data mask registers 0 and 1 (BPDM0, BPDM1)[Note 1]		
	Exception trap status saving registers	DBPC, DBPSW	EIPC, EIPSW
Illegal instruction code		Instruction code areas differ.	
Misaligned access enable/disable setting		Can be set depending on product	Cannot be set. (misaligned access disabled)
Non-maskable interrupt (NMI)	Input	3 (NB85E, NB85ET, NU85E, NU85ET)	1
		1 (V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2)	
	Exception code	0010H, 0020H, 0030H	0010H
	Handler address	00000010H, 00000020H, 00000030H	00000010H
Debug trap[Note 2]		Provided	Not provided
Pipeline		At next instruction, pipeline flow differs. • Arithmetic operation instruction • Branch instruction • Bit manipulation instruction • Special instruction (TRAP, RETI)	

★ **Notes 1.** Used only for the NU85E and NU85ET

★ **2.** Not supported in the NB85E and NB85ET

APPENDIX D INSTRUCTIONS ADDED FOR V850E1 CPU COMPARED WITH V850 CPU

Compared with the instruction codes of the V850 CPU, the instruction codes of the V850E1 CPU are upward compatible at the object code level. In the case of the V850E1 CPU, instructions that even if executed have no meaning in the case of the V850 CPU (mainly instructions performing write to the r0 register) are extended as additional instructions.

The following table shows the V850 CPU instructions corresponding to the instruction codes added in the V850E1 CPU. See the table when switching from products that incorporate the V850 CPU to products that incorporate the V850E1 CPU.

Table D-1. Instructions Added to V850E1 CPU and V850 CPU Instructions with Same Instruction Code (1/2)

Instructions Added in V850E1 CPU	V850 CPU Instructions with Same Instruction Code as V850E1 CPU
CALLT imm6	MOV imm5, r0 or SATADD imm5, r0
DISPOSE imm5, list12	MOVHI imm16, reg1, r0 or SATSUBI imm16, reg1, r0
DISPOSE imm5, list12 [reg1]	MOVHI imm16, reg1, r0 or SATSUBI imm16, reg1, r0
MOV imm32, reg1	MOVEA imm16, reg1, r0
SWITCH reg1	DIVH reg1, r0
SXB reg1	SATSUB reg1, r0
SXH reg1	MULH reg1, r0
ZXB reg1	SATSUBR reg1, r0
ZXH reg1	SATADD reg1, r0
(RFU)	MULH imm5, r0
(RFU)	MULHI imm16, reg1, r0
BSH reg2, reg3	Illegal instruction
BSW reg2, reg3	
CMOV cccc, imm5, reg2, reg3	
CMOV cccc, reg1, reg2, reg3	
CTRET	
DIV reg1, reg2, reg3	
DIVH reg1, reg2, reg3	
DIVHU reg1, reg2, reg3	
DIVU reg1, reg2, reg3	
HSW reg2, reg3	
MUL imm9, reg2, reg3	
MUL reg1, reg2, reg3	
MULU reg1, reg2, reg3	
MULU imm9, reg2, reg3	
SASF cccc, reg2	

Table D-1. Instructions Added to V850E1 CPU and V850 CPU Instructions with Same Instruction Code (2/2)

Instructions Added in V850E1 CPU	V850 CPU Instructions with Same Instruction Code as V850E1 CPU
CLR1 reg2, [reg1]	Undefined
DBRET[Note]	
DBTRAP[Note]	
LD.BU disp16 [reg1], reg2	
LD.HU disp16 [reg1], reg2	
NOT1 reg2, [reg1]	
PREPARE list12, imm5	
PREPARE list12, imm5, sp/imm	
SET1 reg2, [reg1]	
SLD.BU disp4 [ep], reg2	
SLD.HU disp5 [ep], reg2	
TST1 reg2, [reg1]	

★ **Note** Not supported in the NB85E and NB85ET

[P]

[R]

[S]

[T]

[U]

[W]

[X]

[Z]

[MEMO]

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