

OMC952723009

# H8S/2600 Series, H8S/2000Series Programming Manual



## Preface

The H8S/2600 Series and the H8S/2000 Series are built around an H8S/2000 CPU core.

The H8S/2600 and H8S/2000 CPUs have the same internal 32-bit architecture. Both CPUs execute basic instructions in one state, have sixteen 16-bit registers, and have a concise, optimized instruction set. They can address a 16-Mbyte linear address space. Programs coded in the high-level language C can be compiled to high-speed executable code.

For easy migration, the instruction set is upward-compatible with the H8/300H, H8/300, and H8/300L Series at the object-code level.

The H8S/2600 CPU is upward-compatible with the H8S/2000 CPU at the object-code level, and supports sum of products instructions.

This manual gives details of the H8S/2600 and H8S/2000 instructions and can be used with all microcontrollers in the H8S/2600 Series and the H8S/2000 Series.

For hardware details, refer to the relevant microcontroller hardware manuals.

# Contents

Section 1	CPU .....	1
1.1	Overview.....	1
1.1.1	Features.....	1
1.1.2	Differences between H8S/2600 CPU and H8S/2000 CPU.....	2
1.1.3	Differences from H8/300 CPU .....	3
1.1.4	Differences from H8/300H CPU .....	3
1.2	CPU Operating Modes.....	5
1.3	Address Space.....	10
1.4	Register Configuration.....	11
1.4.1	Overview.....	11
1.4.2	General Registers.....	12
1.4.3	Control Registers .....	13
1.4.4	Initial Register Values.....	15
1.5	Data Formats.....	16
1.5.1	General Register Data Formats.....	16
1.5.2	Memory Data Formats.....	18
1.6	Instruction Set.....	19
1.6.1	Overview.....	19
1.6.2	Instructions and Addressing Modes.....	20
1.6.3	Table of Instructions Classified by Function .....	22
1.6.4	Basic Instruction Formats .....	32
1.7	Addressing Modes and Effective Address Calculation .....	33
Section 2	Instruction Descriptions .....	41
2.1	Tables and Symbols .....	41
2.1.1	Assembly-Language Format.....	42
2.1.2	Operation .....	43
2.1.3	Condition Code .....	45
2.1.4	Instruction Format .....	45
2.1.5	Register Specification.....	46
2.1.6	Bit Data Access in Bit Manipulation Instructions .....	47
2.2	Instruction Descriptions.....	47
2.2.1 (1)	ADD (B) .....	48
2.2.1 (2)	ADD (W) .....	49
2.2.1 (3)	ADD (L).....	50
2.2.2	ADDS .....	51
2.2.3	ADDX.....	52
2.2.4 (1)	AND (B) .....	53
2.2.4 (2)	AND (W) .....	54

2.2.4 (3)	AND (L).....	55
2.2.5 (1)	ANDC.....	56
2.2.5 (2)	ANDC.....	57
2.2.6	BAND.....	58
2.2.7	Bcc.....	60
2.2.8	BCLR.....	62
2.2.9	BIAND.....	64
2.2.10	BILD.....	66
2.2.11	BIOR.....	68
2.2.12	BIST.....	70
2.2.13	BIXOR.....	72
2.2.14	BLD.....	74
2.2.15	BNOT.....	76
2.2.16	BOR.....	78
2.2.17	BSET.....	80
2.2.18	BSR.....	82
2.2.19	BST.....	84
2.2.20	BTST.....	86
2.2.21	BXOR.....	88
2.2.22	CLRMAC.....	90
2.2.23 (1)	CMP (B).....	91
2.2.23 (2)	CMP (W).....	92
2.2.23 (3)	CMP (L).....	93
2.2.24	DAA.....	94
2.2.25	DAS.....	96
2.2.26 (1)	DEC (B).....	98
2.2.26 (2)	DEC (W).....	99
2.2.26 (3)	DEC (L).....	100
2.2.27 (1)	DIVXS (B).....	101
2.2.27 (2)	DIVXS (W).....	103
2.2.28 (1)	DIVXU (B).....	105
2.2.28 (2)	DIVXU (W).....	107
2.2.29 (1)	EEPMOV (B).....	109
2.2.29 (2)	EEPMOV (W).....	110
2.2.30 (1)	EXTS (W).....	112
2.2.30 (2)	EXTS (L).....	113
2.2.31 (1)	EXTU (W).....	114
2.2.31 (2)	EXTU (L).....	115
2.2.32 (1)	INC (B).....	116
2.2.32 (2)	INC (W).....	117
2.2.32 (3)	INC (L).....	118

2.2.33	JMP .....	119
2.2.34	JSR .....	120
2.2.35 (1)	LDC (B) .....	122
2.2.35 (2)	LDC (B) .....	123
2.2.35 (3)	LDC (W) .....	124
2.2.35 (4)	LDC (W) .....	126
2.2.36	LDM .....	128
2.2.37	LDMAC .....	130
2.2.38	MAC .....	131
2.2.39 (1)	MOV (B) .....	134
2.2.39 (2)	MOV (W) .....	135
2.2.39 (3)	MOV (L) .....	136
2.2.39 (4)	MOV (B) .....	137
2.2.39 (5)	MOV (W) .....	139
2.2.39 (6)	MOV (L) .....	141
2.2.39 (7)	MOV (B) .....	143
2.2.39 (8)	MOV (W) .....	145
2.2.39 (9)	MOV (L) .....	147
2.2.40	MOVFPE .....	149
2.2.41	MOVTPE .....	150
2.2.42 (1)	MULXS (B) .....	151
2.2.42 (2)	MULXS (W) .....	152
2.2.43 (1)	MULXU (B) .....	153
2.2.43 (2)	MULXU (W) .....	154
2.2.44 (1)	NEG (B) .....	155
2.2.44 (2)	NEG (W) .....	156
2.2.44 (3)	NEG (L) .....	157
2.2.45	NOP .....	158
2.2.46 (1)	NOT (B) .....	159
2.2.46 (2)	NOT (W) .....	160
2.2.46 (3)	NOT (L) .....	161
2.2.47 (1)	OR (B) .....	162
2.2.47 (2)	OR (W) .....	163
2.2.47 (3)	OR (L) .....	164
2.2.48 (1)	ORC .....	165
2.2.48 (2)	ORC .....	166
2.2.49 (1)	POP (W) .....	167
2.2.49 (2)	POP (L) .....	168
2.2.50 (1)	PUSH (W) .....	169
2.2.50 (2)	PUSH (L) .....	170
2.2.51 (1)	ROTL (B) .....	171

2.2.51 (2) ROTL (B).....	172
2.2.51 (3) ROTL (W) .....	173
2.2.51 (4) ROTL (W) .....	174
2.2.51 (5) ROTL (L).....	175
2.2.51 (6) ROTL (L).....	176
2.2.52 (1) ROTR (B) .....	177
2.2.52 (2) ROTR (B) .....	178
2.2.52 (3) ROTR (W) .....	179
2.2.52 (4) ROTR (W) .....	180
2.2.52 (5) ROTR (L).....	181
2.2.52 (6) ROTR (L).....	182
2.2.53 (1) ROTXL (B).....	183
2.2.53 (2) ROTXL (B).....	184
2.2.53 (3) ROTXL (W).....	185
2.2.53 (4) ROTXL (W).....	186
2.2.53 (5) ROTXL (L).....	187
2.2.53 (6) ROTXL (L).....	188
2.2.54 (1) ROTXR (B) .....	189
2.2.54 (2) ROTXR (B) .....	190
2.2.54 (3) ROTXR (W) .....	191
2.2.54 (4) ROTXR (W) .....	192
2.2.54 (5) ROTXR (L).....	193
2.2.54 (6) ROTXR (L).....	194
2.2.55 RTE .....	195
2.2.56 RTS .....	197
2.2.57 (1) SHAL (B).....	198
2.2.57 (2) SHAL (B).....	199
2.2.57 (3) SHAL (W) .....	200
2.2.57 (4) SHAL (W) .....	201
2.2.57 (5) SHAL (L).....	202
2.2.57 (6) SHAL (L).....	203
2.2.58 (1) SHAR (B) .....	204
2.2.58 (2) SHAR (B) .....	205
2.2.58 (3) SHAR (W) .....	206
2.2.58 (4) SHAR (W) .....	207
2.2.58 (5) SHAR (L).....	208
2.2.58 (6) SHAR (L).....	209
2.2.59 (1) SHLL (B) .....	210
2.2.59 (2) SHLL (B) .....	211
2.2.59 (3) SHLL (W).....	212
2.2.59 (4) SHLL (W).....	213

2.2.59 (5) SHLL (L) .....	214
2.2.59 (6) SHLL (L) .....	215
2.2.60 (1) SHLR (B).....	216
2.2.60 (2) SHLR (B).....	217
2.2.60 (3) SHLR (W).....	218
2.2.60 (4) SHLR (W).....	219
2.2.60 (5) SHLR (L).....	220
2.2.60 (6) SHLR (L).....	221
2.2.61 SLEEP.....	222
2.2.62 (1) STC (B).....	223
2.2.62 (2) STC (B).....	224
2.2.62 (3) STC (W).....	225
2.2.62 (4) STC (W).....	227
2.2.63 STM.....	229
2.2.64 STMAC.....	231
2.2.65 (1) SUB (B) .....	232
2.2.65 (2) SUB (W) .....	234
2.2.65 (3) SUB (L) .....	235
2.2.66 SUBS .....	236
2.2.67 SUBX.....	237
2.2.68 TAS .....	238
2.2.69 TRAPA.....	239
2.2.70 (1) XOR (B).....	241
2.2.70 (2) XOR (W) .....	242
2.2.70 (3) XOR (L).....	243
2.2.71 (1) XORC .....	244
2.2.71 (2) XORC .....	245
2.3 Instruction Set Summary .....	246
2.3.1 Instructions and Addressing Modes.....	246
2.3.2 Instruction Set.....	248
2.4 Instruction Codes .....	264
2.5 Operation Code Map.....	275
2.6 Number of States Required for Instruction Execution .....	279
2.7 Condition Code Modification .....	290
2.8 Bus States During Instruction Execution.....	295
<b>Section 3 Processing States.....</b>	<b>309</b>
3.1 Overview.....	309
3.2 Reset State .....	310
3.3 Exception-Handling State.....	311
3.3.1 Types of Exception Handling and Their Priority.....	311



3.3.2	Reset Exception Handling .....	312
3.3.3	Trace .....	312
3.3.4	Interrupt Exception Handling and Trap Instruction Exception Handling.....	312
3.4	Program Execution State .....	313
3.5	Bus-Released State .....	314
3.6	Power-Down State .....	314
3.6.1	Sleep Mode .....	314
3.6.2	Software Standby Mode .....	314
3.6.3	Hardware Standby Mode .....	314
Section 4 Basic Timing .....		315
4.1	Overview.....	315
4.2	On-Chip Memory (ROM, RAM).....	315
4.3	On-Chip Supporting Module Access Timing .....	317
4.4	External Address Space Access Timing .....	318



# Section 1 CPU

## 1.1 Overview

The H8S/2600 CPU and the H8S/2000 CPU are high-speed central processing units with a common an internal 32-bit architecture. Each CPU is upward-compatible with the H8/300 and H8/300H CPUs. The H8S/2600 CPU and H8S/2000 CPU have sixteen 16-bit general registers, can address a 4-Gbyte linear address space, and are ideal for realtime control.

### 1.1.1 Features

The H8S/2600 CPU and H8S/2000 CPU have the following features.

- Upward-compatible with H8/300 and H8/300H CPUs
  - Can execute H8/300 and H8/300H object programs
- General-register architecture
  - Sixteen 16-bit general registers (also usable as sixteen 8-bit registers or eight 32-bit registers)
- Sixty-nine basic instructions (H8S/2000 CPU has sixty-five)
  - 8/16/32-bit arithmetic and logic instructions
  - Multiply and divide instructions
  - Powerful bit-manipulation instructions
  - Multiply-and-accumulate instruction (H8S/2600 CPU only)
- Eight addressing modes
  - Register direct [Rn]
  - Register indirect [@ERn]
  - Register indirect with displacement [@(d:16,ERn) or @(d:32,ERn)]
  - Register indirect with post-increment or pre-decrement [@ERn+ or @-ERn]
  - Absolute address [@aa:8, @aa:16, @aa:24, or @aa:32]
  - Immediate [#xx:8, #xx:16, or #xx:32]
  - Program-counter relative [@(d:8,PC) or @(d:16,PC)]
  - Memory indirect [@@aa:8]
- 4-Gbyte address space
  - Program: 16 Mbytes
  - Data: 4 Gbytes

- High-speed operation
  - All frequently-used instructions execute in one or two states
  - Maximum clock frequency: 20 MHz
  - 8/16/32-bit register-register add/subtract : 50 ns
  - $8 \times 8$ -bit register-register multiply : 150 ns (H8S/2000 CPU: 600 ns)
  - $16 \div 8$ -bit register-register divide : 600 ns
  - $16 \times 16$ -bit register-register multiply : 200 ns (H8S/2000 CPU: 1000 ns)
  - $32 \div 16$ -bit register-register divide : 1000 ns
- Two CPU operating modes
  - Normal mode
  - Advanced mode
- Power-down modes
  - Transition to power-down state by SLEEP instruction
  - CPU clock speed selection

### 1.1.2 Differences between H8/2600 CPU and H8S/2000 CPU

Differences between the H8S/2600 CPU and the H8S/2000 CPU are as follows.

- Register configuration
  - The MAC register is supported only by the H8S/2600 CPU.  
For details, see section 1.4, Register Configuration.
- Basic instructions
  - The MAC, CLRMAC, LDMAC, and STMAC instructions are supported only by the H8S/2600 CPU.  
For details, see section 1.6, Instruction Set, and Section 2, Instruction Descriptions.
- Number of states required for execution
  - The number of states required for execution of the MULXU and MULXS instructions  
For details, see section 2.6, Number of States Required for Execution.

In addition, there may be differences in address spaces, EXR register functions, power-down states, and so on. For details, refer to the relevant microcontroller hardware manual.

### 1.1.3 Differences from H8/300 CPU

In comparison with the H8/300 CPU, the H8S/2600 CPU and H8S/2000 CPU have the following enhancements.

- More general registers and control registers
  - Eight 16-bit registers and one 8-bit control register have been added.
- Expanded address space
  - Normal mode supports the same 64-kbyte address space as the H8/300 CPU.
  - Advanced mode supports a maximum 4-Gbyte address space.
- Enhanced addressing
  - The addressing modes have been enhanced to make effective use of the 4-Gbyte address space.
- Enhanced instructions
  - Addressing modes of bit-manipulation instructions have been enhanced.
  - Signed multiply and divide instructions have been added.
  - A multiply-and-accumulate instruction has been added. (H8S/2600CPU only)
  - Two-bit shift and rotate instructions have been added.
  - Instructions for saving and restoring multiple registers have been added.
  - A test and set instruction has been added.
- Higher speed
  - Basic instructions execute twice as fast.

### 1.1.4 Differences from H8/300H CPU

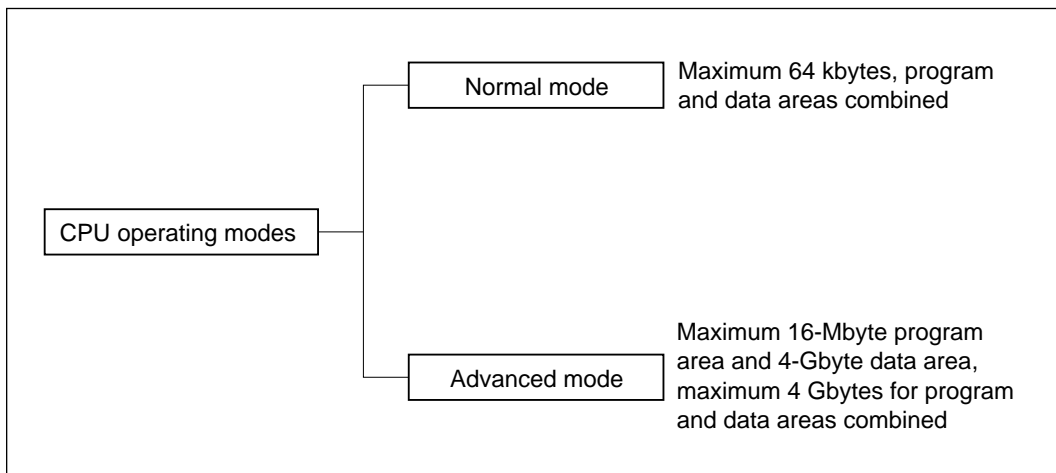
In comparison with the H8/300H CPU, the H8S/2600 CPU and H8S/2000 CPU have the following enhancements.

- Additional control register
  - One 8-bit control register has been added.
- Expanded address space
  - Advanced mode supports a maximum 4-Gbyte data address space.

- Enhanced instructions
  - Addressing modes of bit-manipulation instructions have been enhanced.
  - A multiply-and-accumulate instruction has been added (H8S/2600 CPU only).
  - Two-bit shift and rotate instructions have been added.
  - Instructions for saving and restoring multiple registers have been added.
  - A test and set instruction has been added.
- Higher speed
  - Basic instructions execute twice as fast.

## 1.2 CPU Operating Modes

Like the H8/300H CPU, the H8S/2600 CPU has two operating modes: normal and advanced. Normal mode supports a maximum 64-kbyte address space. Advanced mode supports a maximum 4-Gbyte total address space, of which up to 16 Mbytes can be used for program code and up to 4 Gbytes for data. The mode is selected with the mode pins of the microcontroller. For further information, refer to the relevant microcontroller hardware manual.



**Figure 1-1 CPU Operating Modes**

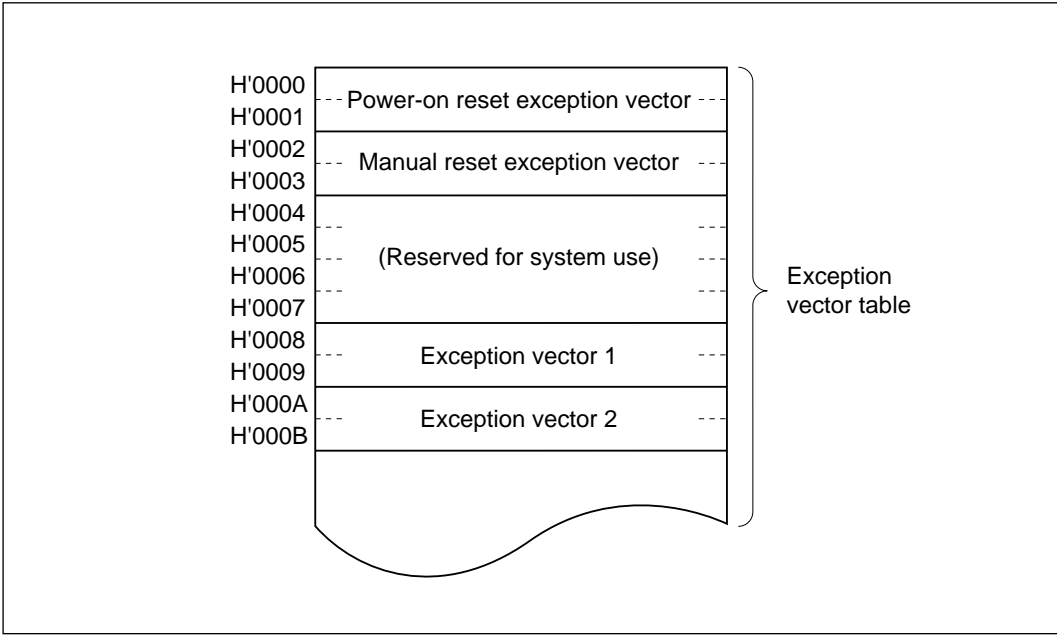
**(1) Normal Mode:** The exception vector table and stack have the same structure as in the H8/300 CPU.

**Address Space:** A maximum address space of 64 kbytes can be accessed, as in the H8/300 CPU.

**Extended Registers (En):** The extended registers (E0 to E7) can be used as 16-bit registers, or as the upper 16-bit segments of 32-bit registers. When En is used as a 16-bit register it can contain any value, even when the corresponding general register (R0 to R7) is used as an address register. If the general register is referenced in the register indirect addressing mode with pre-decrement (@-Rn) or post-increment (@Rn+) and a carry or borrow occurs, however, the value in the corresponding extended register will be affected.

**Instruction Set:** All additional instructions and addressing modes not found in the H8/300 CPU can be used. Only the lower 16 bits of effective addresses (EA) are valid.

**Exception Vector Table and Memory Indirect Branch Addresses:** In normal mode the top area starting at H'0000 is allocated to the exception vector table. One branch address is stored per 16 bits (figure 1-2). The exception vector table differs depending on the microcontroller. Refer to the relevant microcontroller hardware manual for further information.

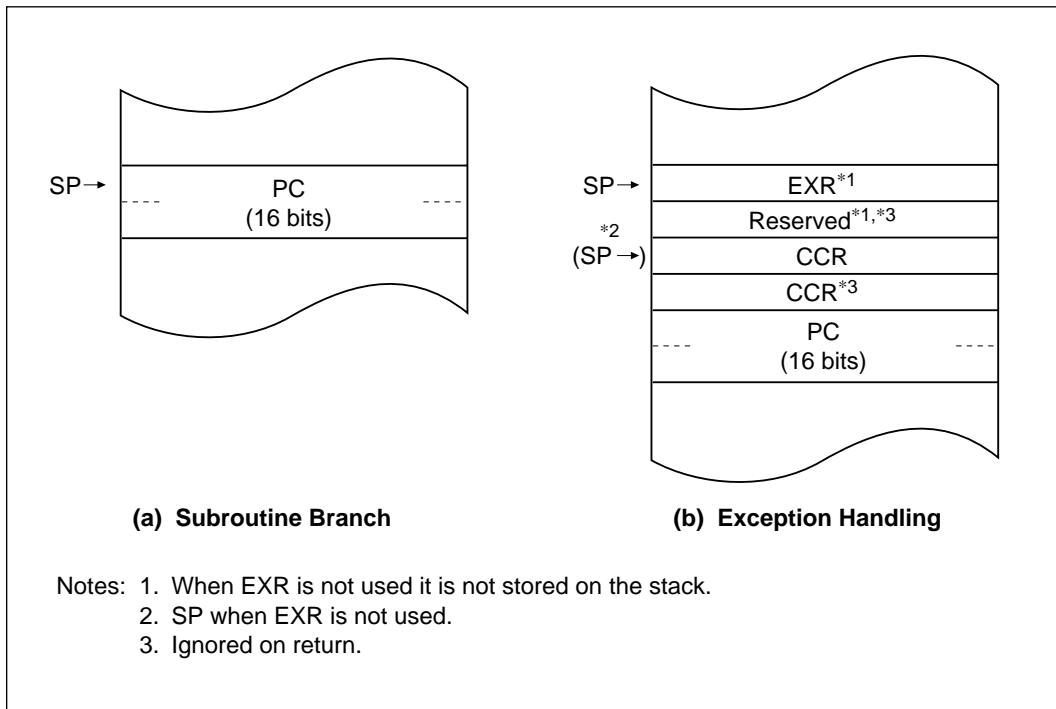


**Figure 1-2 Exception Vector Table (Normal Mode)**

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address included in the instruction code to specify a memory operand that contains a branch address. In normal mode the operand is a 16-bit word operand, providing a 16-bit branch address. Branch addresses can be stored in the top area from H'0000 to H'00FF. Note that this area is also used for the exception vector table.



**Stack Structure:** When the program counter (PC) is pushed onto the stack in a subroutine call, and the PC, condition-code register (CCR), and extended control register (EXR) are pushed onto the stack in exception handling, they are stored as shown in figure 1-3. When EXR is invalid, it is not pushed onto the stack. For details, see the relevant hardware manual.



**Figure 1-3 Stack Structure in Normal Mode**

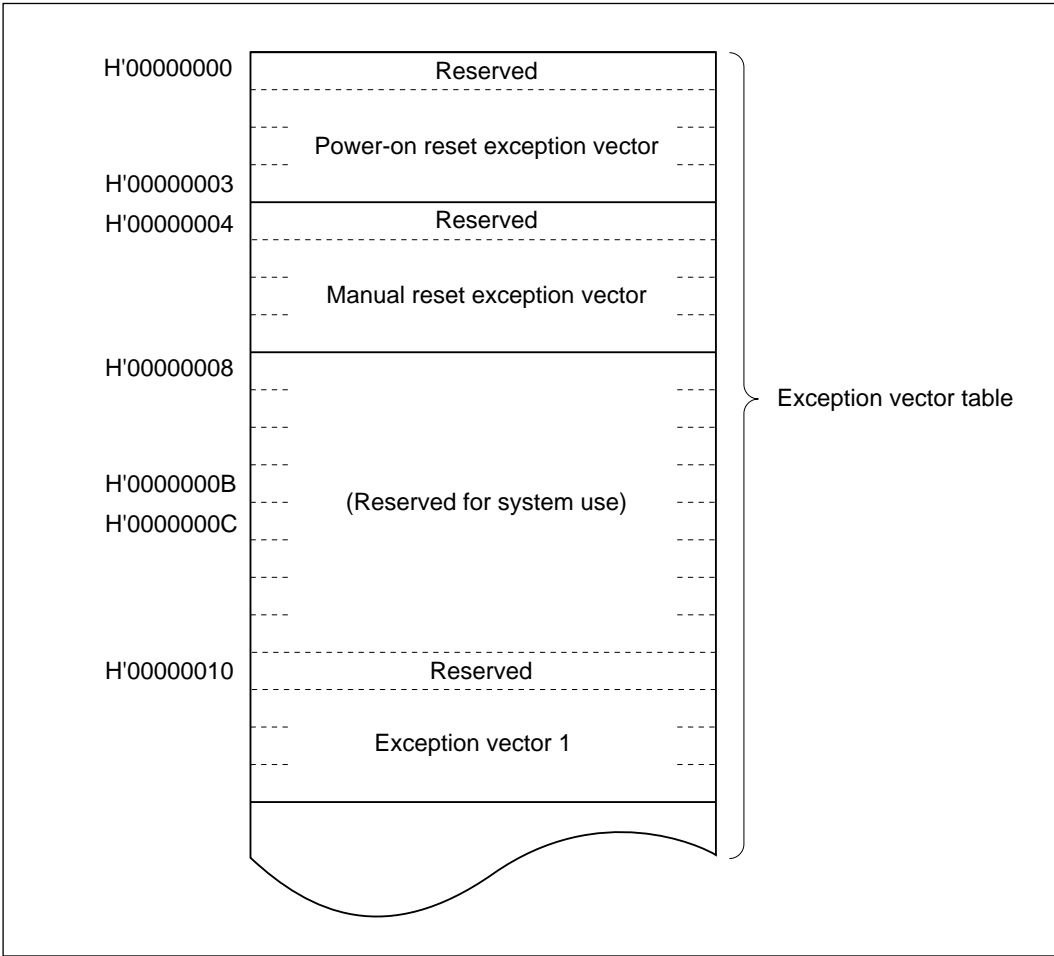
**(2) Advanced Mode:** In advanced mode the data address space is larger than for the H8/300H CPU.

**Address Space:** The 4-Gbyte maximum address space provides linear access to a maximum 16 Mbytes of program code and maximum 4 Gbytes of data.

**Extended Registers (En):** The extended registers (E0 to E7) can be used as 16-bit registers, or as the upper 16-bit segments of 32-bit registers or address registers.

**Instruction Set:** All instructions and addressing modes can be used.

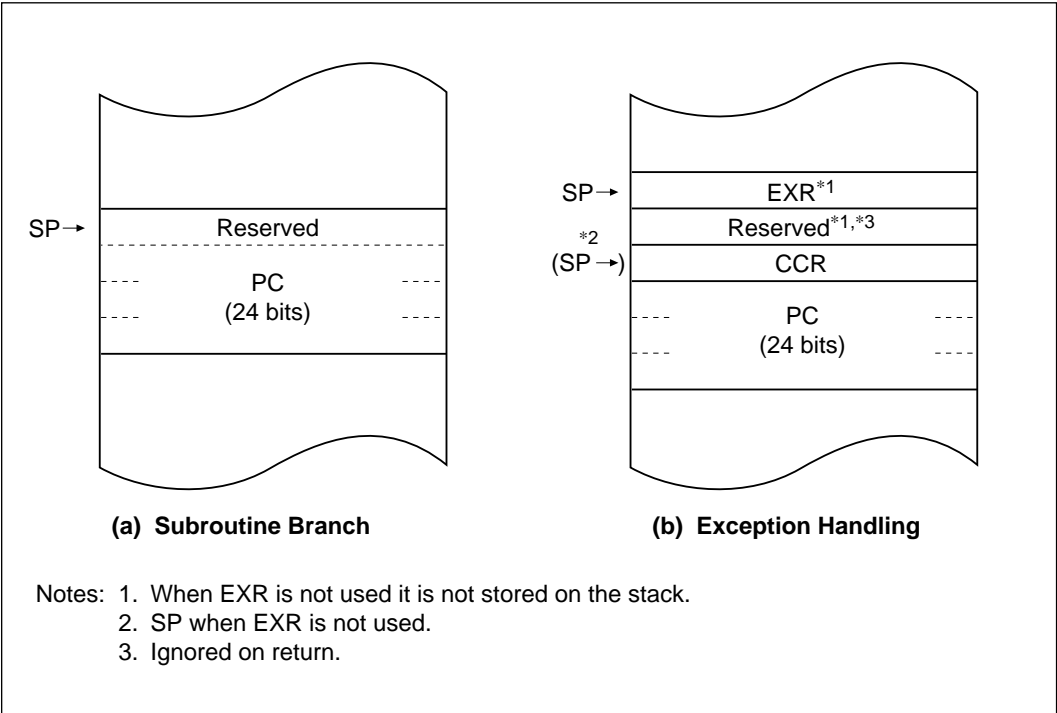
**Exception Vector Table and Memory Indirect Branch Addresses:** In advanced mode the top area starting at H'00000000 is allocated to the exception vector table in units of 32 bits. In each 32 bits, the upper 8 bits are ignored and a branch address is stored in the lower 24 bits (figure 1-4). The exception vector table differs depending on the microcontroller. Refer to the relevant microcontroller hardware manual for further information.



**Figure 1-4 Exception Vector Table (Advanced Mode)**

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address included in the instruction code to specify a memory operand that contains a branch address. In advanced mode the operand is a 32-bit longword operand, providing a 32-bit branch address. The upper 8 bits of these 32 bits are a reserved area that is regarded as H'00. Branch addresses can be stored in the top area from H'00000000 to H'000000FF. Note that this area is also used for the exception vector table.

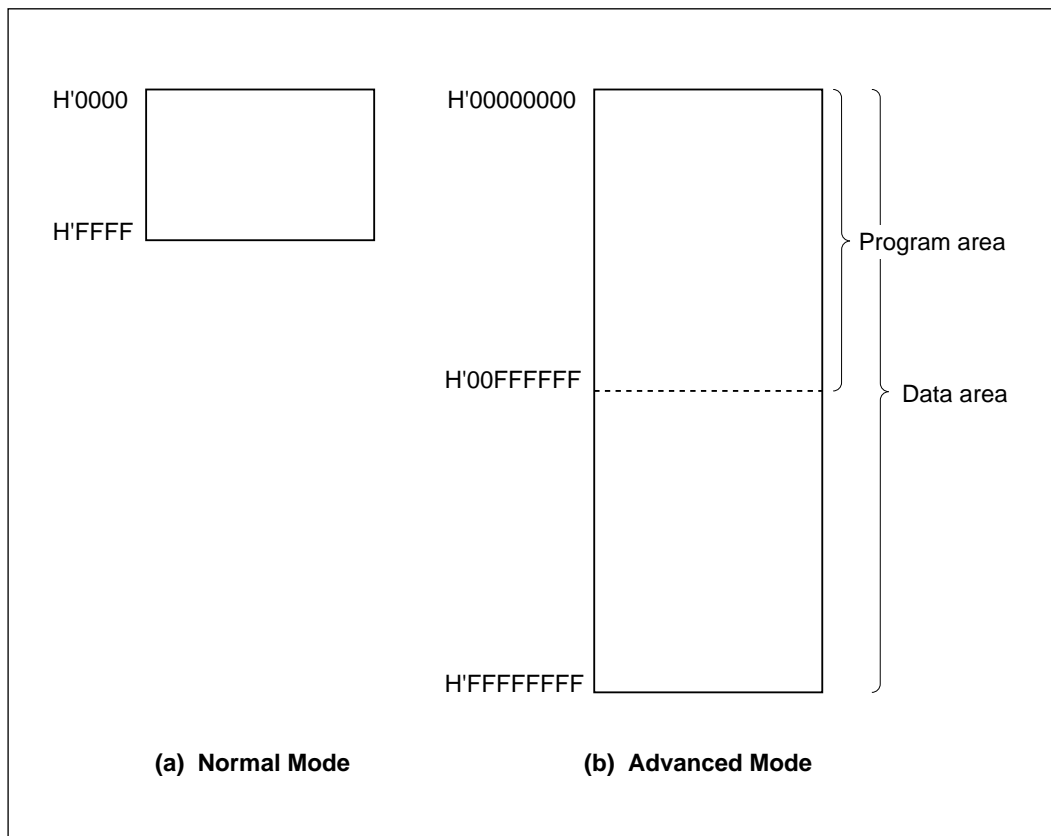
**Stack Structure:** In advanced mode, when the program counter (PC) is pushed onto the stack in a subroutine call, and the PC, condition-code register (CCR), and extended control register (EXR) are pushed onto the stack in exception handling, they are stored as shown in figure 1-5. When EXR is invalid, it is not pushed onto the stack. For details, see the relevant hardware manual.



**Figure 1-5 Stack Structure in Advanced Mode**

### 1.3 Address Space

Figure 1-6 shows a memory map of the H8S/2600 CPU. The H8S/2600 CPU provides linear access to a maximum 64-kbyte address space in normal mode, and a maximum 4-Gbyte address space in advanced mode. The address space differs depending on the operating mode. For details, refer to the relevant microcontroller hardware manual.

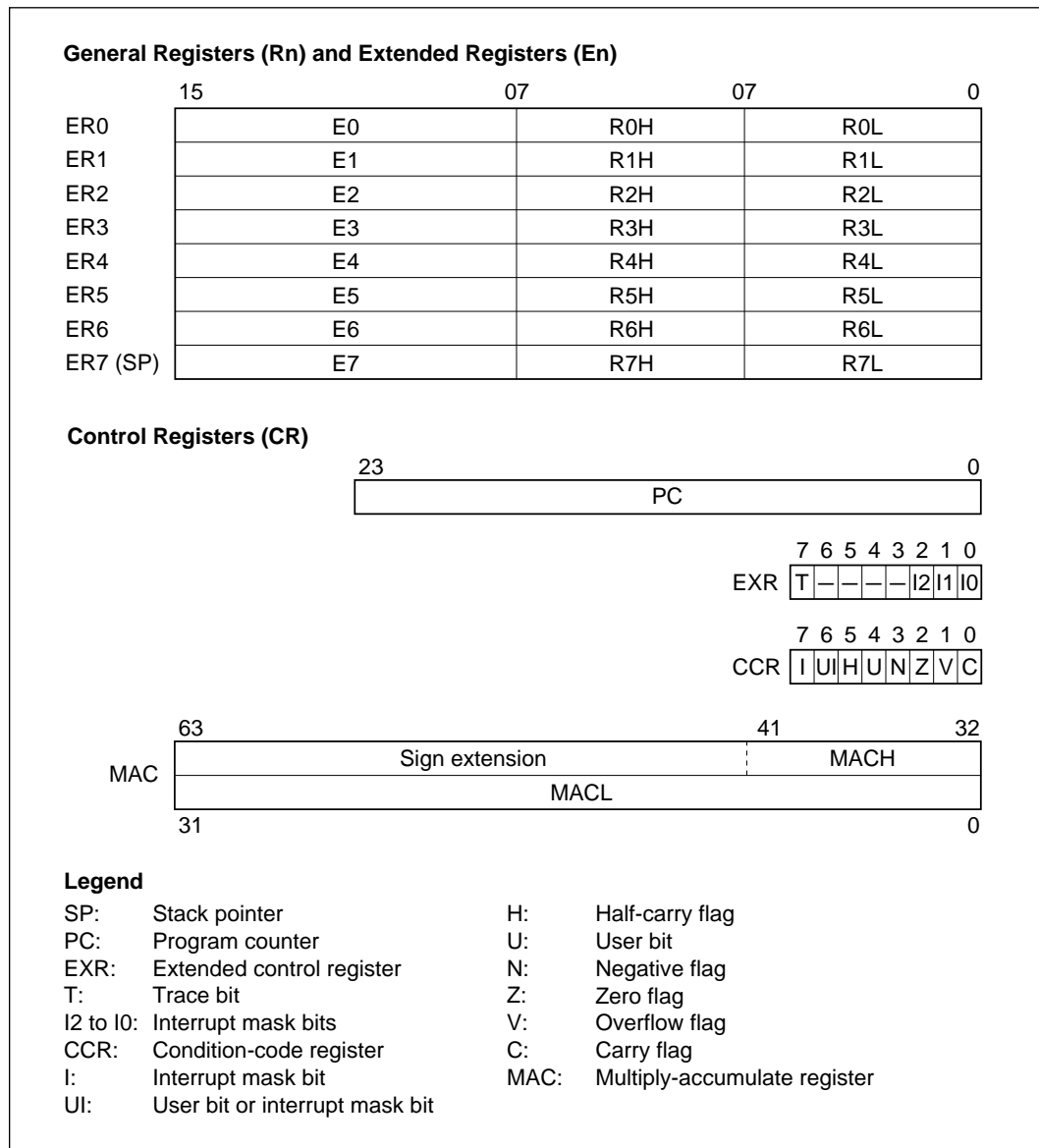


**Figure 1-6 Memory Map**

## 1.4 Register Configuration

### 1.4.1 Overview

The CPUs have the internal registers shown in figure 1-7. There are two types of registers: general registers and control registers. The H8S/2000 CPU does not support the MAC register.



**Figure 1-7 CPU Registers**

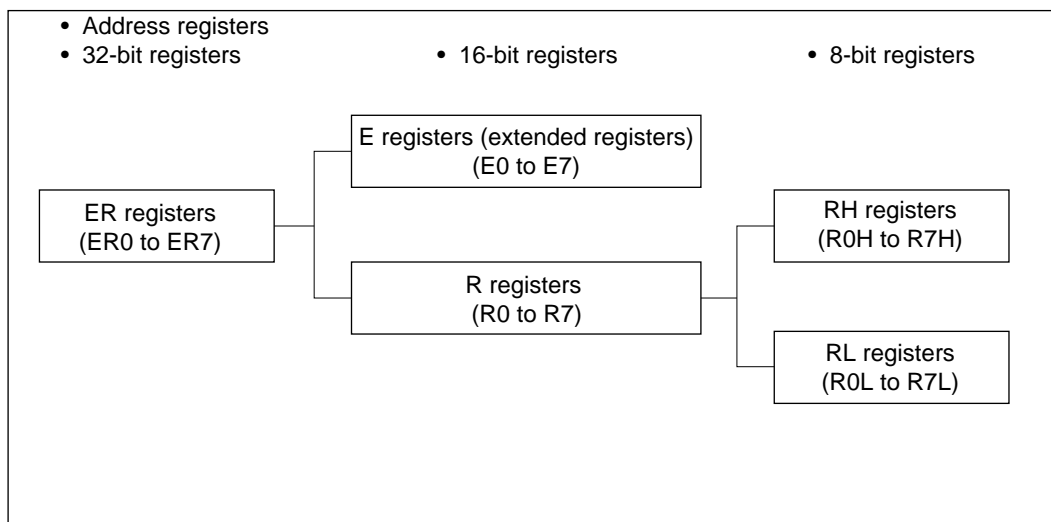
### 1.4.2 General Registers

The CPUs have eight 32-bit general registers. These general registers are all functionally alike and can be used as both address registers and data registers. When a general register is used as a data register, it can be accessed as a 32-bit, 16-bit, or 8-bit register. When the general registers are used as 32-bit registers or address registers, they are designated by the letters ER (ER0 to ER7).

The ER registers divide into 16-bit general registers designated by the letters E (E0 to E7) and R (R0 to R7). These registers are functionally equivalent, providing a maximum sixteen 16-bit registers. The E registers (E0 to E7) are also referred to as extended registers.

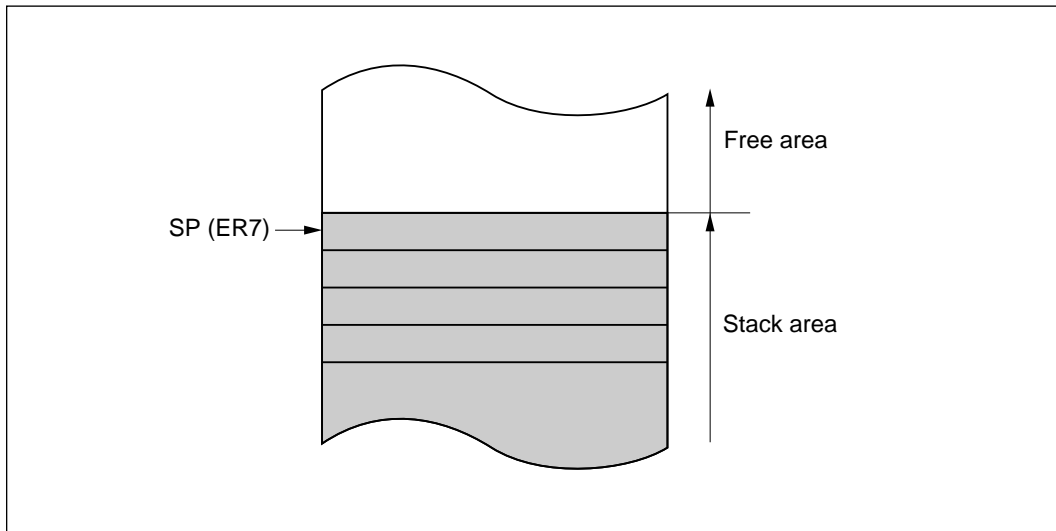
The R registers divide into 8-bit general registers designated by the letters RH (R0H to R7H) and RL (R0L to R7L). These registers are functionally equivalent, providing a maximum sixteen 8-bit registers.

Figure 1-8 illustrates the usage of the general registers. The usage of each register can be selected independently.



**Figure 1-8 Usage of General Registers**

General register ER7 has the function of stack pointer (SP) in addition to its general-register function, and is used implicitly in exception handling and subroutine calls. Figure 1-9 shows the stack.



**Figure 1-9 Stack**

### 1.4.3 Control Registers

The control registers are the 24-bit program counter (PC), 8-bit extended control register (EXR), 8-bit condition-code register (CCR), and 64-bit multiply-accumulate register (MAC).

**(1) Program Counter (PC):** This 24-bit counter indicates the address of the next instruction the CPU will execute. The length of all CPU instructions is 16 bits (one word) or a multiple of 16 bits, so the least significant PC bit is ignored. When an instruction is fetched, the least significant PC bit is regarded as 0.

**(2) Extended Control Register (EXR):** This 8-bit register contains the trace bit (T) and three interrupt mask bits (I2 to I0).

**Bit 7—Trace Bit (T):** Selects trace mode. When this bit is cleared to 0, instructions are executed in sequence. When this bit is set to 1, a trace exception is generated each time an instruction is executed.

**Bits 6 to 3—Reserved:** These bits are reserved.

**Bits 2 to 0—Interrupt Mask Bits (I2 to I0):** These bits designate the interrupt mask level (0 to 7). For details refer to the relevant microcontroller hardware manual.

Operations can be performed on the EXR bits by the LDC, STC, ANDC, ORC, and XORC instructions. All interrupts, including NMI, are disabled for three states after one of these instructions is executed, except for STC.

**(3) Condition-Code Register (CCR):** This 8-bit register contains internal CPU status information, including an interrupt mask bit (I) and half-carry (H), negative (N), zero (Z), overflow (V), and carry (C) flags.

**Bit 7—Interrupt Mask Bit (I):** Masks interrupts other than NMI when set to 1. (NMI is accepted regardless of the I bit setting.) The I bit is set to 1 by hardware at the start of an exception-handling sequence.

**Bit 6—User Bit or Interrupt Mask Bit (UI):** Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions. This bit can also be used as an interrupt mask bit. For details refer to the relevant microcontroller hardware manual.

**Bit 5—Half-Carry Flag (H):** When the ADD.B, ADDX.B, SUB.B, SUBX.B, CMP.B, or NEG.B instruction is executed, this flag is set to 1 if there is a carry or borrow at bit 3, and cleared to 0 otherwise. When the ADD.W, SUB.W, CMP.W, or NEG.W instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 11, and cleared to 0 otherwise. When the ADD.L, SUB.L, CMP.L, or NEG.L instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 27, and cleared to 0 otherwise.

**Bit 4—User Bit (U):** Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions.

**Bit 3—Negative Flag (N):** Stores the value of the most significant bit (sign bit) of data.

**Bit 2—Zero Flag (Z):** Set to 1 to indicate zero data, and cleared to 0 to indicate non-zero data.

**Bit 1—Overflow Flag (V):** Set to 1 when an arithmetic overflow occurs, and cleared to 0 at other times.

**Bit 0—Carry Flag (C):** Set to 1 when a carry occurs, and cleared to 0 otherwise. Used by:

- Add instructions, to indicate a carry
- Subtract instructions, to indicate a borrow
- Shift and rotate instructions, to store the value shifted out of the end bit

The carry flag is also used as a bit accumulator by bit manipulation instructions.

Some instructions leave some or all of the flag bits unchanged. For the action of each instruction on the flag bits, refer to the detailed descriptions of the instructions starting in section 2.2.1.



Operations can be performed on the CCR bits by the LDC, STC, ANDC, ORC, and XORC instructions. The N, Z, V, and C flags are used as branching conditions for conditional branch (Bcc) instructions.

**(4) Multiply-Accumulate Register (MAC):** The MAC register is supported only by the H8S/2600 CPU. This 64-bit register stores the results of multiply-and-accumulate operations. It consists of two 32-bit registers denoted MACH and MACL. The lower 10 bits of MACH are valid; the upper bits are a sign extension.

#### **1.4.4 Initial Register Values**

Reset exception handling loads the CPU's program counter (PC) from the vector table, clears the trace bit in EXR to 0, and sets the interrupt mask bits in CCR and EXR to 1. The other CCR bits and the general registers are not initialized. In particular, the stack pointer (ER7) is not initialized. The stack pointer should therefore be initialized by an MOV.L instruction executed immediately after a reset.

## 1.5 Data Formats

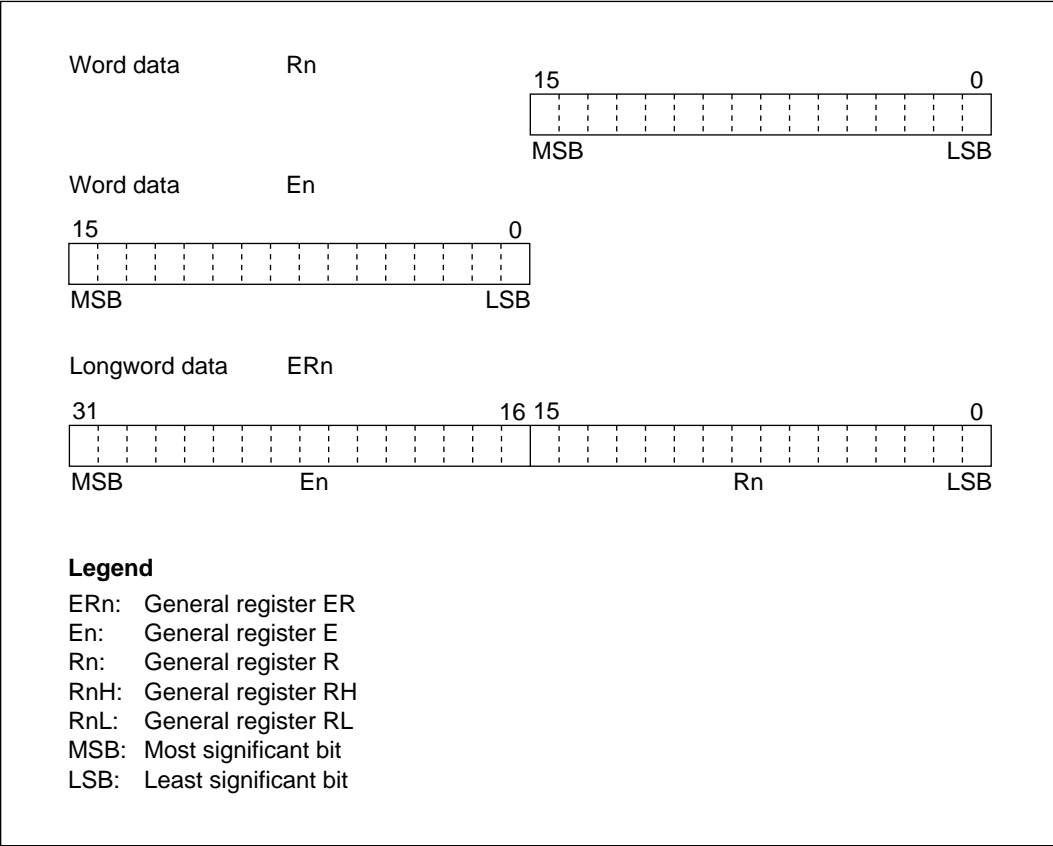
The CPUs can process 1-bit, 4-bit (BCD), 8-bit (byte), 16-bit (word), and 32-bit (longword) data. Bit-manipulation instructions operate on 1-bit data by accessing bit  $n$  ( $n = 0, 1, 2, \dots, 7$ ) of byte operand data. The DAA and DAS decimal-adjust instructions treat byte data as two digits of 4-bit BCD data.

### 1.5.1 General Register Data Formats

Figure 1-10 shows the data formats in general registers.

Data Type	Register Number	Data Format
1-bit data	RnH	<div> <div>7</div> <div>0</div> <div>7</div><div>6</div><div>5</div><div>4</div><div>3</div><div>2</div><div>1</div><div>0</div> <div>Don't care</div> </div>
1-bit data	RnL	<div> <div>7</div> <div>0</div> <div>Don't care</div> <div>7</div><div>6</div><div>5</div><div>4</div><div>3</div><div>2</div><div>1</div><div>0</div> </div>
4-bit BCD data	RnH	<div> <div>7</div> <div>4</div><div>3</div> <div>0</div> <div>Upper</div><div>Lower</div> <div>Don't care</div> </div>
4-bit BCD data	RnL	<div> <div>7</div> <div>4</div><div>3</div> <div>0</div> <div>Don't care</div> <div>Upper</div><div>Lower</div> </div>
Byte data	RnH	<div> <div>7</div> <div>0</div> <div>MSB</div><div>LSB</div> <div>Don't care</div> </div>
Byte data	RnL	<div> <div>7</div> <div>0</div> <div>Don't care</div> <div>MSB</div><div>LSB</div> </div>

**Figure 1-10 General Register Data Formats**



**Figure 1-10 General Register Data Formats (cont)**

1.5.2 Memory Data Formats

Figure 1-11 shows the data formats in memory. The CPU can access word data and longword data in memory, but word or longword data must begin at an even address. If an attempt is made to access word or longword data at an odd address, no address error occurs but the least significant bit of the address is regarded as 0, so the access starts at the preceding address. This also applies to instruction fetches.

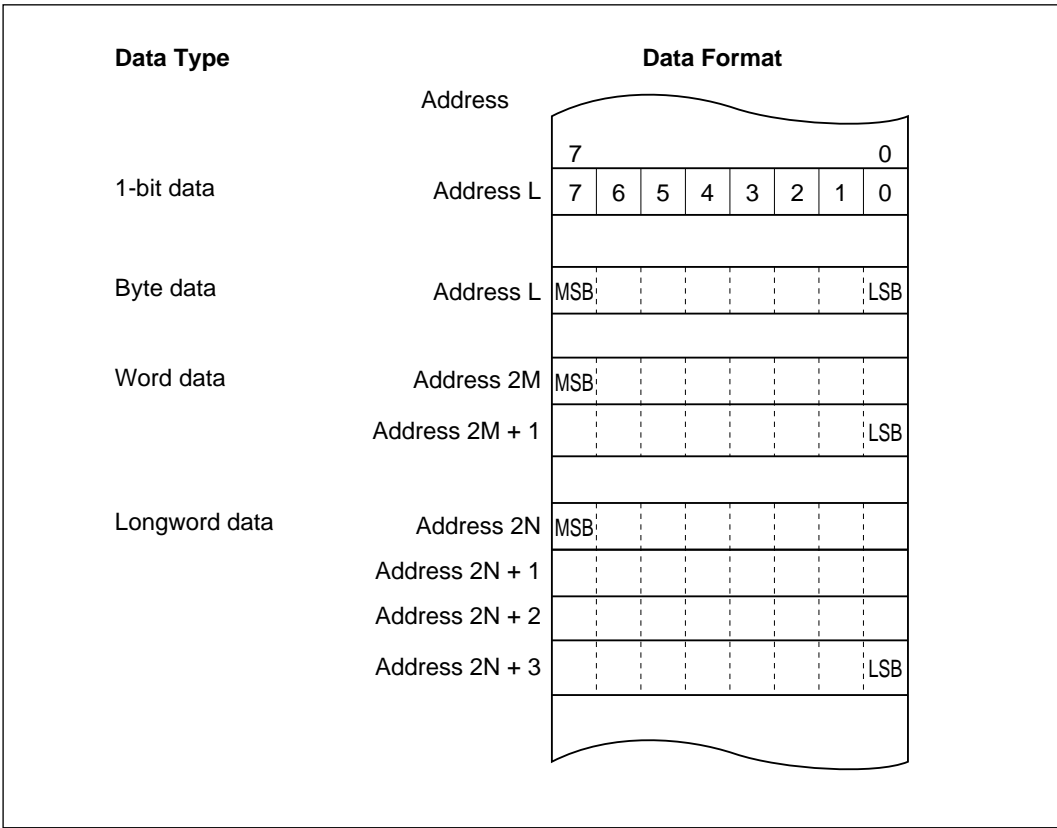


Figure 1-11 Memory Data Formats

When the stack pointer (ER7) is used as an address register to access the stack, the operand size should be word size or longword size.

## 1.6 Instruction Set

### 1.6.1 Overview

The H8S/2600 CPU has 69 types of instructions, while the H8S/2000 CPU has 65 types. The instructions are classified by function as shown in table 1-1. For a detailed description of each instruction, see section 2.2, Instruction Descriptions.

**Table 1-1 Instruction Classification**

Function	Instructions	Size	Types
Data transfer	MOV	BWL	5
	POP*2, PUSH*2	WL	
	LDM, STM	L	
	MOVFPE, MOVTPE	B	
Arithmetic operations	ADD, SUB, CMP, NEG	BWL	19
	ADDX, SUBX, DAA, DAS	B	
	INC, DEC	BWL	
	ADDS, SUBS	L	
	MULXU, DIVXU, MULXS, DIVXS	BW	
	EXTU, EXTS	WL	
	TAS	B	
	MAC, LDMAC, STMAC, CLRMAC*1	—	4*1
Logic operations	AND, OR, XOR, NOT	BWL	4
Shift	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	BWL	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BLD, BILD, BST, BIST, BAND, BIAN, BOR, BIOR, BXOR, BIXOR	B	14
Branch	Bcc*3, JMP, BSR, JSR, RTS	—	5
System control	TRAPA, RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	—	9
Block data transfer	EEPMOV	—	1

H8S/2600 CPU: Total 69 types      H8S/2000 CPU: Total 65 types

Notes: B—byte size; W—word size; L—longword size.

1. The MAC, LDMAC, STMAC, and CLRMAC instructions are supported only by the H8S/2600 CPU.
2. POP.W Rn and PUSH.W Rn are identical to MOV.W @SP+, Rn and MOV.W Rn, @-SP. POP.L ERn and PUSH.L ERn are identical to MOV.L @SP+, ERn and MOV.L ERn, @-SP.
3. Bcc is the generic designation of a conditional branch instruction.

### 1.6.2 Instructions and Addressing Modes

Table 1-2 indicates the combinations of instructions and addressing modes that the H8S/2600 CPU and H8S/2000 CPU can use.

**Table 1-2 Combinations of Instructions and Addressing Modes**

Function	Instruction	Addressing Modes													
		#xx	Rn	@ ERn	@ (d:16,ERn)	@ (d:32,ERn)	@ -ERn/@ ERn+	@ aa:8	@ aa:16	@ aa:24	@ aa:32	@ (d:8,PC)	@ (d:16,PC)	@ @ aa:8	
Data transfer	MOV	BWL	BWL	BWL	BWL	BWL	BWL	B	BWL	—	BWL	—	—	—	—
	POP, PUSH	—	—	—	—	—	—	—	—	—	—	—	—	—	WL
	LDM, STM	—	—	—	—	—	—	—	—	—	—	—	—	—	L
	MOVEPE, MOVTPE	—	—	—	—	—	—	—	B	—	—	—	—	—	—
Arithmetic operations	ADD, CMP	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	SUB	WL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	ADDX, SUBX	B	B	—	—	—	—	—	—	—	—	—	—	—	—
	ADDS, SUBS	—	L	—	—	—	—	—	—	—	—	—	—	—	—
	INC, DEC	WL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	DAA, DAS	—	B	—	—	—	—	—	—	—	—	—	—	—	—
	MULXU, DIVXU	—	BW	—	—	—	—	—	—	—	—	—	—	—	—
	MULXS, DIVXS	—	BW	—	—	—	—	—	—	—	—	—	—	—	—
	NEG	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	EXTU, EXTS	—	WL	—	—	—	—	—	—	—	—	—	—	—	—
	TAS	—	—	B	—	—	—	—	—	—	—	—	—	—	—
	MAC*	—	—	—	—	—	○	—	—	—	—	—	—	—	—
	CLRMAC*	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDMAC*, STMAC*	—	L	—	—	—	—	—	—	—	—	—	—	—	—

Note: \*Supported only by the H8S/2600 CPU

**Table 1-2 Combinations of Instructions and Addressing Modes (cont)**

Function	Instruction	Addressing Modes													
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:32,ERn)	@-ERn/@ERn+	@aa:8	@aa:16	@aa:24	@aa:32	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Logic operations	AND, OR, XOR	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	NOT	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Shift		—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Bit manipulation		—	B	B	—	—	—	B	B	—	B	—	—	—	—
Branch	Bcc, BSR	—	—	—	—	—	—	—	—	—	—	○	○	—	—
	JMP, JSR	—	—	—	—	—	—	—	—	○	—	—	—	○	—
	RTS	—	—	—	—	—	—	—	—	—	—	—	—	—	○
System control	TRAPA	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	RTE	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	SLEEP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDC	B	B	W	W	W	W	—	W	—	W	—	—	—	—
	STC	—	B	W	W	W	W	—	W	—	W	—	—	—	—
	ANDC, ORC, XORC	B	—	—	—	—	—	—	—	—	—	—	—	—	—
	NOP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
Block data transfer		—	—	—	—	—	—	—	—	—	—	—	—	—	BW

**Legend**

B: Byte

W: Word

L: Longword

### 1.6.3 Table of Instructions Classified by Function

Table 1-3 summarizes the instructions in each functional category. The notation used in table 1-3 is defined next.

#### Operation Notation

Rd	General register (destination)*
Rs	General register (source)*
Rn	General register*
ERn	General register (32-bit register)
MAC	Multiply-accumulate register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
EXR	Extended control register
CCR	Condition-code register
N	N (negative) flag in CCR
Z	Z (zero) flag in CCR
V	V (overflow) flag in CCR
C	C (carry) flag in CCR
PC	Program counter
SP	Stack pointer
#IMM	Immediate data
disp	Displacement
+	Addition
–	Subtraction
×	Multiplication
÷	Division
^	Logical AND
∨	Logical OR
⊕	Logical exclusive OR
→	Move
¬	Logical not (logical complement)
:8/:16/:24/32	8-, 16-, 24-, or 32-bit length

Note: \* General registers include 8-bit registers (R0H to R7H, R0L to R7L), 16-bit registers (R0 to R7, E0 to E7), and 32-bit registers (ER0 to ER7).



**Table 1-3 Instructions Classified by Function**

Type	Instruction	Size*	Function
Data transfer	MOV	B/W/L	(EAs) → Rd, Rs → (EAd) Moves data between two general registers or between a general register and memory, or moves immediate data to a general register.
	MOVFP	B	(EAs) → Rd Moves external memory contents (addressed by @aa:16) to a general register in synchronization with an E clock.
	MOVTP	B	Rs → (EAs) Moves general register contents to an external memory location (addressed by @aa:16) in synchronization with an E clock.
	POP	W/L	@SP+ → Rn Pops a register from the stack. POP.W Rn is identical to MOV.W @SP+, Rn. POP.L ERn is identical to MOV.L @SP+, ERn.
	PUSH	W/L	Rn → @-SP Pushes a register onto the stack. PUSH.W Rn is identical to MOV.W Rn, @-SP. PUSH.L ERn is identical to MOV.L ERn, @-SP.
	LDM	L	@SP+ → Rn (register list) Pops two or more general registers from the stack.
	STM	L	Rn (register list) → @-SP Pushes two or more general registers onto the stack.

Note: \* Size refers to the operand size.

B: Byte  
W: Word  
L: Longword

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Arithmetic operations	ADD SUB	B/W/L	$Rd \pm Rs \rightarrow Rd$ , $Rd \pm \#IMM \rightarrow Rd$ Performs addition or subtraction on data in two general registers, or on immediate data and data in a general register. (Immediate byte data cannot be subtracted from byte data in a general register. Use the SUBX or ADD instruction.)
	ADDX SUBX	B	$Rd \pm Rs \pm C \rightarrow Rd$ , $Rd \pm \#IMM \pm C \rightarrow Rd$ Performs addition or subtraction with carry or borrow on byte data in two general registers, or on immediate data and data in a general register.
	INC DEC	B/W/L	$Rd \pm 1 \rightarrow Rd$ , $Rd \pm 2 \rightarrow Rd$ Increments or decrements a general register by 1 or 2. (Byte operands can be incremented or decremented by 1 only.)
	ADDS SUBS	L	$Rd \pm 1 \rightarrow Rd$ , $Rd \pm 2 \rightarrow Rd$ , $Rd \pm 4 \rightarrow Rd$ Adds or subtracts the value 1, 2, or 4 to or from data in a 32-bit register.
	DAA DAS	B	$Rd$ decimal adjust $\rightarrow Rd$ Decimal-adjusts an addition or subtraction result in a general register by referring to the CCR to produce 4-bit BCD data.
	MULXU	B/W	$Rd \times Rs \rightarrow Rd$ Performs unsigned multiplication on data in two general registers: either 8 bits $\times$ 8 bits $\rightarrow$ 16 bits or 16 bits $\times$ 16 bits $\rightarrow$ 32 bits.
	MULXS	B/W	$Rd \times Rs \rightarrow Rd$ Performs signed multiplication on data in two general registers: either 8 bits $\times$ 8 bits $\rightarrow$ 16 bits or 16 bits $\times$ 16 bits $\rightarrow$ 32 bits.
	DIVXU	B/W	$Rd \div Rs \rightarrow Rd$ Performs unsigned division on data in two general registers: either 16 bits $\div$ 8 bits $\rightarrow$ 8-bit quotient and 8-bit remainder or 32 bits $\div$ 16 bits $\rightarrow$ 16-bit quotient and 16-bit remainder.

Note: \* Size refers to the operand size.

B: Byte

W: Word

L: Longword

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Arithmetic operations	DIVXS	B/W	$Rd \div Rs \rightarrow Rd$ Performs signed division on data in two general registers: either 16 bits $\div$ 8 bits $\rightarrow$ 8-bit quotient and 8-bit remainder or 32 bits $\div$ 16 bits $\rightarrow$ 16-bit quotient and 16-bit remainder.
	CMP	B/W/L	$Rd - Rs$ , $Rd - \#IMM$ Compares data in a general register with data in another general register or with immediate data, and sets CCR bits according to the result.
	NEG	B/W/L	$0 - Rd \rightarrow Rd$ Takes the two's complement (arithmetic complement) of data in a general register.
	EXTU	W/L	$Rd$ (zero extension) $\rightarrow Rd$ Extends the lower 8 bits of a 16-bit register to word size, or the lower 16 bits of a 32-bit register to longword size, by padding with zeros on the left.
	EXTS	W/L	$Rd$ (sign extension) $\rightarrow Rd$ Extends the lower 8 bits of a 16-bit register to word size, or the lower 16 bits of a 32-bit register to longword size, by extending the sign bit.
	TAS	B	$@ERd - 0, 1 \rightarrow (<bit\ 7> \text{ of } @ERd)$ Tests memory contents, and sets the most significant bit (bit 7) to 1.
	MAC	—	$(EAs) \times (EAd) + MAC \rightarrow MAC$ Performs signed multiplication on memory contents and adds the result to the multiply-accumulate register. The following operations can be performed:  16 bits $\times$ 16 bits + 32 bits $\rightarrow$ 32 bits, saturating 16 bits $\times$ 16 bits + 42 bits $\rightarrow$ 42 bits, non-saturating Supported by H8S/2600 CPU only
	CLRMAC	—	$0 \rightarrow MAC$ Clears the multiply-accumulate register to zero. Supported by H8S/2600 CPU only
	LDMAC STMAC	L	$Rs \rightarrow MAC$ , $MAC \rightarrow Rd$ Transfers data between a general register and the multiply-accumulate register. Supported by H8S/2600 CPU only.

Note: \* Size refers to the operand size.

B: Byte

W: Word

L: Longword

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Logic operations	AND	B/W/L	$Rd \wedge Rs \rightarrow Rd$ , $Rd \wedge \#IMM \rightarrow Rd$ Performs a logical AND operation on a general register and another general register or immediate data.
	OR	B/W/L	$Rd \vee Rs \rightarrow Rd$ , $Rd \vee \#IMM \rightarrow Rd$ Performs a logical OR operation on a general register and another general register or immediate data.
	XOR	B/W/L	$Rd \oplus Rs \rightarrow Rd$ , $Rd \oplus \#IMM \rightarrow Rd$ Performs a logical exclusive OR operation on a general register and another general register or immediate data.
	NOT	B/W/L	$\neg (Rd) \rightarrow (Rd)$ Takes the one's complement of general register contents.
Shift operations	SHAL SHAR	B/W/L	$Rd \text{ (shift)} \rightarrow Rd$ Performs an arithmetic shift on general register contents. 1-bit or 2-bit shift is possible.
	SHLL SHLR	B/W/L	$Rd \text{ (shift)} \rightarrow Rd$ Performs a logical shift on general register contents. 1-bit or 2-bit shift is possible.
	ROTL ROTR	B/W/L	$Rd \text{ (rotate)} \rightarrow Rd$ Rotates general register contents. 1-bit or 2-bit rotation is possible.
	ROTXL ROTXR	B/W/L	$Rd \text{ (rotate)} \rightarrow Rd$ Rotates general register contents through the carry bit. 1-bit or 2-bit rotation is possible.

Note: \* Size refers to the operand size.

B: Byte

W: Word

L: Longword

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Bit-manipulation instructions	BSET	B	$1 \rightarrow (\text{<bit-No.> of <EAd>})$ Sets a specified bit in a general register or memory operand to 1. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BCLR	B	$0 \rightarrow (\text{<bit-No.> of <EAd>})$ Clears a specified bit in a general register or memory operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BNOT	B	$\neg (\text{<bit-No.> of <EAd>}) \rightarrow (\text{<bit-No.> of <EAd>})$ Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BTST	B	$\neg (\text{<bit-No.> of <EAd>}) \rightarrow Z$ Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BAND	B	$C \wedge (\text{<bit-No.> of <EAd>}) \rightarrow C$ ANDs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIAND	B	$C \wedge \neg (\text{<bit-No.> of <EAd>}) \rightarrow C$ ANDs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag.  The bit number is specified by 3-bit immediate data.
	BOR	B	$C \vee (\text{<bit-No.> of <EAd>}) \rightarrow C$ ORs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIOR	B	$C \vee \neg (\text{<bit-No.> of <EAd>}) \rightarrow C$ ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag.  The bit number is specified by 3-bit immediate data.

Note: \* Size refers to the operand size.

B: Byte

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Bit-manipulation instructions	BXOR	B	$C \oplus (\text{<bit-No.> of <EAd>}) \rightarrow C$ Exclusive-ORs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIXOR	B	$C \oplus \neg (\text{<bit-No.> of <EAd>}) \rightarrow C$ Exclusive-ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag. The bit number is specified by 3-bit immediate data.
	BLD	B	$(\text{<bit-No.> of <EAd>}) \rightarrow C$ Transfers a specified bit in a general register or memory operand to the carry flag.
	BILD	B	$\neg (\text{<bit-No.> of <EAd>}) \rightarrow C$ Transfers the inverse of a specified bit in a general register or memory operand to the carry flag. The bit number is specified by 3-bit immediate data.
	BST	B	$C \rightarrow (\text{<bit-No.> of <EAd>})$ Transfers the carry flag value to a specified bit in a general register or memory operand.
	BIST	B	$\neg C \rightarrow (\text{<bit-No.> of <EAd>})$ Transfers the inverse of the carry flag value to a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data.

Note: \* Size refers to the operand size.

B: Byte

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
Branch instructions	Bcc	—	Branches to a specified address if a specified condition is true. The branching conditions are listed below.
			<b>Mnemonic      Description      Condition</b>
			BRA(BT)      Always (true)      Always
			BRN(BF)      Never (false)      Never
			BHI      High $C \vee Z = 0$
			BLS      Low or same $C \vee Z = 1$
			BCC(BHS)      Carry clear (high or same) $C = 0$
			BCS(BLO)      Carry set (low) $C = 1$
			BNE      Not equal $Z = 0$
			BEQ      Equal $Z = 1$
			BVC      Overflow clear $V = 0$
			BVS      Overflow set $V = 1$
			BPL      Plus $N = 0$
			BMI      Minus $N = 1$
			BGE      Greater or equal $N \oplus V = 0$
			BLT      Less than $N \oplus V = 1$
			BGT      Greater than $Z \vee (N \oplus V) = 0$
			BLE      Less or equal $Z \vee (N \oplus V) = 1$
	JMP	—	Branches unconditionally to a specified address.
	BSR	—	Branches to a subroutine at a specified address.
	JSR	—	Branches to a subroutine at a specified address.
	RTS	—	Returns from a subroutine

**Table 1-3 Instructions Classified by Function (cont)**

Type	Instruction	Size*	Function
System control instructions	TRAPA	—	Starts trap-instruction exception handling.
	RTE	—	Returns from an exception-handling routine.
	SLEEP	—	Causes a transition to a power-down state.
	LDC	B/W	(EAs) → CCR, (EAs) → EXR Moves the source operand contents or immediate data to CCR or EXR. Although CCR and EXR are 8-bit registers, word-size transfers are performed between them and memory. The upper 8 bits are valid.
	STC	B/W	CCR → (EAd), EXR → (EAd) Transfers CCR or EXR contents to a general register or memory. Although CCR and EXR are 8-bit registers, word-size transfers are performed between them and memory. The upper 8 bits are valid.
	ANDC	B	$CCR \wedge \#IMM \rightarrow CCR$ , $EXR \wedge \#IMM \rightarrow EXR$ Logically ANDs the CCR or EXR contents with immediate data.
	ORC	B	$CCR \vee \#IMM \rightarrow CCR$ , $EXR \vee \#IMM \rightarrow EXR$ Logically ORs the CCR or EXR contents with immediate data.
	XORC	B	$CCR \oplus \#IMM \rightarrow CCR$ , $EXR \oplus \#IMM \rightarrow EXR$ Logically exclusive-ORs the CCR or EXR contents with immediate data.
	NOP	—	$PC + 2 \rightarrow PC$ Only increments the program counter.

Note: \* Size refers to the operand size.

B: Byte

W: Word



**Table 1-3 Instructions Classified by Function (cont)**

<b>Type</b>	<b>Instruction</b>	<b>Size*</b>	<b>Function</b>
Block data transfer instruction	EEPMOV.B	—	if R4L $\neq$ 0 then Repeat @ER5+ $\rightarrow$ @ER6+ R4L $- 1 \rightarrow$ R4L Until R4L = 0 else next;
	EEPMOV.W	—	if R4 $\neq$ 0 then Repeat @ER5+ $\rightarrow$ @ER6+ R4 $- 1 \rightarrow$ R4 Until R4 = 0 else next;  Transfers a data block according to parameters set in general registers R4L or R4, R5, and R6.  R4L or R4: size of block (bytes) ER5: starting source address ER6: starting destination address  Execution of the next instruction begins as soon as the transfer is completed.

#### 1.6.4 Basic Instruction Formats

The H8S/2600 instructions consist of 2-byte (1-word) units. An instruction consists of an operation field (op field), a register field (r field), an effective address extension (EA field), and a condition field (cc).

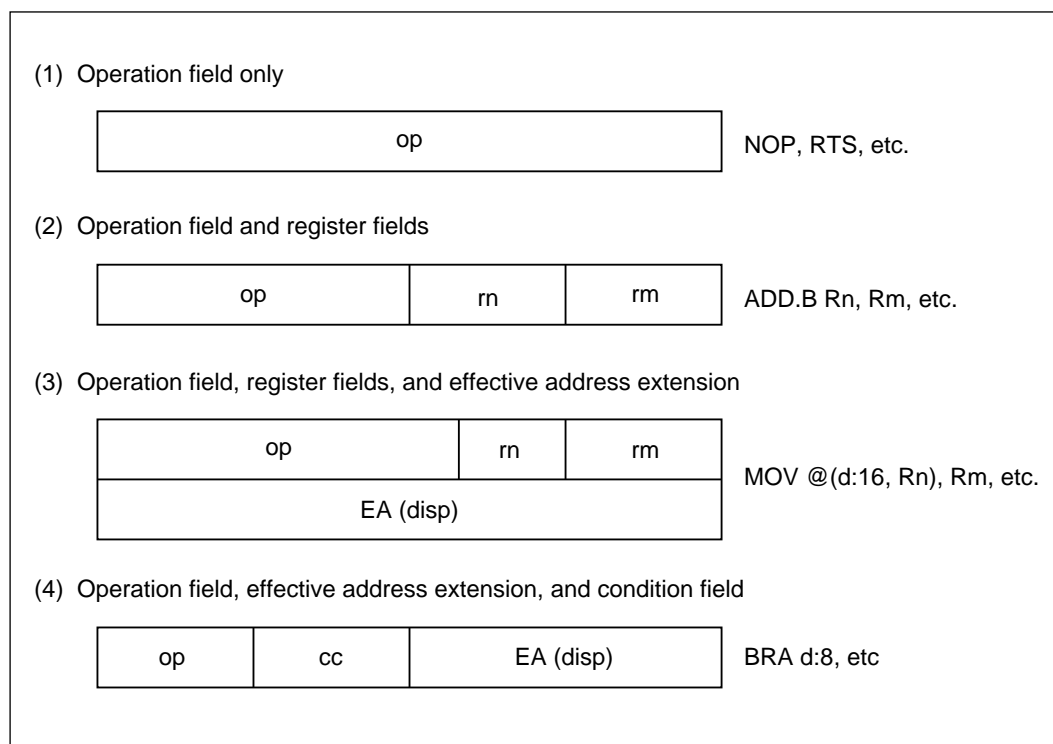
**Operation Field:** Indicates the function of the instruction, the addressing mode, and the operation to be carried out on the operand. The operation field always includes the first four bits of the instruction. Some instructions have two operation fields.

**Register Field:** Specifies a general register. Address registers are specified by 3 bits, data registers by 3 bits or 4 bits. Some instructions have two register fields. Some have no register field.

**Effective Address Extension:** Eight, 16, or 32 bits specifying immediate data, an absolute address, or a displacement.

**Condition Field:** Specifies the branching condition of Bcc instructions.

Figure 1-12 shows examples of instruction formats.



**Figure 1-12 Instruction Formats**

## 1.7 Addressing Modes and Effective Address Calculation

**Addressing Modes:** The CPUs support the eight addressing modes listed in table 1-4. Each instruction uses a subset of these addressing modes. Arithmetic and logic instructions can use the register direct and immediate modes. Data transfer instructions can use all addressing modes except program-counter relative and memory indirect. Bit manipulation instructions use register direct, register indirect, or absolute addressing mode to specify an operand, and register direct (BSET, BCLR, BNOT, and BTST instructions) or immediate (3-bit) addressing mode to specify a bit number in the operand.

**Table 1-4 Addressing Modes**

No.	Addressing Mode	Symbol
1	Register direct	Rn
2	Register indirect	@ERn
3	Register indirect with displacement	@(d:16,ERn)/@(d:32,ERn)
4	Register indirect with post-increment Register indirect with pre-decrement	@ERn+ @-ERn
5	Absolute address	@aa:8/@aa:16/@aa:24/@aa:32
6	Immediate	#xx:8/#xx:16/#xx:32
7	Program-counter relative	@(d:8,PC)/@(d:16,PC)
8	Memory indirect	@@aa:8

**1 Register Direct—Rn:** The register field of the instruction specifies an 8-, 16-, or 32-bit general register containing the operand. R0H to R7H and R0L to R7L can be specified as 8-bit registers. R0 to R7 and E0 to E7 can be specified as 16-bit registers. ER0 to ER7 can be specified as 32-bit registers.

**2 Register Indirect—@ERn:** The register field of the instruction code specifies an address register (ERn) which contains the address of the operand in memory. If the address is a program instruction address, the lower 24 bits are valid and the upper 8 bits are all assumed to be 0 (H'00).

**3 Register Indirect with Displacement—@(d:16, ERn) or @(d:32, ERn):** A 16-bit or 32-bit displacement contained in the instruction is added to an address register (ERn) specified by the register field of the instruction, and the sum gives the address of a memory operand. A 16-bit displacement is sign-extended when added.

#### 4 Register Indirect with Post-Increment or Pre-Decrement—@ERn+ or @-ERn:

- Register indirect with post-increment—@ERn+

The register field of the instruction code specifies an address register (ERn) which contains the address of a memory operand. After the operand is accessed, 1, 2, or 4 is added to the address register contents and the sum is stored in the address register. The value added is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the register value should be even.

- Register indirect with pre-decrement—@-ERn

The value 1, 2, or 4 is subtracted from an address register (ERn) specified by the register field in the instruction code, and the result becomes the address of a memory operand. The result is also stored in the address register. The value subtracted is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the register value should be even.

**5 Absolute Address—@aa:8, @aa:16, @aa:24, or @aa:32:** The instruction code contains the absolute address of a memory operand. The absolute address may be 8 bits long (@aa:8), 16 bits long (@aa:16), 24 bits long (@aa:24), or 32 bits long (@aa:32).

To access data, the absolute address should be 8 bits (@aa:8), 16 bits (@aa:16), or 32 bits (@aa:32) long. For an 8-bit absolute address, the upper 24 bits are all assumed to be 1 (H'FFFF). For a 16-bit absolute address the upper 16 bits are a sign extension. A 32-bit absolute address can access the entire address space.

A 24-bit absolute address (@aa:24) indicates the address of a program instruction. The upper 8 bits are all assumed to be 0 (H'00).

Table 1-5 indicates the accessible absolute address ranges.

**Table 1-5 Absolute Address Access Ranges**

Absolute Address		Normal Mode	Advanced Mode
Data address	8 bits (@aa:8)	H'FF00 to H'FFFF	H'FFFFFF00 to H'FFFFFFF
	16 bits (@aa:16)	H'0000 to H'FFFF	H'00000000 to H'00007FFF, H'FFFF8000 to H'FFFFFFF
	32 bits (@aa:32)		H'00000000 to H'FFFFFFF
Program instruction address	24 bits (@aa:24)		H'00000000 to H'00FFFFFF

For further details on the accessible range, refer to the relevant microcontroller hardware manual.

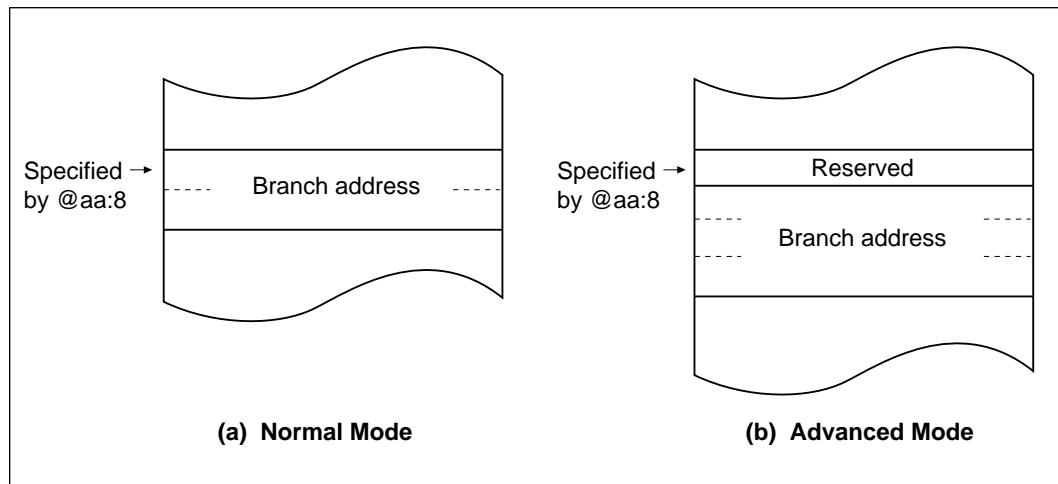
**6 Immediate—#xx:8, #xx:16, or #xx:32:** The instruction contains 8-bit (#xx:8), 16-bit (#xx:16), or 32-bit (#xx:32) immediate data as an operand.

The ADDS, SUBS, INC, and DEC instructions contain immediate data implicitly. Some bit manipulation instructions contain 3-bit immediate data in the instruction code, specifying a bit number. The TRAPA instruction contains 2-bit immediate data in its instruction code, specifying a vector address.

**7 Program-Counter Relative—@(d:8, PC) or @(d:16, PC):** This mode is used in the Bcc and BSR instructions. An 8-bit or 16-bit displacement contained in the instruction is sign-extended and added to the 24-bit PC contents to generate a branch address. Only the lower 24 bits of this branch address are valid; the upper 8 bits are all assumed to be 0 (H'00). The PC value to which the displacement is added is the address of the first byte of the next instruction, so the possible branching range is –126 to +128 bytes (–63 to +64 words) or –32766 to +32768 bytes (–16383 to +16384 words) from the branch instruction. The resulting value should be an even number.

**8 Memory Indirect—@@aa:8:** This mode can be used by the JMP and JSR instructions. The second byte of the instruction specifies a memory operand by an 8-bit absolute address. This memory operand contains a branch address. The upper bits of the absolute address are all assumed to be 0, so the address range is 0 to 255 (H'0000 to H'00FF in normal mode, H'00000000 to H'000000FF in advanced mode). In normal mode the memory operand is a word operand and the branch address is 16 bits long. In advanced mode the memory operand is a longword operand, the first byte of which is assumed to be all 0 (H'00).

Note that the first part of the address range is also the exception vector area. For further details refer to the relevant microcontroller hardware manual.



**Figure 1-13 Branch Address Specification in Memory Indirect Mode**

If an odd address is specified in word or longword memory access, or as a branch address, the least significant bit is regarded as 0, causing data to be accessed or an instruction code to be fetched at the address preceding the specified address. (For further information, see section 1.5.2, Memory Data Formats.)

**(2) Effective Address Calculation:** Table 1-6 indicates how effective addresses are calculated in each addressing mode. In normal mode the upper 8 bits of the effective address are ignored in order to generate a 16-bit address.

**Table 1-6 Effective Address Calculation**

No.	Addressing Mode and Instruction Format	Effective Address Calculation	Effective Address (EA)																																
1	Register direct (Rn) <table border="1"><tr><td>op</td><td>rm</td><td>rn</td></tr></table>	op	rm	rn		Operand is general register contents.																													
op	rm	rn																																	
2	Register indirect (@ERn) <table border="1"><tr><td>op</td><td>r</td><td></td></tr></table>	op	r		<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2">General register contents</td></tr></table>	31	0	General register contents		<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2"></td></tr></table>	31	0																							
op	r																																		
31	0																																		
General register contents																																			
31	0																																		
3	Register indirect with displacement @(d:16, ERn) or @(d:32, ERn) <table border="1"><tr><td>op</td><td>r</td><td></td><td>disp</td></tr></table>	op	r		disp	<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2">General register contents</td></tr></table> <table border="1"><tr><td>31</td><td>0</td></tr><tr><td>Sign extension</td><td>disp</td></tr></table>	31	0	General register contents		31	0	Sign extension	disp	<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2"></td></tr></table>	31	0																		
op	r		disp																																
31	0																																		
General register contents																																			
31	0																																		
Sign extension	disp																																		
31	0																																		
4	Register indirect with post-increment or pre-decrement • Register indirect with post-increment @ERn+ <table border="1"><tr><td>op</td><td>r</td><td></td></tr></table>  • Register indirect with pre-decrement @-ERn <table border="1"><tr><td>op</td><td>r</td><td></td></tr></table>	op	r		op	r		<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2">General register contents</td></tr></table> <table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2">General register contents</td></tr></table> <table border="1"><tr><td>1, 2, or 4</td></tr></table> <table border="1"><tr><td>1, 2, or 4</td></tr></table> <table border="1"><tr><td>Operand Size</td><td>Value added</td></tr><tr><td>Byte</td><td>1</td></tr><tr><td>Word</td><td>2</td></tr><tr><td>Longword</td><td>4</td></tr></table>	31	0	General register contents		31	0	General register contents		1, 2, or 4	1, 2, or 4	Operand Size	Value added	Byte	1	Word	2	Longword	4	<table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2"></td></tr></table> <table border="1"><tr><td>31</td><td>0</td></tr><tr><td colspan="2"></td></tr></table>	31	0			31	0		
op	r																																		
op	r																																		
31	0																																		
General register contents																																			
31	0																																		
General register contents																																			
1, 2, or 4																																			
1, 2, or 4																																			
Operand Size	Value added																																		
Byte	1																																		
Word	2																																		
Longword	4																																		
31	0																																		
31	0																																		

Table 1-6 Effective Address Calculation (cont)

No.	Addressing Mode and Instruction Format	Effective Address Calculation	Effective Address (EA)
5	Absolute address  @aa:8 <div><div>op</div><div>abs</div></div>		<div><div>31870</div><div>H'FFFFFF</div><div></div></div>
	@aa:16 <div><div>op</div><div>abs</div></div>		<div><div>3116150</div><div>Sign extension</div><div></div></div>
	@aa:24 <div><div>op</div><div>abs</div></div>		<div><div>3124230</div><div>H'00</div><div></div></div>
	@aa:32 <div><div>op</div><div>abs</div></div>		<div><div>310</div><div></div><div></div></div>
6	Immediate #xx:8/#xx:16/#xx:32 <div><div>op</div><div>IMM</div></div>		Operand is immediate data.



Table 1-6 Effective Address Calculation (cont)

No.	Addressing Mode and Instruction Format	Effective Address Calculation	Effective Address (EA)
7	<p>Program-counter relative</p> <p>@(d:8, PC)/@(d:16, PC)</p> <div><div>op</div><div>disp</div></div>	<div><div>230PC contents</div><div><div>230Sign extension</div><div>0disp</div></div></div> <div><div>230</div><div>0</div></div> <div><div>230</div><div>0</div></div> <div><div>230</div><div>0</div></div>	<div><div>3124230</div><div>H'00</div></div>
8	<p>Memory indirect @@aa:8</p> <ul style="list-style-type: none"><li>Normal mode</li></ul> <div><div>op</div><div>abs</div></div> <ul style="list-style-type: none"><li>Advanced mode</li></ul> <div><div>op</div><div>abs</div></div>	<div><div>31870H'000000abs</div><div>150Memory contents</div></div> <div><div>31870H'000000abs</div><div>31230ReservedMemory contents</div></div>	<div><div>3116150H'0000</div></div> <div><div>3124230H'00</div></div>



## Section 2 Instruction Descriptions

### 2.1 Tables and Symbols

This section explains how to read the tables in section 2.2, describing each instruction. Note that the descriptions of some instructions extend over more than one page.

[1] Mnemonic (Full Name)	[2] Type
[3] Operation	[6] Condition Code
[4] Assembly-Language Format	
[5] Operand Size	
[7] Description	
[8] Available Registers	
[9] Operand Format and Number of States Required for Execution	
[10] Notes	

**[1] Mnemonic (Full Name):** Gives the full and mnemonic names of the instruction.

**[2] Type:** Indicates the type of instruction.

**[3] Operation:** Describes the instruction in symbolic notation. (See section 2.1.2, Operation.)

**[4] Assembly-Language Format:** Indicates the assembly-language format of the instruction. (See section 2.1.1, Assembler Format.)

**[5] Operand Size:** Indicates the available operand sizes.

**[6] Condition Code:** Indicates the effect of instruction execution on the flag bits in the CCR. (See section 2.1.3, Condition Code.)

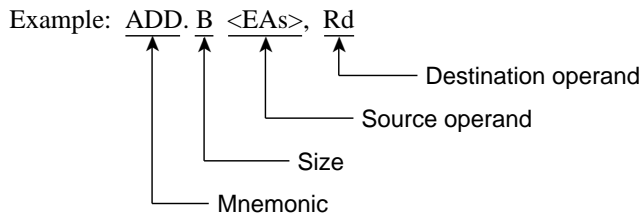
**[7] Description:** Describes the operation of the instruction in detail.

**[8] Available Registers:** Indicates which registers can be specified in the register field of the instruction.

**[9] Operand Format and Number of States Required for Execution:** Shows the addressing modes and instruction format together with the number of states required for execution.

**[10] Notes:** Gives notes concerning execution of the instruction.

### 2.1.1 Assembly-Language Format



The operand size is byte (B), word (W), or longword (L). Some instructions are restricted to a limited set of operand sizes.

The symbol <EA> indicates that two or more addressing modes can be used. The H8S/2600 CPU supports the eight addressing modes listed next. Effective address calculation is described in section 1.7, Addressing Modes and Effective Address Calculation.

Symbol	Addressing Mode
Rn	Register direct
@ERn	Register indirect
@(d:16, ERn)/@(d:32, ERn)	Register indirect with displacement (16-bit or 32-bit)
@ERn+/@-ERn	Register indirect with post-increment or pre-decrement
@aa:8/@aa:16/@aa:24/@aa:32	Absolute address (8-bit, 16-bit, 24-bit, or 32-bit)
#xx:8/#xx:16/#xx:32	Immediate (8-bit, 16-bit, or 32-bit)
@(d:8, PC)/@(d:16, PC)	Program-counter relative (8-bit or 16-bit)
@ @aa:8	Memory indirect

The suffixes :8, :16, :24, and :32 may be omitted. In particular, if the :8, :16, :24, or :32 designation is omitted in an absolute address or displacement, the assembler will optimize the length according to the value range. For details, refer to the H8S, H8/300 Series cross assembler user's manual.

### 2.1.2 Operation

The symbols used in the operation descriptions are defined as follows.

Rd	General register (destination)*
Rs	General register (source)*
Rn	General register*
ERn	General register (32-bit register)
MAC	Multiply-accumulate register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
EXR	Extended control register
CCR	Condition-code register
N	N (negative) flag in CCR
Z	Z (zero) flag in CCR
V	V (overflow) flag in CCR
C	C (carry) flag in CCR
PC	Program counter
SP	Stack pointer
#IMM	Immediate data
disp	Displacement
+	Add
−	Subtract
×	Multiply
÷	Divide
^	Logical AND
∨	Logical OR
⊕	Logical exclusive OR
→	Transfer from the operand on the left to the operand on the right, or transition from the state on the left to the state on the right
¬	Logical NOT (logical complement)
( ) < >	Contents of effective address of the operand

Note: \* General registers include 8-bit registers (R0H to R7H and R0L to R7L), 16-bit registers (R0 to R7 and E0 to E7), and 32-bit registers (ER0 to ER7).

### 2.1.3 Condition Code

The symbols used in the condition-code description are defined as follows.

Symbol	Meaning
↑	Changes according to the result of instruction execution
*	Undetermined (no guaranteed value)
0	Always cleared to 0
1	Always set to 1
—	Not affected by execution of the instruction
△	Varies depending on conditions; see the notes

For details on changes of the condition code, see section 2.7, Condition Code Modification.

### 2.1.4 Instruction Format

The symbols used in the instruction format descriptions are listed below.

Symbol	Meaning
IMM	Immediate data (2, 3, 8, 16, or 32 bits)
abs	Absolute address (8, 16, 24, or 32 bits)
disp	Displacement (8, 16, or 32 bits)
rs, rd, rn	Register field (4 bits). The symbols rs, rd, and rn correspond to operand symbols Rs, Rd, and Rn.
ers, erd, ern	Register field (3 bits). The symbols ers, erd, and ern correspond to operand symbols ERs, ERd, and ERn.

### 2.1.5 Register Specification

**Address Register Specification:** When a general register is used as an address register [ $@ERn$ ,  $@(d:16, ERn)$ ,  $@(d:32, ERn)$ ,  $@ERn+$ , or  $@-ERn$ ], the register is specified by a 3-bit register field (ers or erd).

**Data Register Specification:** A general register can be used as a 32-bit, 16-bit, or 8-bit data register.

When used as a 32-bit register, it is specified by a 3-bit register field (ers, erd, or ern).

When used as a 16-bit register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify an extended register (En) or cleared to 0 to specify a general register (Rn).

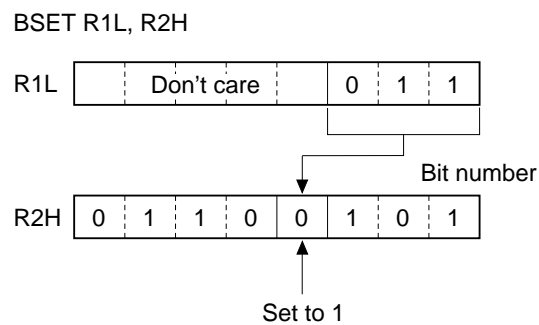
When used as an 8-bit register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify a low register (RnL) or cleared to 0 to specify a high register (RnH). This is shown next.

Address Register 32-Bit Register		16-Bit Register		8-Bit Register	
Register Field	General Register	Register Field	General Register	Register Field	General Register
000	ER0	0000	R0	0000	R0H
001	ER1	0001	R1	0001	R1H
.	.	.	.	.	.
.	.	.	.	.	.
111	ER7	0111	R7	0111	R7H
		1000	E0	1000	R0L
		1001	E1	1001	R1L
		.	.	.	.
		.	.	.	.
		1111	E7	1111	R7L

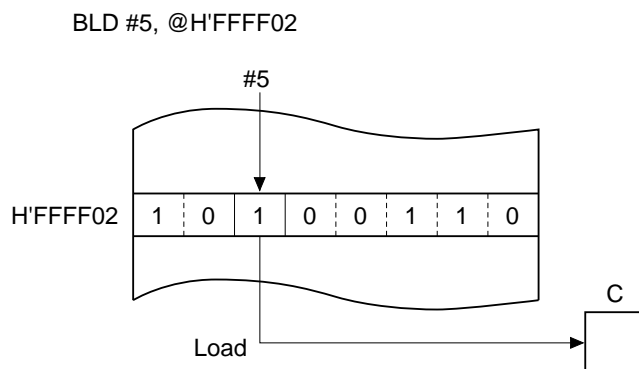
### 2.1.6 Bit Data Access in Bit Manipulation Instructions

Bit data is accessed as the n-th bit ( $n = 0, 1, 2, 3, \dots, 7$ ) of a byte operand in a general register or memory. The bit number is given by 3-bit immediate data, or by the lower 3 bits of a general register value.

Example 1: To set bit 3 in R2H to 1



Example 2: To load bit 5 at address H'FFFF02 into the bit accumulator



The operand size and addressing mode are as indicated for register or memory operand data.



## **2.2 Instruction Descriptions**

The instructions are described starting in section 2.2.1.

### 2.2.1 (1) ADD (B)

#### ADD (ADD Binary)

#### Add Binary

##### Operation

Rd + (EAs) → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↓	↓	↓	↓

##### Assembly-Language Format

ADD . B <EAs>, Rd

H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.

##### Operand Size

Byte

##### Description

This instruction adds the source operand to the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

##### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ADD.B	#xx:8, Rd	8	rd	IMM		1
Register direct	ADD.B	Rs, Rd	0	8	rs	rd	1

##### Notes

### 2.2.1 (2) ADD (W)

#### ADD (ADD Binary)

#### Add Binary

##### Operation

$Rd + (EAs) \rightarrow Rd$

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

##### Assembly-Language Format

ADD.W <EAs>, Rd

- H: Set to 1 if there is a carry at bit 11; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a carry at bit 15; otherwise cleared to 0.

##### Operand Size

Word

##### Description

This instruction adds the source operand to the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

##### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte		2nd byte		3rd byte			4th byte
Immediate	ADD.W	#xx:16, Rd	7	9	1	rd	IMM			2
Register direct	ADD.W	Rs, Rd	0	9	rs	rd				1

##### Notes

### 2.2.1 (3) ADD (L)

#### ADD (ADD Binary)

#### Add Binary

##### Operation

ERd + (EAs) → ERd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

##### Assembly-Language Format

ADD . L <EAs>, ERd

- H: Set to 1 if there is a carry at bit 27; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a carry at bit 31; otherwise cleared to 0.

##### Operand Size

Longword

##### Description

This instruction adds the source operand to the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

##### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte		
Immediate	ADD.L	#xx:32, ERd	7	A	1	0:erd	IMM			3
Register direct	ADD.L	ERs, ERd	0	A	1	ers:0:erd				1

##### Notes

### 2.2.2 ADDS

#### ADDS (ADD with Sign extension)

#### Add Binary Address Data

##### Operation

Rd + 1 → ERd

Rd + 2 → ERd

Rd + 4 → ERd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

##### Assembly-Language Format

ADDS #1, ERd

ADDS #2, ERd

ADDS #4, ERd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

##### Operand Size

Longword

#### Description

This instruction adds the immediate value 1, 2, or 4 to the contents of a 32-bit register ERd (destination operand). Unlike the ADD instruction, it does not affect the condition code flags.

#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ADDS	#1, ERd	0 B	0 0 erd			1
Register direct	ADDS	#2, ERd	0 B	8 0 erd			1
Register direct	ADDS	#4, ERd	0 B	9 0 erd			1

#### Notes

### 2.2.3 ADDX

**ADDX (ADD with eXtend carry)**

**Add with Carry**

---

#### Operation

$Rd + (EAs) + C \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↓	↓	↓	↓

---

#### Assembly-Language Format

ADDX <EAs>, Rd

H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.

---

#### Operand Size

Byte

---

#### Description

This instruction adds the source operand and carry flag to the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

---

#### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

---

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ADDX	#xx:8, Rd	9    rd	IMM			1
Register direct	ADDX	Rs, Rd	0    E	rs    rd			1

---

#### Notes

---

### 2.2.4 (1) AND (B)

#### AND (AND logical)

#### Logical AND

---

**Operation**

$Rd \wedge (EAs) \rightarrow Rd$

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	↓	↓	0	—

---

**Assembly-Language Format**

AND.B <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

---

**Operand Size**

Byte

---

**Description**

This instruction ANDs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

---

**Available Registers**

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

---

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	AND.B	#xx:8, Rd	E    rd	IMM			1
Register direct	AND.B	Rs, Rd	1    6	rs    rd			1

---

**Notes**

---

## 2.2.4 (2) AND (W)

### AND (AND logical)

### Logical AND

#### Operation

$Rd \wedge (EAs) \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

AND.W <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

### Description

This instruction ANDs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte		
Immediate	AND.W	#xx:16, Rd	7	9	6	rd	IMM	2
Register direct	AND.W	Rs, Rd	6	6	rs	rd		1

### Notes



### 2.2.4 (3) AND (L)

#### AND (AND logical)

#### Logical AND

##### Operation

$ERd \wedge (EAs) \rightarrow ERd$

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

##### Assembly-Language Format

AND.L <EAs>, ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Longword

#### Description

This instruction ANDs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	AND.L	#xx:32, ERd	7 : A	6 : 0:erd	IMM				3
Register direct	AND.L	ERs, ERd	0 : 1	F : 0	6 : 6	0:ers:0:erd			2

#### Notes

## 2.2.5 (1) ANDC

### ANDC (AND Control register)

### Logical AND with CCR

#### Operation

$CCR \wedge \#IMM \rightarrow CCR$

#### Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

#### Assembly-Language Format

ANDC #xx:8, CCR

I: Stores the corresponding bit of the result.

UI: Stores the corresponding bit of the result.

H: Stores the corresponding bit of the result.

U: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

#### Operand Size

Byte

#### Description

This instruction ANDs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ANDC	#xx:8, CCR	0 : 6	IMM			1

#### Notes

### 2.2.5 (2) ANDC

ANDC (AND Control register)

Logical AND with EXR

---

#### Operation

$\text{EXR} \wedge \# \text{IMM} \rightarrow \text{EXR}$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

---

#### Assembly-Language Format

ANDC #xx:8, EXR

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

#### Operand Size

Byte

---

#### Description

This instruction ANDs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

---

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ANDC	#xx:8, EXR	0 : 1	4 : 1	0 : 6	IMM	2

---

#### Notes

---

2.2.6 BAND

BAND (Bit AND)

Bit Logical AND

Operation

$C \wedge (\text{<bit No.> of <EAd>}) \rightarrow C$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BAND #xx:3, <EAd>

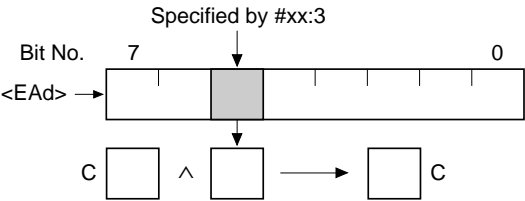
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

Operand Size

Byte

Description

This instruction ANDs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States								
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte	
Register direct	BAND	#xx:3, Rd	7	6	0:IMM: rd														1
Register indirect	BAND	#xx:3, @ERd	7	C	0:erd: 0		7	6	0:IMM: 0										3
Absolute address	BAND	#xx:3, @aa:8	7	E	abs		7	6	0:IMM: 0										3
Absolute address	BAND	#xx:3, @aa:16	6	A	1	0	abs			7	6	0:IMM: 0						4	
Absolute address	BAND	#xx:3, @aa:32	6	A	3	0	abs								7	6	0:IMM: 0		5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

## 2.2.7 Bcc

### Bcc (Branch conditionally)

### Conditional Branch

#### Operation

If condition is true, then  
 $PC + \text{disp} \rightarrow PC$   
 else next;

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

#### Assembly-Language Format

Bcc disp  
 └───────────> Condition field

H: Previous value remains unchanged.  
 N: Previous value remains unchanged.  
 Z: Previous value remains unchanged.  
 V: Previous value remains unchanged.  
 C: Previous value remains unchanged.

#### Operand Size

#### Description

If the condition specified in the condition field (cc) is true, a displacement is added to the program counter (PC) and execution branches to the resulting address. If the condition is false, the next instruction is executed. The PC value used in the address calculation is the starting address of the instruction immediately following the Bcc instruction. The displacement is a signed 8-bit or 16-bit value. The branch destination address can be located in the range from -126 to +128 bytes or -32766 to +32768 bytes from the Bcc instruction.

Mnemonic	Meaning	cc	Condition	Signed/Unsigned*
BRA (BT)	Always (true)	0000	True	
BRN (BF)	Never (false)	0001	False	
BHI	High	0010	$C \vee Z = 0$	$X > Y$ (unsigned)
BLS	Low or Same	0011	$C \vee Z = 1$	$X \leq Y$ (unsigned)
BCC (BHS)	Carry Clear (High or Same)	0100	$C = 0$	$X \geq Y$ (unsigned)
BCS (BLO)	Carry Set (LOW)	0101	$C = 1$	$X < Y$ (unsigned)
BNE	Not Equal	0110	$Z = 0$	$X \neq Y$ (unsigned or signed)
BEQ	Equal	0111	$Z = 1$	$X = Y$ (unsigned or signed)
BVC	oVerflow Clear	1000	$V = 0$	
BVS	oVerflow Set	1001	$V = 1$	
BPL	PLus	1010	$N = 0$	
BMI	MInus	1011	$N = 1$	
BGE	Greater or Equal	1100	$N \oplus V = 0$	$X \geq Y$ (signed)
BLT	Less Than	1101	$N \oplus V = 1$	$X < Y$ (signed)
BGT	Greater Than	1110	$Z \vee (N \oplus V) = 0$	$X > Y$ (signed)
BLE	Less or Equal	1111	$Z \vee (N \oplus V) = 1$	$X \leq Y$ (signed)

Note: \* If the immediately preceding instruction is a CMP instruction, X is the general register contents (destination operand) and Y is the source operand.

## 2.2.7 Bcc

**Bcc (Branch conditionally)**

**Conditional Branch**

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Program-counter relative	BRA (BT)	d:8	4	0	disp		2
		d:16	5	8	0	0	3
Program-counter relative	BRN (BF)	d:8	4	1	disp		2
		d:16	5	8	1	0	3
Program-counter relative	BHI	d:8	4	2	disp		2
		d:16	5	8	2	0	3
Program-counter relative	BLS	d:8	4	3	disp		2
		d:16	5	8	3	0	3
Program-counter relative	Bcc (BHS)	d:8	4	4	disp		2
		d:16	5	8	4	0	3
Program-counter relative	BCS (BLO)	d:8	4	5	disp		2
		d:16	5	8	5	0	3
Program-counter relative	BNE	d:8	4	6	disp		2
		d:16	5	8	6	0	3
Program-counter relative	BEQ	d:8	4	7	disp		2
		d:16	5	8	7	0	3
Program-counter relative	BVC	d:8	4	8	disp		2
		d:16	5	8	8	0	3
Program-counter relative	BVS	d:8	4	9	disp		2
		d:16	5	8	9	0	3
Program-counter relative	BPL	d:8	4	A	disp		2
		d:16	5	8	A	0	3
Program-counter relative	BMI	d:8	4	B	disp		2
		d:16	5	8	B	0	3
Program-counter relative	BGE	d:8	4	C	disp		2
		d:16	5	8	C	0	3
Program-counter relative	BLT	d:8	4	D	disp		2
		d:16	5	8	D	0	3
Program-counter relative	BGT	d:8	4	E	disp		2
		d:16	5	8	E	0	3
Program-counter relative	BLE	d:8	4	F	disp		2
		d:16	5	8	F	0	3

### Notes

1. The branch destination address must be even.
2. In machine language BRA, BRN, BCC, and BCS are identical to BT, BF, BHS, and BLO, respectively.

2.2.8 BCLR

BCLR (Bit CLear) Bit Clear

Operation

0 → (<bit No.> of <EAd>)

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

BCLR #xx:3, <EAd>

BCLR Rn, <EAd>

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

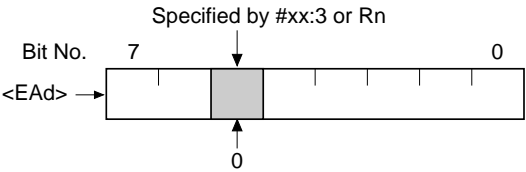
C: Previous value remains unchanged.

Operand Size

Byte

Description

This instruction clears a specified bit in the destination operand to 0. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The specified bit is not tested. The condition-code flags are not altered.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

Rn: R0L to R7L, R0H to R7H



### Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States						
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte	
Register direct	BCLR	#xx:3, Rd	7	2	0:IMM: rd												1
Register indirect	BCLR	#xx:3, @ERd	7	D	0:erd: 0		7	2	0:IMM: 0								4
Absolute address	BCLR	#xx:3, @aa:8	7	F	abs		7	2	0:IMM: 0								4
Absolute address	BCLR	#xx:3, @aa:16	6	A	1	8	abs			7	2	0:IMM: 0					5
Absolute address	BCLR	#xx:3, @aa:32	6	A	3	8	abs						7	2	0:IMM: 0		6
Register direct	BCLR	Rn, Rd	6	2	rn	rd											1
Register indirect	BCLR	Rn, @ERd	7	D	0:erd: 0		6	2	rn	0							4
Absolute address	BCLR	Rn, @aa:8	7	F	abs		6	2	rn	0							4
Absolute address	BCLR	Rn, @aa:16	6	A	1	8	abs			6	2	rn	0				5
Absolute address	BCLR	Rn, @aa:32	6	A	3	8	abs						6	2	rn	0	6

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

### Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.8 BCLR

BCLR (Bit Clear)

Bit Clear

2.2.9 BIAND

BIAND (Bit Invert AND)

Bit Logical AND

Operation

$C \wedge [\neg (\text{<bit No.> of <EAd>})] \rightarrow C$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BIAND #xx:3, <EAd>

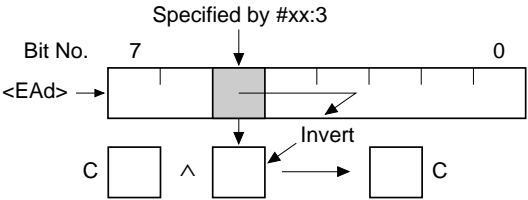
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

Operand Size

Byte

Description

This instruction ANDs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format										No. of States						
			1st byte		2nd byte		3rd byte		4th byte		5th byte			6th byte		7th byte		8th byte	
Register direct	BIAND	#xx:3, Rd	7	6	1:IMM:	rd												1	
Register indirect	BIAND	#xx:3, @ERd	7	C	0:erd:	0	7	6	1:IMM:	0								3	
Absolute address	BIAND	#xx:3, @aa:8	7	E		abs	7	6	1:IMM:	0								3	
Absolute address	BIAND	#xx:3, @aa:16	6	A	1	0	abs			7	6	1:IMM:	0					4	
Absolute address	BIAND	#xx:3, @aa:32	6	A	3	0	abs								7	6	1:IMM:	0	5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.10 BILD

BILD (Bit Invert Load) Bit Load

Operation

$\neg$  (<bit No.> of <EAd>)  $\rightarrow$  C

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BILD #xx:3, <EAd>

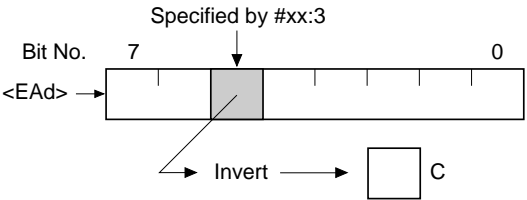
H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Loaded with the inverse of the specified bit.

Operand Size

Byte

Description

This instruction loads the inverse of a specified bit from the destination operand into the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States							
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte
Register direct	BILD	#xx:3, Rd	7	7	1:IMM: rd												1	
Register indirect	BILD	#xx:3, @ERd	7	C	0:erd 0	7	7	1:IMM: 0									3	
Absolute address	BILD	#xx:3, @aa:8	7	E	abs	7	7	1:IMM: 0									3	
Absolute address	BILD	#xx:3, @aa:16	6	A	1 0	abs			7 7	1:IMM: 0							4	
Absolute address	BILD	#xx:3, @aa:32	6	A	3 0	abs								7 7	1:IMM: 0			5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.11 BIOR

BIOR (Bit Invert inclusive OR)

Bit Logical OR

Operation

$C \vee [\neg (\text{<bit No.> of <EAd>})] \rightarrow C$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BIOR #xx:3, <EAd>

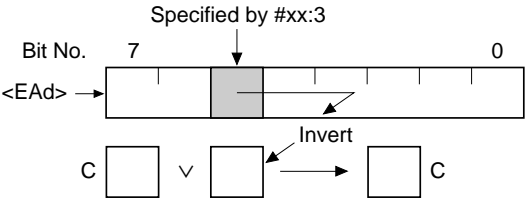
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

Operand Size

Byte

Description

This instruction ORs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States							
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte
Register direct	BIOR	#xx:3, Rd	7	4	1:IMM: rd												1	
Register indirect	BIOR	#xx:3, @ERd	7	C	0:erd	0	7	4	1:IMM: 0								3	
Absolute address	BIOR	#xx:3, @aa:8	7	E	abs		7	4	1:IMM: 0								3	
Absolute address	BIOR	#xx:3, @aa:16	6	A	1	0	abs			7	4	1:IMM: 0					4	
Absolute address	BIOR	#xx:3, @aa:32	6	A	3	0	abs								7	4	1:IMM: 0	5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.12 BIST

BIST (Bit Invert STore) Bit Store

Operation

$\neg C \rightarrow (\text{<bit No.> of <EAd>})$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

BIST #xx:3, <EAd>

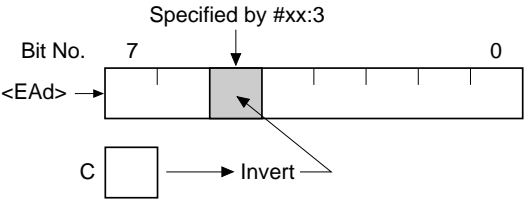
H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Previous value remains unchanged.

Operand Size

Byte

Description

This instruction stores the inverse of the carry flag in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data. Other bits in the destination operand remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7



**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States						
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte	
Register direct	BIST	#xx:3, Rd	6	7	1:IMM: rd												1
Register indirect	BIST	#xx:3, @ERd	7	D	0:erd: 0	6	7	1:IMM: 0									4
Absolute address	BIST	#xx:3, @aa:8	7	F	abs	6	7	1:IMM: 0									4
Absolute address	BIST	#xx:3, @aa:16	6	A	1:8	abs			6	7	1:IMM: 0						5
Absolute address	BIST	#xx:3, @aa:32	6	A	3:8	abs						6	7	1:IMM: 0			6

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.13 BIXOR

BIXOR (Bit Invert eXclusive OR)

Bit Exclusive Logical OR

Operation

$C \oplus [\neg (\text{<bit No.> of <EAd>})] \rightarrow C$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BIXOR #xx:3, <EAd>

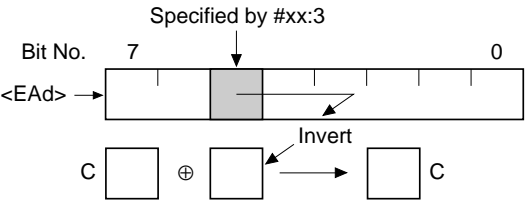
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

Operand Size

Byte

Description

This instruction exclusively ORs the inverse of a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7

## 2.2.13 BIXOR

BIXOR (Bit Invert eXclusive OR)

Bit Exclusive Logical OR

### Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States						
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte	
Register direct	BIXOR	#xx:3, Rd	7	5	1:IMM:rd												1
Register indirect	BIXOR	#xx:3, @ERd	7	C	0:erd	0	7	5	1:IMM:0								3
Absolute address	BIXOR	#xx:3, @aa:8	7	E	abs		7	5	1:IMM:0								3
Absolute address	BIXOR	#xx:3, @aa:16	6	A	1	0	abs			7	5	1:IMM:0					4
Absolute address	BIXOR	#xx:3, @aa:32	6	A	3	0	abs						7	5	1:IMM:0		5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

### Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.14 BLD

BLD (Bit Load) Bit Load

Operation

(<Bit No.> of <EAd>) → C

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BLD #xx:3, <EAd>

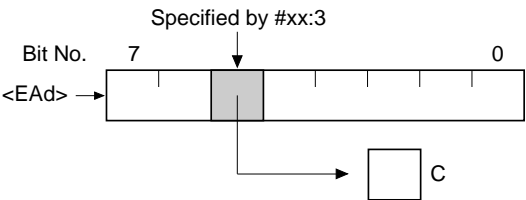
H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Loaded from the specified bit.

Operand Size

Byte

Description

This instruction loads a specified bit from the destination operand into the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States					
			1st byte		2nd byte		3rd byte		4th byte	5th byte		6th byte	7th byte	8th byte		
Register direct	BLD	#xx:3, Rd	7	7	0:IMM: rd										1	
Register indirect	BLD	#xx:3, @ERd	7	C	0:erd: 0		7	7	0:IMM: 0						3	
Absolute address	BLD	#xx:3, @aa:8	7	E	abs		7	7	0:IMM: 0						3	
Absolute address	BLD	#xx:3, @aa:16	6	A	1	0	abs			7	7	0:IMM: 0				4
Absolute address	BLD	#xx:3, @aa:32	6	A	3	0	abs					7	7	0:IMM: 0		5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.15 BNOT

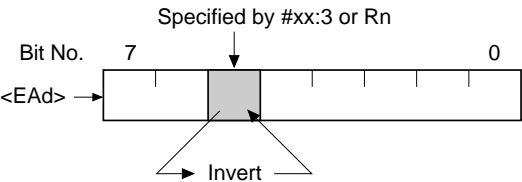
BNOT (Bit NOT)

Bit NOT

Operation	Condition Code																
$\neg$ (<bit No.> of <EAd>) $\rightarrow$ (bit No. of <EAd>)	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	—	—	—	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	—	—	—	—										
<hr/> <b>Assembly-Language Format</b>  BNOT #xx:3, <EAd> BNOT Rn, <EAd>	H: Previous value remains unchanged. N: Previous value remains unchanged. Z: Previous value remains unchanged. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<hr/> <b>Operand Size</b>  Byte																	

**Description**

This instruction inverts a specified bit in the destination operand. The bit number is specified by 3-bit immediate data or by the lower 3 bits of an 8-bit register Rn. The specified bit is not tested. The condition code remains unchanged.



**Available Registers**

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7  
Rn: R0L to R7L, R0H to R7H

## Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States			
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte				
Register direct	BNOT	#xx:3, Rd	7	1	0:IMM:rd						1			
Register indirect	BNOT	#xx:3, @ERd	7	D	0:erd:0	7	1	0:IMM:0			4			
Absolute address	BNOT	#xx:3, @aa:8	7	F	abs	7	1	0:IMM:0			4			
Absolute address	BNOT	#xx:3, @aa:16	6	A	1	8	abs		7	1	0:IMM:0	5		
Absolute address	BNOT	#xx:3, @aa:32	6	A	3	8	abs			7	1	0:IMM:0	6	
Register direct	BNOT	Rn, Rd	6	1	rn	rd						1		
Register indirect	BNOT	Rn, @ERd	7	D	0:erd:0	6	1	rn	0			4		
Absolute address	BNOT	Rn, @aa:8	7	F	abs	6	1	rn	0			4		
Absolute address	BNOT	Rn, @aa:16	6	A	1	8	abs		6	1	rn	0	5	
Absolute address	BNOT	Rn, @aa:32	6	A	3	8	abs			6	1	rn	0	6

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

## Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.16 BOR

BOR (Bit inclusive OR)

Bit Logical OR

Operation

$C \vee (\text{<bit No.> of <EAd>}) \rightarrow C$

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

Assembly-Language Format

BOR #xx:3, <EAd>

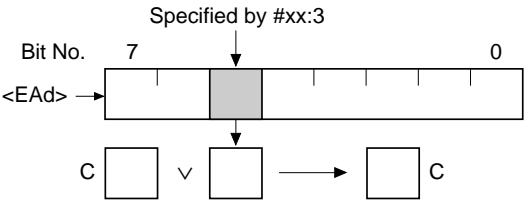
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Stores the result of the operation.

Operand Size

Byte

Description

This instruction ORs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7



**Operand Format and Number of States Required for Execution**

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States							
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte
Register direct	BOR	#xx:3, Rd	7	4	0:IMM: rd												1	
Register indirect	BOR	#xx:3, @ERd	7	C	0:erd: 0		7	4	0:IMM: 0								3	
Absolute address	BOR	#xx:3, @aa:8	7	E	abs		7	4	0:IMM: 0								3	
Absolute address	BOR	#xx:3, @aa:16	6	A	1	0	abs			7	4	0:IMM: 0						4
Absolute address	BOR	#xx:3, @aa:32	6	A	3	0	abs						7	4	0:IMM: 0		5	

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

**Notes**

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.17 BSET

BSET (Bit SET)

Bit Set

Operation

1 → (<bit No.> of <EAd>)

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

BSET #xx:3, <EAd>

BSET Rn, <EAd>

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

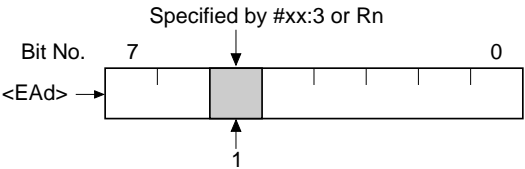
C: Previous value remains unchanged.

Operand Size

Byte

Description

This instruction sets a specified bit in the destination operand to 1. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The specified bit is not tested. The condition code flags are not altered.



Available Registers

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

Rn: R0L to R7L, R0H to R7H

## Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States							
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte
Register direct	BSET	#xx:3, Rd	7	0	0:IMM:rd												1	
Register indirect	BSET	#xx:3, @ERd	7	D	0:erd:0	7	0	0:IMM:0									4	
Absolute address	BSET	#xx:3, @aa:8	7	F	abs	7	0	0:IMM:0									4	
Absolute address	BSET	#xx:3, @aa:16	6	A	1	8	abs			7	0	0:IMM:0					5	
Absolute address	BSET	#xx:3, @aa:32	6	A	3	8	abs						7	0	0:IMM:0			6
Register direct	BSET	Rn, Rd	6	0	rn:rd												1	
Register indirect	BSET	Rn, @ERd	7	D	0:erd:0	6	0	rn:0									4	
Absolute address	BSET	Rn, @aa:8	7	F	abs	6	0	rn:0									4	
Absolute address	BSET	Rn, @aa:16	6	A	1	8	abs			6	0	rn:0					5	
Absolute address	BSET	Rn, @aa:32	6	A	3	8	abs						6	0	rn:0			6

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

## Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

## 2.2.18 BSR

### BSR (Branch to SubRoutine)

### Branch to Subroutine

#### Operation

PC → @-SP  
PC + disp → PC

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

#### Assembly-Language Format

BSR disp

H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Previous value remains unchanged.

#### Operand Size

—

### Description

This instruction branches to a subroutine at a specified address. It pushes the program counter (PC) value onto the stack as a restart address, then adds a specified displacement to the PC value and branches to the resulting address. The PC value pushed onto the stack is the address of the instruction following the BSR instruction. The displacement is a signed 8-bit or 16-bit value, so the possible branching range is –126 to +128 bytes or –32766 to +32768 bytes from the address of the BSR instruction.

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte	Normal	Advanced
Program-counter relative	BSR	d:8	5 : 5	disp			3	4
		d:16	5 : C	0 : 0	disp		4	5

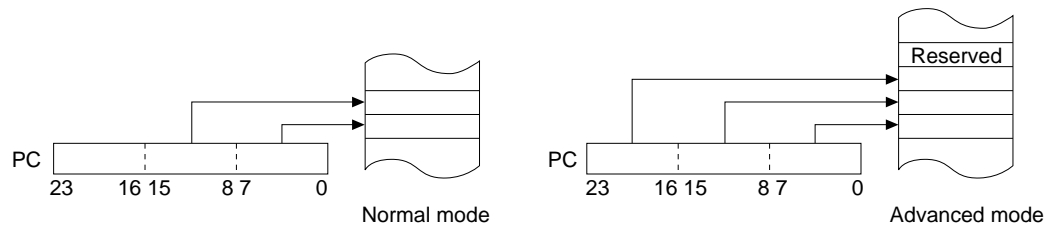
### 2.2.18 BSR

#### BSR (Branch to SubRoutine)

#### Branch to Subroutine

##### Notes

The stack structure differs between normal mode and advanced mode. In normal mode only the lower 16 bits of the program counter are pushed onto the stack.



2.2.19 BST

BST (Bit STore)

Bit Store

Operation

C → (<bit No.> of <EAd>)

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

BST #xx:3, <EAd>

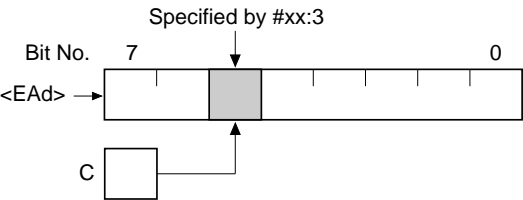
- H: Previous value remains unchanged.
- N: Previous value remains unchanged.
- Z: Previous value remains unchanged.
- V: Previous value remains unchanged.
- C: Previous value remains unchanged.

Operand Size

Byte

Description

This instruction stores the carry flag in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data.



Available Registers

Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format										No. of States						
			1st byte		2nd byte		3rd byte		4th byte		5th byte			6th byte		7th byte		8th byte	
Register direct	BST	#xx:3, Rd	6	7	0:IMM:	rd												1	
Register indirect	BST	#xx:3, @ERd	7	D	0:erd:	0	6	7	0:IMM:	0								4	
Absolute address	BST	#xx:3, @aa:8	7	F		abs	6	7	0:IMM:	0								4	
Absolute address	BST	#xx:3, @aa:16	6	A	1	8	abs				6	7	0:IMM:	0				5	
Absolute address	BST	#xx:3, @aa:32	6	A	3	8	abs								6	7	0:IMM:	0	6

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

### Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.20 BTST

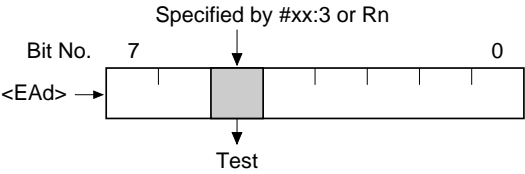
BTST (Bit TeST)

Bit Test

Operation	Condition Code																
$\neg$ (<Bit No.> of <EAd>) $\rightarrow$ Z	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td><math>\updownarrow</math></td><td>—</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	—	$\updownarrow$	—	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	—	$\updownarrow$	—	—										
<hr/> <b>Assembly-Language Format</b>  BTST #xx:3, <EAd> BTST Rn, <EAd>  <hr/>	H: Previous value remains unchanged. N: Previous value remains unchanged. Z: Set to 1 if the specified bit is zero; otherwise cleared to 0. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<b>Operand Size</b>  Byte																	

Description

This instruction tests a specified bit in the destination operand and sets or clears the zero flag according to the result. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit register Rn. The destination operand contents remain unchanged.



Available Registers

- Rd: R0L to R7L, R0H to R7H  
ERd: ER0 to ER7  
Rn: R0L to R7L, R0H to R7H



## Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States								
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte	
Register direct	BTST	#xx:3, Rd	7	3	0:IMM: rd														1
Register indirect	BTST	#xx:3, @ERd	7	C	0:erd: 0		7	3	0:IMM: 0										3
Absolute address	BTST	#xx:3, @aa:8	7	E	abs		7	3	0:IMM: 0										3
Absolute address	BTST	#xx:3, @aa:16	6	A	1	0	abs			7	3	0:IMM: 0						4	
Absolute address	BTST	#xx:3, @aa:32	6	A	3	0	abs						7	3	0:IMM: 0				5
Register direct	BTST	Rn, Rd	6	3	rn rd														1
Register indirect	BTST	Rn, @ERd	7	C	0:erd: 0		6	3	rn 0										3
Absolute address	BTST	Rn, @aa:8	7	E	abs		6	3	rn 0										3
Absolute address	BTST	Rn, @aa:16	6	A	1	0	abs			6	3	rn 0						4	
Absolute address	BTST	Rn, @aa:32	6	A	3	0	abs						6	3	rn 0				5

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

## Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

2.2.21 BXOR

**BXOR (Bit eXclusive OR)**

**Bit Exclusive Logical OR**

**Operation**

$C \oplus (\text{bit No.} > \text{ of } \text{<EAd>}) \rightarrow C$

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

**Assembly-Language Format**

BXOR #xx:3, <EAd>

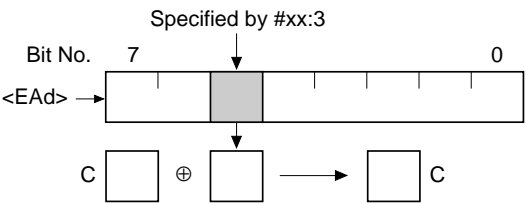
- H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Stores the result of the operation.

**Operand Size**

Byte

**Description**

This instruction exclusively ORs a specified bit in the destination operand with the carry flag and stores the result in the carry flag. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



**Available Registers**

Rd: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

## Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format										No. of States						
			1st byte		2nd byte		3rd byte		4th byte		5th byte			6th byte		7th byte		8th byte	
Register direct	BXOR	#xx:3, Rd	7	5	0:IMM: rd														1
Register indirect	BXOR	#xx:3, @ERd	7	C	0:erd: 0		7	5	0:IMM: 0										3
Absolute address	BXOR	#xx:3, @aa:8	7	E	abs		7	5	0:IMM: 0										3
Absolute address	BXOR	#xx:3, @aa:16	6	A	1	0	abs			7	5	0:IMM: 0						4	
Absolute address	BXOR	#xx:3, @aa:32	6	A	3	0	abs							7	5	0:IMM: 0		5	

Note: \* The addressing mode is the addressing mode of the destination operand <EAd>.

## Notes

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

## 2.2.22 CLRMAC

**CLRMAC (CLear MAC register)**

**Initialize Multiply-Accumulate Register**

### Operation

0 → MACH, MACL

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

CLRMAC

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### Operand Size

—

### Description

This instruction simultaneously clears registers MACH and MACL.

It is supported only by the H8S/2600 CPU

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	CLRMAC	—	0	1	A	0	2

### Notes

Execution of this instruction also clears the overflow flag in the multiplier to 0.

### 2.2.23 (1) CMP (B)

#### CMP (CoMPare)

#### Compare

##### Operation

Rd – (EAs), set/clear CCR

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

##### Assembly-Language Format

CMP.B <EAs>, Rd

- H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

##### Operand Size

Byte

##### Description

This instruction subtracts the source operand from the contents of an 8-bit register Rd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 8-bit register Rd remain unchanged.

##### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	CMP.B	#xx:8, Rd	A : rd	IMM			1
Register direct	CMP.B	Rs, Rd	1 : C	rs : rd			1

##### Notes

### 2.2.23 (2) CMP (W)

#### CMP (CoMPare)

#### Compare

##### Operation

Rd – (EAs), set/clear CCR

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

##### Assembly-Language Format

CMP.W <EAs>, Rd

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

##### Operand Size

Word

##### Description

This instruction subtracts the source operand from the contents of a 16-bit register Rd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 16-bit register Rd remain unchanged.

##### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	CMP.W	#xx:16, Rd	7	9	2	rd	IMM		2
Register direct	CMP.W	Rs, Rd	1	D	rs	rd			1

##### Notes

### 2.2.23 (3) CMP (L)

#### CMP (CoMPare)

#### Compare

##### Operation

ERd – (EAs), set/clear CCR

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

##### Assembly-Language Format

CMP.L <EAs>, ERd

##### Operand Size

Longword

- H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

##### Description

This instruction subtracts the source operand from the contents of a 32-bit register ERd (destination operand) and sets or clears the condition code bits according to the result. The contents of the 32-bit register ERd remain unchanged.

##### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte		
Immediate	CMP.L	#xx:32, ERd	7	A	2	0:erd	IMM			3
Register direct	CMP.L	ERs, ERd	1	F	1:ers	0:erd				1

##### Notes

## 2.2.24 DAA

### DAA (Decimal Adjust Add)

### Decimal Adjust

#### Operation

Rd (decimal adjust) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	*	—	↕	↕	*	↕

#### Assembly-Language Format

DAA Rd

H: Undetermined (no guaranteed value).

N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.

Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.

V: Undetermined (no guaranteed value).

C: Set to 1 if there is a carry at bit 7; otherwise left unchanged.

#### Operand Size

Byte

#### Description

Given that the result of an addition operation performed by an ADD.B or ADDX instruction on 4-bit BCD data is contained in an 8-bit register Rd and the carry and half-carry flags, the DAA instruction adjusts the contents of the 8-bit register Rd (destination operand) by adding H'00, H'06, H'60, or H'66 according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (Hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	0	A to F	06	0
0	0 to 9	1	0 to 3	06	0
0	A to F	0	0 to 9	60	1
0	9 to F	0	A to F	66	1
0	A to F	1	0 to 3	66	1
1	1 to 2	0	0 to 9	60	1
1	1 to 2	0	A to F	66	1
1	1 to 3	1	0 to 3	66	1



## 2.2.24 DAA

**DAA (Decimal Adjust Add)**

**Decimal Adjust**

---

### Available Registers

Rd: R0L to R7L, R0H to R7H

---

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DAA	Rd	0 : F	0 : rd			1

---

### Notes

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.

### 2.2.25 DAS

#### DAS (Decimal Adjust Subtract)

#### Decimal Adjust

##### Operation

Rd (decimal adjust) → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	*	—	↕	↕	*	—

##### Assembly-Language Format

DAS Rd

H: Undetermined (no guaranteed value).

N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.

Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.

V: Undetermined (no guaranteed value).

C: Previous value remains unchanged.

##### Operand Size

Byte

#### Description

Given that the result of a subtraction operation performed by a SUB.B, SUBX.B, or NEG.B instruction on 4-bit BCD data is contained in an 8-bit register Rd and the carry and half-carry flags, the DAS instruction adjusts the contents of the 8-bit register Rd (destination operand) by adding H'00, H'FA, H'A0, or H'9A according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (Hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	1	6 to F	FA	0
1	7 to F	0	0 to 9	A0	1
1	6 to F	1	6 to F	9A	1

#### Available Registers

Rd: R0L to R7L, R0H to R7H

### 2.2.25 DAS

**DAS (Decimal Adjust Subtract)**

**Decimal Adjust**

---

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DAS	Rd	1 : F	0 : rd			1

---

**Notes**

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.

### 2.2.26 (1) DEC (B)

#### DEC (DECrement)

Decrement

#### Operation

$Rd - 1 \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

#### Assembly-Language Format

DEC.B Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Byte

#### Description

This instruction decrements an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DEC.B	Rd	1 : A	0 : rd			1

#### Notes

An overflow is caused by the operation  $H'80 - 1 \rightarrow H'7F$ .

### 2.2.26 (2) DEC (W)

#### DEC (DECrement)

Decrement

#### Operation

Rd – 1 → Rd

Rd – 2 → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	↑	—

#### Assembly-Language Format

DEC.W #1, Rd

DEC.W #2, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DEC.W	#1, Rd	1 B	5 rd			1
Register direct	DEC.W	#2, Rd	1 B	D rd			1

#### Notes

An overflow is caused by the operations H'8000 – 1 → H'7FFF, H'8000 – 2 → H'7FFE, and H'8001 – 2 → H'7FFF.

### 2.2.26 (3) DEC (L)

#### DEC (DECrement)

Decrement

##### Operation

ERd – 1 → ERd

ERd – 2 → ERd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

##### Assembly-Language Format

DEC.L #1, ERd

DEC.L #2, ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Longword

#### Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DEC.L	#1, ERd	1 B	7 0:erd			1
Register direct	DEC.L	#2, ERd	1 B	F 0:erd			1

#### Notes

An overflow is caused by the operations H'80000000 – 1 → H'7FFFFFFF, H'80000000 – 2 → H'7FFFFFFE, and H'80000001 – 2 → H'7FFFFFFF.

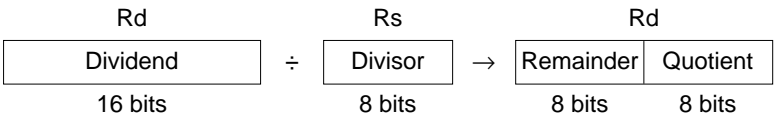
2.2.27 (1) DIVXS (B)

**DIVXS (DIVide eXtend as Signed)** **Divide Signed**

Operation	Condition Code																
$Rd \div Rs \rightarrow Rd$	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>—</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	—	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	—	—										
<hr/> <b>Assembly-Language Format</b> <code>DIVXS.B Rs, Rd</code>	H: Previous value remains unchanged. N: Set to 1 if the quotient is negative; otherwise cleared to 0. Z: Set to 1 if the divisor is zero; otherwise cleared to 0. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<hr/> <b>Operand Size</b> Byte																	

**Description**

This instruction divides the contents of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) and stores the result in the 16-bit register Rd. The division is signed. The operation performed is 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd. The sign of the remainder matches the sign of the dividend.



Valid results are not assured if division by zero is attempted or an overflow occurs.

**Available Registers**

Rd: R0 to R7, E0 to E7  
Rs: R0L to R7L, R0H to R7H

### 2.2.27 (1) DIVXS (B)

**DIVXS (DIVide eXtend as Signed)**

**Divide Signed**

---

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DIVXS.B	Rs, Rd	0 : 1	D : 0	5 : 1	rs : rd	13

---

#### Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.



### 2.2.27 (2) DIVXS (W)

**DIVXS (DIVide eXtend as Signed)**

**Divide Signed**

#### Operation

$ERd \div Rs \rightarrow ERd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	—	—

#### Assembly-Language Format

`DIVXS.W Rs, ERd`

H: Previous value remains unchanged.

N: Set to 1 if the quotient is negative; otherwise cleared to 0.

Z: Set to 1 if the divisor is zero; otherwise cleared to 0.

V: Previous value remains unchanged.

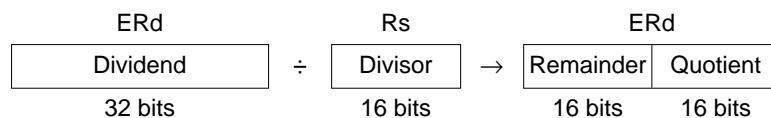
C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction divides the contents of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) and stores the result in the 32-bit register ERd. The division is signed. The operation performed is 32 bits  $\div$  16 bits  $\rightarrow$  16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 16 bits (Ed). The sign of the remainder matches the sign of the dividend.



Valid results are not assured if division by zero is attempted or an overflow occurs.

#### Available Registers

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

## 2.2.27 (2) DIVXS (W)

**DIVXS (DIVide eXtend as Signed)**

**Divide Signed**

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	DIVXS.W	Rs, ERd	0	1	D	0	5	3	rs	0:erd	21

### Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.

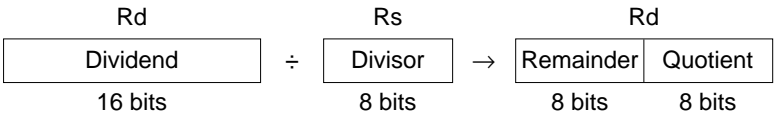
2.2.28 (1) DIVXU (B)

**DIVXU (DIVide eXtend as Unsigned)** **Divide**

Operation	Condition Code																
$Rd \div Rs \rightarrow Rd$	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>—</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	—	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	—	—										
<hr/> <b>Assembly-Language Format</b> <code>DIVXu.B Rs, Rd</code>	H: Previous value remains unchanged. N: Set to 1 if the divisor is negative; otherwise cleared to 0. Z: Set to 1 if the divisor is zero; otherwise cleared to 0. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<hr/> <b>Operand Size</b> Byte																	

**Description**

This instruction divides the contents of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) and stores the result in the 16-bit register Rd. The division is unsigned. The operation performed is 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd.



Valid results are not assured if division by zero is attempted or an overflow occurs.

**Available Registers**

Rd: R0 to R7, E0 to E7  
Rs: R0L to R7L, R0H to R7H

### 2.2.28 (1) DIVXU (B)

**DIVXU (DIVide eXtend as Unsigned)**

**Divide**

---

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DIVXU.B	Rs, Rd	5    1	rs    rd			12

---

**Notes**

## 2.2.28 (2) DIVXU (W)

### DIVXU (DIVide eXtend as Unsigned)

Divide

#### Operation

$ERd \div Rs \rightarrow ERd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	—	—

#### Assembly-Language Format

DIVXU.W Rs, ERd

H: Previous value remains unchanged.

N: Set to 1 if the divisor is negative; otherwise cleared to 0.

Z: Set to 1 if the divisor is zero; otherwise cleared to 0.

V: Previous value remains unchanged.

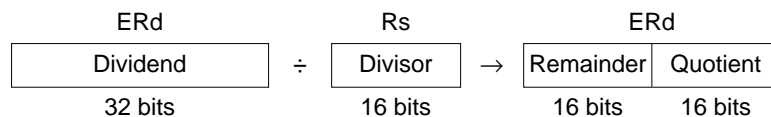
C: Previous value remains unchanged.

#### Operand Size

Word

### Description

This instruction divides the contents of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source register) and stores the result in the 32-bit register ERd. The division is unsigned. The operation performed is 32 bits  $\div$  16 bits  $\rightarrow$  16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 16 bits of (Ed).



Valid results are not assured if division by zero is attempted or an overflow occurs.

### Available Registers

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

### 2.2.28 (2) DIVXU (W)

**DIVXU (DIVide eXtend as Unsigned)**

**Divide**

---

#### **Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DIVXU.W	Rs, ERd	5 : 3	rs : 0 : erd			20

---

**Notes**

## 2.2.29 (1) EEPMOV (B)

### EEPMOV (MOVE data to EEPROM)

### Block Data Transfer

#### Operation

if R4L  $\neq$  0 then  
 repeat @ER5+  $\rightarrow$  @ER6+  
     R4L - 1  $\rightarrow$  R4L  
 until R4L = 0  
 else next;

#### Assembly-Language Format

EEPMOV . B

#### Operand Size

—

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

#### Description

This instruction performs a block data transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4L, and repeats these operations until R4L reaches zero. Execution then proceeds to the next instruction. The data transfer is performed a byte at a time, with R4L indicating the number of bytes to be transferred. The byte symbol in the assembly-language format designates the size of R4L (and limits the maximum number of bytes that can be transferred to 255). No interrupts are detected while the block transfer is in progress.

When the EEPMOV.B instruction ends, R4L contains 0 (zero), and ER5 and ER6 contain the last transfer address + 1.

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	EEPMOV.B		7 : B	5 : C	5 : 9	8 : F	4 + 2n*

Note: \* n is the initial value of R4L. Although n bytes of data are transferred,  $2(n + 1)$  data accesses are performed, requiring  $2(n + 1)$  states. (n = 0, 1, 2, ..., 255).

#### Notes

This instruction first reads the memory locations indicated by ER5 and ER6, then carries out the block data transfer.

## 2.2.29 (2) EEPMOV (W)

### EEPMOV (MOVE data to EEPROM)

### Block Data Transfer

---

#### Operation

if R4  $\neq$  0 then  
  repeat @ER5+  $\rightarrow$  @ER6+  
    R4 - 1  $\rightarrow$  R4  
  until R4 = 0  
else next;

---

#### Assembly-Language Format

EEPMOV.W

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

#### Operand Size

—

---

#### Description

This instruction performs a block data transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4, and repeats these operations until R4 reaches zero. Execution then proceeds to the next instruction. The data transfer is performed a byte at a time, with R4 indicating the number of bytes to be transferred. The word symbol in the assembly-language format designates the size of R4 (allowing a maximum 65535 bytes to be transferred). All interrupts are detected while the block transfer is in progress.

If no interrupt occurs while the EEPMOV.W instruction is executing, when the EEPMOV.W instruction ends, R4 contains 0 (zero), and ER5 and ER6 contain the last transfer address + 1.

If an interrupt occurs, interrupt exception handling begins after the current byte has been transferred. R4 indicates the number of bytes remaining to be transferred. ER5 and ER6 indicate the next transfer addresses. The program counter value pushed onto the stack in interrupt exception handling is the address of the next instruction after the EEPMOV.W instruction.

- See the note on EEPMOV.W and interrupts.



### 2.2.29 (2) EEPMOV (W)

#### EEPMOV (MOVE data to EEPROM)

#### Block Data Transfer

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	EEPMOV.W		7 : B	D : 4	5 : 9	8 : F	4 + 2n*

Note: \* n is the initial value of R4. Although n bytes of data are transferred, 2(n + 1) data accesses are performed, requiring 2(n + 1) states. (n = 0, 1, 2, ..., 65535).

##### Notes

This instruction first reads memory at the addresses indicated by ER5 and ER6, then carries out the block data transfer.

##### EEPMOV.W Instruction and Interrupt

If an interrupt request occurs while the EEPMOV.W instruction is being executed, interrupt exception handling is carried out after the current byte has been transferred. Register contents are then as follows:

ER5: address of the next byte to be transferred

ER6: destination address of the next byte

R4: number of bytes remaining to be transferred

The program counter value pushed on the stack in interrupt exception handling is the address of the next instruction after the EEPMOV.W instruction. Programs should be coded as follows to allow for interrupts during execution of the EEPMOV.W instruction.

##### Example:

```
L1: EEPMOV.W
    MOV.W    R4, R4
    BNE      L1
```

Interrupt requests other than NMI are not accepted if they are masked in the CPU.

During execution of the EEPMOV.B instruction no interrupts are accepted, including NMI.

2.2.30 (1) EXTS (W)

EXTS (EXTend as Signed)

Sign Extension

Operation

(<Bit 7> of Rd) → (<bits 15 to 8> of Rd)

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

Assembly-Language Format

EXTS.W Rd

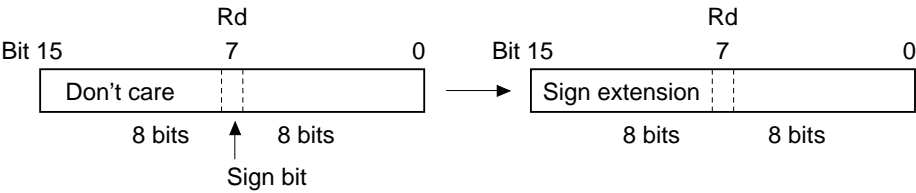
Operand Size

Word

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

Description

This instruction copies the sign of the lower 8 bits in a 16-bit register Rd in the upward direction (copies Rd bit 7 to bits 15 to 8) to extend the data to signed word data.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTS.W	Rd	1 : 7	D : rd			1

Notes

### 2.2.30 (2) EXTS (L)

#### EXTS (EXTend as Signed)

#### Sign Extension

##### Operation

(<Bit 15> of ERd) → (<bits 31 to 16> of ERd)

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

##### Assembly-Language Format

EXTS.L ERd

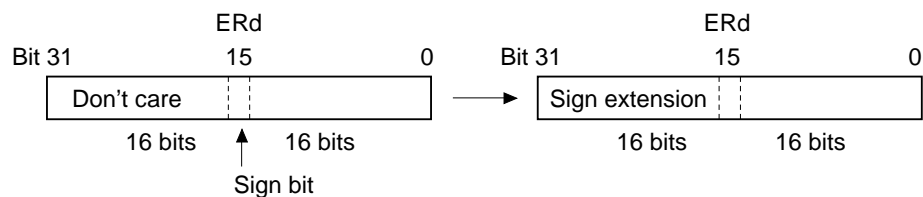
##### Operand Size

Longword

- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Previous value remains unchanged.

##### Description

This instruction copies the sign of the lower 16 bits in a 32-bit register ERd in the upward direction (copies ERd bit 15 to bits 31 to 16) to extend the data to signed longword data.



##### Available Registers

ERd: ER0 to ER7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTS.L	ERd	1 7	F 0 erd			1

##### Notes

2.2.31 (1) EXTU (W)

EXTU (EXTend as Unsigned)

Zero Extension

Operation

0 → (<bits 15 to 8> of Rd)

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	—

Assembly-Language Format

EXTU.W Rd

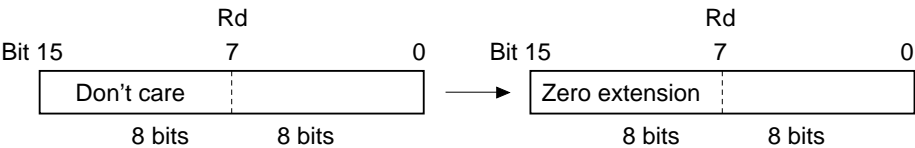
- H: Previous value remains unchanged.  
N: Always cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

Operand Size

Word

Description

This instruction extends the lower 8 bits in a 16-bit register Rd to word data by padding with zeros. That is, it clears the upper 8 bits of Rd (bits 15 to 8) to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTU.W	Rd	1    7	5    rd			1

Notes

## 2.2.31 (2) EXTU (L)

### EXTU (EXTend as Unsigned)

### Zero Extension

#### Operation

0 → (<bits 31 to 16> of ERd)

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↓	0	—

#### Assembly-Language Format

EXTU.L ERd

H: Previous value remains unchanged.

N: Always cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

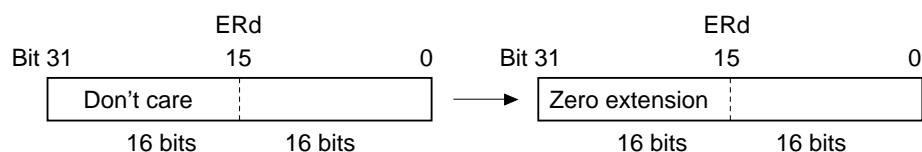
C: Previous value remains unchanged.

#### Operand Size

Longword

#### Description

This instruction extends the lower 16 bits (general register Rd) in a 32-bit register ERd to longword data by padding with zeros. That is, it clears the upper 16 bits of ERd (bits 31 to 16) to 0.



#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTU.L	ERd	1 : 7	7 : 0 : erd			1

#### Notes

### 2.2.32 (1) INC (B)

#### INC (INCrement)

#### Increment

##### Operation

$Rd + 1 \rightarrow Rd$

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

##### Assembly-Language Format

INC.B Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Byte

##### Description

This instruction increments an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

##### Available Registers

Rd: R0L to R7L, R0H to R7H

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	INC.B	Rd	0 : A	0 : rd			1

##### Notes

An overflow is caused by the operation  $H'7F + 1 \rightarrow H'80$ .

## 2.2.32 (2) INC (W)

### INC (INCRement)

Increment

#### Operation

$Rd + 1 \rightarrow Rd$

$Rd + 2 \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	↑	—

#### Assembly-Language Format

INC.W #1, Rd

INC.W #2, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction adds the immediate value 1 or 2 to the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	INC.W	#1, Rd	0 B	5 rd			1
Register direct	INC.W	#2, Rd	0 B	D rd			1

#### Notes

An overflow is caused by the operations  $H'7FFF + 1 \rightarrow H'8000$ ,  $H'7FFF + 2 \rightarrow H'8001$ , and  $H'7FFE + 2 \rightarrow H'8000$ .

## 2.2.32 (3) INC (L)

### INC (INCrement)

### Increment

#### Operation

ERd + 1 → ERd

ERd + 2 → ERd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

#### Assembly-Language Format

INC.L #1, ERd

INC.L #2, ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Longword

### Description

This instruction adds the immediate value 1 or 2 to the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

### Available Registers

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	INC.L	#1, ERd	0 B	7 0:erd			1
Register direct	INC.L	#2, ERd	0 B	F 0:erd			1

### Notes

An overflow is caused by the operations  $H'7FFFFFFF + 1 \rightarrow H'80000000$ ,  $H'7FFFFFFF + 2 \rightarrow H'80000001$ , and  $H'7FFFFFFE + 2 \rightarrow H'80000000$ .



### 2.2.33 JMP

#### JMP (JuMP)

#### Unconditional Branch

##### Operation

Effective address → PC

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

##### Assembly-Language Format

JMP <EA>

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

##### Operand Size

—

##### Description

This instruction branches unconditionally to a specified effective address.

##### Available Registers

ERn: ER0 to ER7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte	Normal	Advanced
Register indirect	JMP	@ERn	5 : 9	0 : ern: 0			2	
Absolute address	JMP	@aa:24	5 : A	abs			3	
Memory indirect	JMP	@@aa:8	5 : B	abs			4	5

##### Notes

The structure of the branch address and the number of states required for execution differ between normal mode and advanced mode.

## 2.2.34 JSR

### JSR (Jump to SubRoutine)

### Jump to Subroutine

#### Operation

PC → @-SP

Effective address → PC

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

#### Assembly-Language Format

JSR <EA>

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

#### Operand Size

—

#### Description

This instruction pushes the program counter onto the stack as a return address, then branches to a specified effective address. The program counter value pushed onto the stack is the address of the instruction following the JSR instruction.

#### Available Registers

ERn: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte	Normal	Advanced
Register indirect	JSR	@ERn	5	D	0:ern: 0		3	4
Absolute address	JSR	@aa:24	5	E	abs		4	5
Memory indirect	JSR	@ @aa:8	5	F	abs		4	6

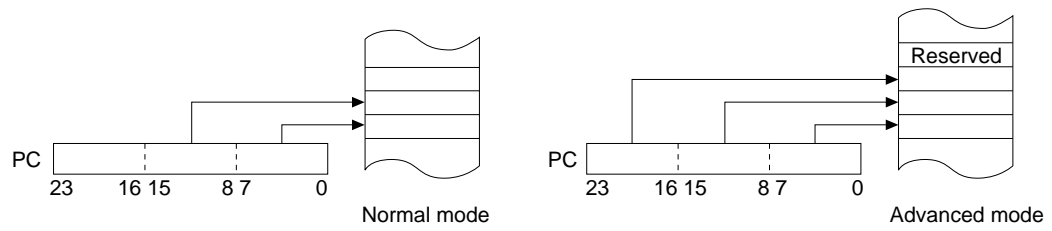
### 2.2.34 JSR

#### JSR (Jump to SubRoutine)

#### Jump to Subroutine

##### Notes

The stack structure differs between normal mode and advanced mode. In normal mode only the lower 16 bits of the program counter are pushed onto the stack.



## 2.2.35 (1) LDC (B)

**LDC (Load to Control register)**

**Load CCR**

### Operation

<EAs> → CCR

### Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

### Assembly-Language Format

LDC.B <EAs>, CCR

### Operand Size

Byte

- I: Loaded from the corresponding bit in the source operand.  
H: Loaded from the corresponding bit in the source operand.  
N: Loaded from the corresponding bit in the source operand.  
Z: Loaded from the corresponding bit in the source operand.  
V: Loaded from the corresponding bit in the source operand.  
C: Loaded from the corresponding bit in the source operand.

### Description

This instruction loads the source operand contents into the condition-code register (CCR).

No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

### Available Registers

Rs: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	LDC.B	#xx:8, CCR	0	7	IMM		1
Register direct	LDC.B	Rs, CCR	0	3	0	rs	1

### Notes

## 2.2.35 (2) LDC (B)

**LDC (Load to Control register)**

**Load EXR**

---

### Operation

<EAs> → EXR

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

---

### Assembly-Language Format

LDC.B <EAs>, EXR

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

### Operand Size

Byte

---

### Description

This instruction loads the source operand contents into the extended control register (EXR).

No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

---

### Available Registers

Rs: R0L to R7L, R0H to R7H

---

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte		2nd byte		3rd byte			4th byte
Immediate	LDC.B	#xx:8, EXR	0	1	4	1	0	7	IMM	2
Register direct	LDC.B	Rs, EXR	0	3	1	rs				1

---

### Notes

---

2.2.35 (3) LDC (W)

LDC (Load to Control register)

Load CCR

Operation

(EAs) → CCR

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

Assembly-Language Format

LDC .W <EAs>, CCR

- I: Loaded from the corresponding bit in the source operand.  
H: Loaded from the corresponding bit in the source operand.  
N: Loaded from the corresponding bit in the source operand.  
Z: Loaded from the corresponding bit in the source operand.  
V: Loaded from the corresponding bit in the source operand.  
C: Loaded from the corresponding bit in the source operand.

Operand Size

Word

Description

This instruction loads the source operand contents into the condition-code register (CCR). Although CCR is a byte register, the source operand is word size. The contents of the even address are loaded into CCR.

No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

Available Registers

ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format												No. of States							
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte			7th byte		8th byte		9th byte		10th byte
Register indirect	LDC.W	@ERs, CCR	0	1	4	0	6	9	0	ers	0											3
Register indirect with displacement	LDC.W	@(d:16, ERs), CCR	0	1	4	0	6	F	0	ers	0	disp										4
	LDC.W	@(d:32, ERs), CCR	0	1	4	0	7	8	0	ers	0	6	B	2	0	disp						6
Register indirect with post-increment	LDC.W	@ERs+, CCR	0	1	4	0	6	D	0	ers	0											4
Absolute address	LDC.W	@aa:16, CCR	0	1	4	0	6	B	0	0	abs										4	
	LDC.W	@aa:32, CCR	0	1	4	0	6	B	2	0	abs										5	

Notes

2.2.35 (4) LDC (W)

LDC (Load to Control register)

Load EXR

Operation

(EAs) → EXR

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

LDC . W <EAs>, EXR

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Operand Size

Word

Description

This instruction loads the source operand contents into the extended control register (EXR). Although EXR is a byte register, the source operand is word size. The contents of the even address are loaded into EXR.

No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

Available Registers

ERs: ER0 to ER7



Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format												No. of States							
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte			7th byte		8th byte		9th byte		10th byte
Register indirect	LDC.W	@ERs, EXR	0	1	4	1	6	9	0	ers	0											3
Register indirect with displacement	LDC.W	@(d:16, ERs), EXR	0	1	4	1	6	F	0	ers	0	disp										4
	LDC.W	@(d:32, ERs), EXR	0	1	4	1	7	8	0	ers	0	6	B	2	0	disp						6
Register indirect with post-increment	LDC.W	@ERs+, EXR	0	1	4	1	6	D	0	ers	0											4
Absolute address	LDC.W	@aa:16, EXR	0	1	4	1	6	B	0	0	abs										4	
	LDC.W	@aa:32, EXR	0	1	4	1	6	B	2	0	abs										5	

Notes

2.2.36 LDM

**LDM (Load to Multiple registers)** **Restore Data from Stack**

**Operation**

@SP+ → ERn (register list)

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

**Assembly-Language Format**

LDM.L @SP+, <register list>

**Operand Size**

Longword

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

**Description**

This instruction restores data saved on the stack to a specified list of registers. Registers are restored in descending order of register number.

Two, three, or four registers can be restored by one LDM instruction. The following ranges can be specified in the register list.

Two registers: ER0-ER1, ER2-ER3, ER4-ER5, or ER6-ER7

Three registers: ER0-ER2 or ER4-ER6

Four registers: ER0-ER3 or ER4-ER7

**Available Registers**

ERn: ER0 to ER7

## 2.2.36 LDM

LDM (Load to Multiple registers)

Restore Data from Stack

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
–	LDM.L	@SP+, (ERn–ERn+1)	0	1	1	0	6	D	7	0:ern+1	7
–	LDM.L	@SP+, (ERn–ERn+2)	0	1	2	0	6	D	7	0:ern+2	9
–	LDM.L	@SP+, (ERn–ERn+3)	0	1	3	0	6	D	7	0:ern+3	11

### Notes

## 2.2.37 LDMAC

**LDMAC (LoaD to MAC register)**

**Load MAC Register**

### Operation

ERs → MACH  
or  
ERs → MACL

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

LDMAC ERs, MAC register

### Operand Size

Longword

H: Previous value remains unchanged.  
N: Previous value remains unchanged.  
Z: Previous value remains unchanged.  
V: Previous value remains unchanged.  
C: Previous value remains unchanged.

### Description

This instruction moves the contents of a general register to a multiply-accumulate register (MACH or MACL). If the transfer is to MACH, only the lowest 10 bits of the general register are transferred.

Supported only by the H8S/2600 CPU.

### Available Registers

ERs: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	LDMAC	ERs, MACH	0 3	2 0:ers			2
Register direct	LDMAC	ERs, MACL	0 3	3 0:ers			2

### Notes

Execution of this instruction clears the overflow flag in the multiplier to 0.

## 2.2.38 MAC

### MAC (Multiply and ACcumulate)

### Multiply and Accumulate

Operation	Condition Code																
$(EAn) \times (EAm) + \text{MAC register} \rightarrow$ MAC register $ERn + 2 \rightarrow ERn$ $ERm + 2 \rightarrow ERm$	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—*</td><td>—*</td><td>—*</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	—*	—*	—*	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	—*	—*	—*	—										
<b>Assembly-Language Format</b>	H: Previous value remains unchanged. N: Previous value remains unchanged. Z: Previous value remains unchanged. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<b>Operand Size</b>																	
—																	

### Description

This instruction performs signed multiplication on two 16-bit operands at addresses given by the contents of general registers ERn and ERm, adds the 32-bit product to the contents of the MAC register, and stores the sum in the MAC register. After this operation, ERn and ERm are both incremented by 2.

The operation can be carried out in saturating or non-saturating mode, depending on the MACS bit in a system control register. (SYSCR)

See the relevant hardware manual for further information.

In non-saturating mode, MACH and MACL are concatenated to store a 42-bit result. The value of bit 41 is copied into the upper 22 bits of MACH as a sign extension.

In saturating mode, only MACL is valid, and the result is limited to the range from H'80000000 (minimum value) to H'7FFFFFFF (maximum value). If the result overflows in the negative direction, H'80000000 (the minimum value) is stored in MACL. If the result overflows in the positive direction, H'7FFFFFFF (the maximum value) is stored in MACL. MACH retains its previous contents.

This instruction is supported only by the H8S/2600 CPU.

## 2.2.38 MAC

MAC (Multiply and ACcumulate)

Multiply and Accumulate

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States		
			1st byte		2nd byte		3rd byte		4th byte				
Register indirect with post-increment	MAC	@ERn+, @ERm+	0	1	6	0	6	D	0	ern	0	erm	4

### Notes

Flags (N, Z, V) indicating the result of the MAC instruction can be set in the condition-code register (CCR) by the STMAC instruction.

### Further Explanation of Instructions Using Multiplier

#### 1. Modification of flags

The multiplier has N-MULT, Z-MULT, and V-MULT flags that indicate the results of MAC instructions. These flags are separated from the condition-code register (CCR). The values of these flags can be set in the N, Z, and V flags of the CCR only by the STMAC instruction.

N-MULT and Z-MULT are modified only by MAC instructions. V-MULT retains a value indicating whether an overflow has occurred in the past, until it is cleared by execution of the CLRMAC or LDMAC instruction.

The setting and clearing conditions for these flags are given below.

- N-MULT (negative flag)

Saturating mode	Set when bit 31 of register MACL is set to 1 by execution of a MAC instruction
	Cleared when bit 31 of register MACL is cleared to 0 by execution of a MAC instruction
Non-saturating mode	Set when bit 41 of register MACH is set to 1 by execution of a MAC instruction
	Cleared when bit 41 of register MACH is cleared to 0 by execution of a MAC instruction

## 2.2.38 MAC

### MAC (Multiply and ACcumulate)

### Multiply and Accumulate

- Z-MULT (zero flag)

Saturating mode	Set when register MACL is cleared to 0 by execution of a MAC instruction
	Cleared when register MACL is not cleared to 0 by execution of a MAC instruction
Non-saturating mode	Set when registers MACH and MACL are both cleared to 0 by execution of a MAC instruction
	Cleared when register MACH or MACL is not cleared to 0 by execution of a MAC instruction

- V-MULT (overflow flag)

Saturating mode	Set when the result of the MAC instruction overflows the range from H'80000000 (minimum) to H'7FFFFFFF (maximum)
	Cleared when a CLRMAC or LDMAC instruction is executed Note: Not cleared when the result of the MAC instruction is within the above range
Non-saturating mode	Set when the result of the MAC instruction overflows the range from H'2000000000 (minimum) to H'1FFFFFFFFF (maximum)
	Cleared when a CLRMAC or LDMAC instruction is executed Note: Not cleared when the result of the MAC instruction is within the above range

The N-MULT, Z-MULT, and V-MULT flags are not modified by switching between saturating and non-saturating modes, or by execution of a multiply instruction (MULXU or MULXS).

## 2. Example

CLRMAC

MAC @ER1+, @ER2+

MAC @ER1+, @ER2+ ← Overflow occurs

:

MAC @ER1+, @ER2+ ← Result = 0

NOP

STMACH, ER3 ← CCR (N = 0, Z = 1, V = 1)

CLRMAC

STMACH, ER3 ← CCR (N = 0, Z = 1, V = 0)

### 2.2.39 (1) MOV (B)

#### MOV (MOVE data)

Move

##### Operation

Rs → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

##### Assembly-Language Format

MOV.B Rs, Rd

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Byte

#### Description

This instruction transfers one byte of data from an 8-bit register Rs to an 8-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### Available Registers

Rs: R0L to R7L, R0H to R7H

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MOV.B	Rs, Rd	0 : C	rs : rd			1

#### Notes



## 2.2.39 (2) MOV (W)

### MOV (MOVE data)

Move

#### Operation

Rs → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

#### Assembly-Language Format

MOV.W Rs, Rd

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction transfers one word of data from a 16-bit register Rs to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

#### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MOV.W	Rs, Rd	0 : D	rs : rd			1

#### Notes

### 2.2.39 (3) MOV (L)

#### MOV (MOVE data)

Move

##### Operation

ERs → ERd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

##### Assembly-Language Format

MOV.L ERs, ERd

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Longword

##### Description

This instruction transfers one word of data from a 32-bit register ERs to a 32-bit register ERd, tests the transferred data, and sets condition-code flags according to the result.

##### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MOV.L	ERs, ERd	0 : F	1 : ers : 0 : erd			1

##### Notes

### 2.2.39 (4) MOV (B)

**MOV (MOVE data)**

**Move**

---

#### Operation

(EAs) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

#### Assembly-Language Format

MOV.B <EAs>, Rd

H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

---

#### Operand Size

Byte

---

#### Description

This instruction transfers the source operand contents to an 8-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

---

#### Available Registers

Rd: R0L to R7L, R0H to R7H

ERs: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States		
			1st byte		2nd byte		3rd byte		4th byte	5th byte		6th byte	7th byte
Immediate	MOV.B	#xx:8, Rd	F	rd	IMM								1
Register indirect	MOV.B	@ERs, Rd	6	8	0	ers	rd						2
Register indirect with displacement	MOV.B	@(d:16, ERs), Rd	6	E	0	ers	rd	disp					3
	MOV.B	@(d:32, ERs), Rd	7	8	0	ers	0	6	A	2	rd	disp	
Register indirect with post-increment	MOV.B	@ERs+, Rd	6	C	0	ers	rd						3
Absolute address	MOV.B	@aa:8, Rd	2	rd	abs								2
	MOV.B	@aa:16, Rd	6	A	0	rd	abs						3
	MOV.B	@aa:32, Rd	6	A	2	rd	abs						4

#### Notes

The MOV.B @ER7+, Rd instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3, Exception-Handling State, or to the relevant hardware manual.

For the @aa:8/@aa:16 access range, refer to the relevant microcontroller hardware manual.

MOV (MOVE data)

2.2.39 (4) MOV (B)

Move

### 2.2.39 (5) MOV (W)

**MOV (MOVE data)**

**Move**

---

**Operation**

(EAs) → Rd

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

**Assembly-Language Format**

MOV.W <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

---

**Operand Size**

Word

---

**Description**

This instruction transfers the source operand contents to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

---

**Available Registers**

Rd: R0 to R7, E0 to E7

ERs: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States									
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte		7th byte		8th byte		
Immediate	MOV.W	#xx:16, Rd	7	9	0	rd	IMM												2	
Register indirect	MOV.W	@ERs, Rd	6	9	0	ers	rd													2
Register indirect with displacement	MOV.W	@(d:16, ERs), Rd	6	F	0	ers	rd	disp												3
	MOV.W	@(d:32, ERs), Rd	7	8	0	ers	0	6	B	2	rd	disp								5
Register indirect with post-increment	MOV.W	@ERs+, Rd	6	D	0	ers	rd													3
Absolute address	MOV.W	@aa:16, Rd	6	B	0	rd	abs												3	
	MOV.W	@aa:32, Rd	6	B	2	rd	abs												4	

#### Notes

1. The source operand <EAs> must be located at an even address.
2. In machine language, MOV.W @ER7+, Rd is identical to POP.W Rd.

MOV (MOVE data)

2.2.39 (S) MOV (W)

Move

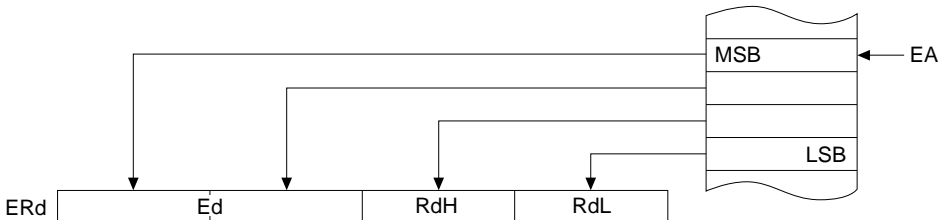
2.2.39 (6) MOV (L)

MOV (MOVE data) Move

Operation	Condition Code																
(EAs) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↑</td><td>↑</td><td>0</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↑	↑	0	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	↑	↑	0	—										
<hr/> <b>Assembly-Language Format</b> MOV.L <EAs>, ERd	H: Previous value remains unchanged. N: Set to 1 if the transferred data is negative; otherwise cleared to 0. Z: Set to 1 if the transferred data is zero; otherwise cleared to 0. V: Always cleared to 0. C: Previous value remains unchanged.																
<hr/> <b>Operand Size</b> Longword																	

Description

This instruction transfers the source operand contents to a specified 32-bit register (ERd), tests the transferred data, and sets condition-code flags according to the result. The first memory word located at the effective address is stored in extended register Ed. The next word is stored in general register Rd.



Available Registers

ERs: ER0 to ER7  
ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format										No. of States									
			1st byte		2nd byte		3rd byte		4th byte		5th byte			6th byte		7th byte		8th byte		9th byte		10th byte
Immediate	MOV.L	#xx:32, Rd	7	A	0	0:erd	IMM										3					
Register indirect	MOV.L	@ERs, ERd	0	1	0	0	6	9	0:ers:0:erd													4
Register indirect with displacement	MOV.L	@(d:16, ERs), ERd	0	1	0	0	6	F	0:ers:0:erd	disp												5
	MOV.L	@(d:32, ERs), ERd	0	1	0	0	7	8	0:ers:0	6	B	2	0:erd	disp								7
Register indirect with post-increment	MOV.L	@ERs+, ERd	0	1	0	0	6	D	0:ers:0:erd													5
Absolute address	MOV.L	@aa:16, ERd	0	1	0	0	6	B	0	0:erd	abs											5
	MOV.L	@aa:32, ERd	0	1	0	0	6	B	2	0:erd	abs										6	

#### Notes

1. The source operand <EAs> must be located at an even address.
2. In machine language, MOV.L @R7+, ERd is identical to POPL ERd.

MOV (MOVE data)

2.2.39 (6) MOV (L)

Move



### 2.2.39 (7) MOV (B)

**MOV (MOVE data)**

**Move**

---

**Operation**

Rs → (EAd)

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

**Assembly-Language Format**

MOV.B Rs, <EAd>

H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

---

**Operand Size**

Byte

---

**Description**

This instruction transfers the contents of an 8-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

---

**Available Registers**

Rs: R0L to R7L, R0H to R7H

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	
Register indirect	MOV.B	Rs, @ERd	6	8	1:erd	rs					2
Register indirect with displacement	MOV.B	Rs, @ (d:16, ERd)	6	E	1:erd	rs	disp				3
	MOV.B	Rs, @ (d:32, ERd)	7	8	0:erd	0	6	A	A	rs	5
Register indirect with pre-decrement	MOV.B	Rs, @-ERd	6	C	1:erd	rs					3
Absolute address	MOV.B	Rs, @aa:8	3	rs	abs						2
	MOV.B	Rs, @aa:16	6	A	8	rs	abs				3
	MOV.B	Rs, @aa:32	6	A	A	rs	abs				4

#### Notes

1. The MOV.B Rs, @-ER7 instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3, Exception Handling State, or to the relevant hardware manual.
2. Execution of MOV.B RnL, @-ERn or MOV.B RnH, @-ERn first decrements ERn by one, then transfers the designated part (RnL or RnH) of the resulting ERn value.

MOV (MOVE data)

2.2.39 (7) MOV (B)

Move

### 2.2.39 (8) MOV (W)

**MOV (MOVE data)**

**Move**

---

#### Operation

Rs → (EAd)

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

#### Assembly-Language Format

MOV.W Rs, <EAd>

H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

---

#### Operand Size

Word

---

#### Description

This instruction transfers the contents of a 16-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

---

#### Available Registers

Rs: R0 to R7, E0 to E7

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format										No. of States		
			1st byte		2nd byte		3rd byte		4th byte		5th byte	6th byte		7th byte	8th byte
Register indirect	MOV.W	Rs, @ERd	6	9	1	erd	rs								2
Register indirect with displacement	MOV.W	Rs, @ (d:16, ERd)	6	F	1	erd	rs	disp							3
	MOV.W	Rs, @ (d:32, ERd)	7	8	0	erd	0	6	B	A	rs	disp			5
Register indirect with pre-decrement	MOV.W	Rs, @-ERd	6	D	1	erd	rs								3
Absolute address	MOV.W	Rs, @aa:16	6	B	8	rs	abs								3
	MOV.W	Rs, @aa:32	6	B	A	rs	abs								4

#### Notes

1. The destination operand <EAd> must be located at an even address.
2. In machine language, MOV.W Rs, @-ER7 is identical to PUSH.W Rs.
3. When MOV.W Rn, @-ERn is executed, the transferred value comes from (value of ERn before execution) – 2.

MOV (MOVE data)

2.2.39 (8) MOV (W)

Move

## 2.2.39 (9) MOV (L)

**MOV (MOVE data)**

**Move**

### Operation

ERs → (EAd)

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

### Assembly-Language Format

MOV.L ERs, <EAd>

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

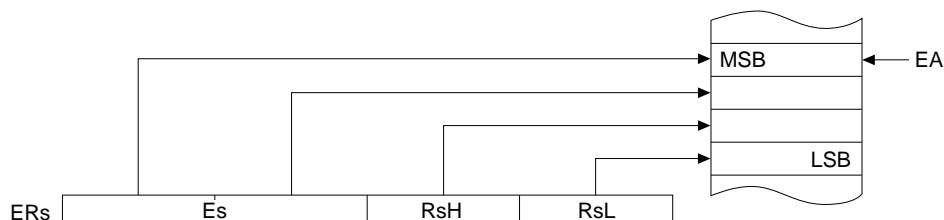
C: Previous value remains unchanged.

### Operand Size

Longword

### Description

This instruction transfers the contents of a 32-bit register ERs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result. The extended register (Es) contents are stored at the first word indicated by the effective address. The general register (Rs) contents are stored at the next word.



### Available Registers

ERs: ER0 to ER7

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format										No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte	
Register indirect	MOV.L	ERs, @ERd	0 1	0 0	6 9	1:erd;0:ers							4
Register indirect with displacement	MOV.L	ERs, @(d:16, ERd)	0 1	0 0	6 F	1:erd;0:ers	disp						5
	MOV.L	ERs, @(d:32, ERd)	0 1	0 0	7 8	1:erd;0:ers	6 B	A:0:ers	disp				7
Register indirect with pre-decrement	MOV.L	ERs, @-ERd	0 1	0 0	6 D	1:erd;0:ers							5
Absolute address	MOV.L	ERs, @aa:16	0 1	0 0	6 B	8:0:ers	abs						5
	MOV.L	ERs, @aa:32	0 1	0 0	6 B	A:0:ers	abs						6

#### Notes

1. The destination operand <EAd> must be located at an even address.
2. In machine language, MOV.L ERs, @-ER7 is identical to PUSH.L ERs.
3. When MOV.L ERn, @-ERn is executed, the transferred value is (value of ERn before execution) – 4.

MOV (MOVE data)

2.2.39 (9) MOV (L)

Move

## 2.2.40 MOVFPE

**MOVFPE (MOVE From Peripheral with E clock)**

**Move Data with E Clock**

### Operation

(EAs) → Rd  
Synchronized with E clock

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

### Assembly-Language Format

MOVFPE @aa:16, Rd

H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

### Operand Size

Byte

### Description

This instruction transfers memory contents specified by a 16-bit absolute address to a general register Rd in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers without an E clock output pin, or in single-chip mode.

### Available Registers

Rd: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Absolute address	MOVFPE	@aa:16, Rd	6 A	4 rd	abs		*

### Notes

1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
2. The number of states required for execution is variable. For details, refer to the relevant microcontroller hardware manual.

## 2.2.41 MOVTPE

### MOVTPE (MOVE To Peripheral with E clock)

### Move Data with E Clock

#### Operation

Rs → (EAd)  
Synchronized with E clock

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

MOVTPE Rs, @aa:16

H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative;  
otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero;  
otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

#### Operand Size

Byte

#### Description

This instruction transfers the contents of a general register Rs (source operand) to a destination location specified by a 16-bit absolute address in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers without an E clock output pin, or in single-chip mode.

#### Available Registers

Rs: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Absolute address	MOVTPE	Rs, @aa:16	6    A	C    rs	abs		*

#### Notes

1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
2. The number of states required for execution is variable. For details, refer to the relevant microcontroller hardware manual.



## 2.2.42 (1) MULXS (B)

### MULXS (MULTIPLY eXtend as Signed)

### Multiply Signed

#### Operation

$Rd \times Rs \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	—	—

#### Assembly-Language Format

MULXS.B Rs, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Previous value remains unchanged.

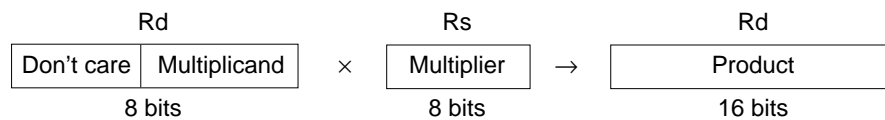
C: Previous value remains unchanged.

#### Operand Size

Byte

#### Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) as signed data and stores the result in the 16-bit register Rd. If Rd is one of general registers R0 to R7, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8 bits  $\times$  8 bits  $\rightarrow$  16 bits signed multiplication.



#### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	MULXS.B	Rs, Rd	0	1	C	0	5	0	rs	rd	4*

#### Notes

The number of states in the H8S/2000 CPU is 13.

## 2.2.42 (2) MULXS (W)

### MULXS (MULTIPLY eXtend as Signed)

### Multiply Signed

#### Operation

$ERd \times Rs \rightarrow ERd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

#### Assembly-Language Format

MULXS.W Rs, ERd

#### Operand Size

Word

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

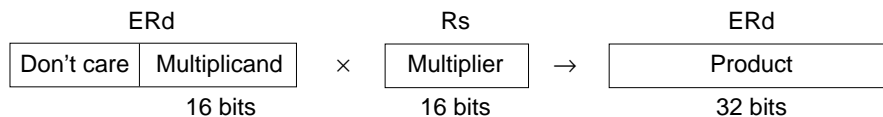
Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

#### Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) as signed data and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16 bits  $\times$  16 bits  $\rightarrow$  32 bits signed multiplication.



#### Available Registers

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MULXS.W	Rs, ERd	0 : 1	C : 0	5 : 2	rs : 0 : erd	5*

#### Notes

The number of states in the H8S/2000 CPU is 21.

## 2.2.43 (1) MULXU (B)

MULXU (MULTIPLY eXtend as Unsigned)

Multiply

### Operation

$Rd \times Rs \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

MULXU.B Rs, Rd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

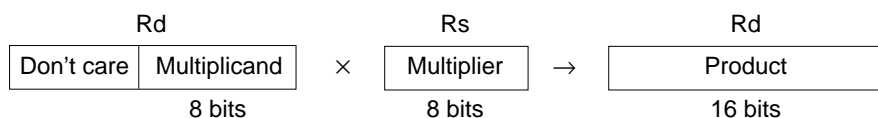
C: Previous value remains unchanged.

### Operand Size

Byte

### Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) as unsigned data and stores the result in the 16-bit register Rd. If Rd is one of general registers R0 to R7, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8 bits  $\times$  8 bits  $\rightarrow$  16 bits unsigned multiplication.



### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MULXU.B	Rs, Rd	5 0	rs rd			3*

### Notes

The number of states in the H8S/2000 CPU is 12.

## 2.2.43 (2) MULXU (W)

**MULXU (MULTiply eXtend as Unsigned)**

**Multiply**

### Operation

$ERd \times Rs \rightarrow ERd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

MULXU.W Rs, ERd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

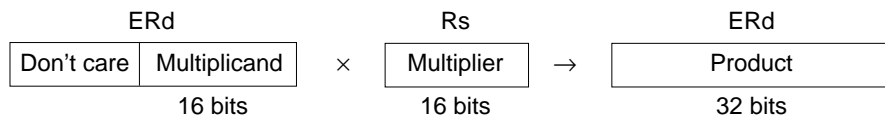
C: Previous value remains unchanged.

### Operand Size

Word

### Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) as unsigned data and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16 bits  $\times$  16 bits  $\rightarrow$  32 bits unsigned multiplication.



### Available Registers

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MULXU.W	Rs, ERd	5	2	rs	0:erd	4*

### Notes

The number of states in the H8S/2000 CPU is 20.

## 2.2.44 (1) NEG (B)

NEG (NEGate)

Negate Binary Signed

### Operation

0 – Rd → Rd

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

### Assembly-Language Format

NEG.B Rd

- H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

### Operand Size

Byte

### Description

This instruction takes the two's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd (subtracting the register contents from H'00). If the original contents of Rd were H'80, however, the result remains H'80.

### Available Registers

Rd: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NEG.B	Rd	1 : 7	8 : rd			1

### Notes

An overflow occurs if the original contents of Rd were H'80.

## 2.2.44 (2) NEG (W)

NEG (NEGate)

Negate Binary Signed

### Operation

0 – Rd → Rd

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

### Assembly-Language Format

NEG.W Rd

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

### Operand Size

Word

### Description

This instruction takes the two's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd (subtracting the register contents from H'0000). If the original contents of Rd were H'8000, however, the result remains H'8000.

### Available Registers

Rd: R0 to R7, E0 to E7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NEG.W	Rd	1    7	9    rd			1

### Notes

An overflow occurs if the original contents of Rd were H'8000.

### 2.2.44 (3) NEG (L)

NEG (NEGate)

Negate Binary Signed

---

#### Operation

0 – ERd → ERd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

---

#### Assembly-Language Format

NEG.L ERd

- H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

---

#### Operand Size

Longword

---

#### Description

This instruction takes the two's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd (subtracting the register contents from H'00000000). If the original contents of ERd were H'80000000, however, the result remains H'80000000.

---

#### Available Registers

ERd: ER0 to ER7

---

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NEG.L	ERd	1	7	B 0:erd		1

---

#### Notes

An overflow occurs if the original contents of ERd were H'80000000.

---

2.2.45 NOP

**NOP (No OPeration)** **No Operation**

**Operation**

PC + 2 → PC

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

**Assembly-Language Format**

NOP

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

**Operand Size**

—

**Description**

This instruction only increments the program counter, causing the next instruction to be executed. The internal state of the CPU does not change.

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	NOP		0 0	0 0			1

**Notes**



## 2.2.46 (1) NOT (B)

NOT (NOT = logical complement)

Logical Complement

### Operation

$\neg Rd \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↓	↓	0	—

### Assembly-Language Format

NOT.B Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

### Operand Size

Byte

### Description

This instruction takes the one's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### Available Registers

Rd: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NOT.B	Rd	1 : 7	0 : rd			1

### Notes

## 2.2.46 (2) NOT (W)

NOT (NOT = logical complement)

Logical Complement

### Operation

$\neg Rd \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

### Assembly-Language Format

NOT.W Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

### Operand Size

Word

### Description

This instruction takes the one's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

### Available Registers

Rd: R0 to R7, E0 to E7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NOT.W	Rd	1 : 7	1 : rd			1

### Notes

### 2.2.46 (3) NOT (L)

NOT (NOT = logical complement)

Logical Complement

---

#### Operation

$\neg \text{ERd} \rightarrow \text{ERd}$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

#### Assembly-Language Format

NOT.L ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

---

#### Operand Size

Longword

---

#### Description

This instruction takes the one's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

---

#### Available Registers

ERd: ER0 to ER7

---

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NOT.L	ERd	1 7	3 0:erd			1

---

#### Notes

---

## 2.2.47 (1) OR (B)

### OR (inclusive OR logical)

### Logical OR

#### Operation

Rd  $\vee$  (EAs)  $\rightarrow$  Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	$\updownarrow$	$\updownarrow$	0	—

#### Assembly-Language Format

OR.B <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Byte

### Description

This instruction ORs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	OR.B	#xx:8, Rd	C    rd	IMM			1
Register direct	OR.B	Rs, Rd	1    4	rs    rd			1

### Notes

## 2.2.47 (2) OR (W)

**OR (inclusive OR logical)**

**Logical OR**

---

### Operation

$Rd \vee (EAs) \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

---

### Assembly-Language Format

OR.W <EAs>, Rd

H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

---

### Operand Size

Word

---

### Description

This instruction ORs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

---

### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

---

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte		
Immediate	OR.W	#xx:16, Rd	7	9	4	rd	IMM	2
Register direct	OR.W	Rs, Rd	6	4	rs	rd		1

---

### Notes

---

## 2.2.47 (3) OR (L)

**OR (inclusive OR logical)**

**Logical OR**

### Operation

$ERd \vee (EAs) \rightarrow ERd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

### Assembly-Language Format

OR.L <EAs>, ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

### Operand Size

Longword

### Description

This instruction ORs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States				
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte	
Immediate	OR.L	#xx:32, ERd	7	A	4	0:erd	IMM								3
Register direct	OR.L	ERs, ERd	0	1	F	0	6	4	0:ers:0:erd					2	

### Notes

## 2.2.48 (1) ORC

ORC (inclusive OR Control register)

Logical OR with CCR

### Operation

CCR  $\vee$  #IMM  $\rightarrow$  CCR

### Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

### Assembly-Language Format

ORC #xx:8, CCR

I: Stores the corresponding bit of the result.

UI: Stores the corresponding bit of the result.

H: Stores the corresponding bit of the result.

U: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

### Operand Size

Byte

### Description

This instruction ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ORC	#xx:8, CCR	0 $\vdots$ 4	IMM			1

### Notes

## 2.2.48 (2) ORC

ORC (inclusive OR Control register)

Logical OR with EXR

### Operation

EXR  $\vee$  #IMM  $\rightarrow$  EXR

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

ORC #xx:8, EXR

(Example)

ORC #H'FF', EXR

H: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

### Operand Size

Byte

### Description

This instruction ORs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction (EXR rewrite instruction).

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	ORC	#xx:8, EXR	0 : 1	4 : 1	0 : 4	IMM	2

### Notes



## 2.2.49 (1) POP (W)

### POP (POP data)

### Pop Data from Stack

#### Operation

@SP+ → Rn

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

#### Assembly-Language Format

POP.W Rn

- H: Previous value remains unchanged.  
N: Set to 1 if the transferred data is negative; otherwise cleared to 0.  
Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction restores data from the stack to a 16-bit general register Rn, tests the restored data, and sets condition-code flags according to the result.

#### Available Registers

Rn: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	POP.W	Rn	6 : D	7 : m			3

#### Notes

POP.W Rn is identical to MOV.W @SP+, Rn.

## 2.2.49 (2) POP (L)

### POP (POP data)

### Pop Data from Stack

#### Operation

@SP+ → ERn

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

POP.L ERn

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Longword

#### Description

This instruction restores data from the stack to a 32-bit general register ERn, tests the restored data, and sets condition-code flags according to the result.

#### Available Registers

ERn: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	POP.L	ERn	0 1	0 0	6 D	7 0:ern	5

#### Notes

POP.L ERn is identical to MOV.L @SP+, ERn.

## 2.2.50 (1) PUSH (W)

### PUSH (PUSH data)

### Push Data on Stack

#### Operation

Rn → @-SP

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

#### Assembly-Language Format

PUSH.W Rn

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction saves data from a 16-bit register Rn onto the stack, tests the saved data, and sets condition-code flags according to the result.

#### Available Registers

Rn: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	PUSH.W	Rn	6 : D	F : rn			3

#### Notes

1. PUSH.W Rn is identical to MOV.W Rn, @-SP.
2. When PUSH.W R7 or PUSH.W E7 is executed, the value saved on the stack is the R7 or E7 value after effective address calculation (after ER7 is decremented by 2).

## 2.2.50 (2) PUSH (L)

### PUSH (PUSH data)

### Push Data on Stack

#### Operation

ERn → @-SP

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

PUSH.L ERn

H: Previous value remains unchanged.

N: Set to 1 if the transferred data is negative; otherwise cleared to 0.

Z: Set to 1 if the transferred data is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Longword

#### Description

This instruction pushes data from a 32-bit register ERn onto the stack, tests the saved data, and sets condition-code flags according to the result.

#### Available Registers

ERn: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format							No. of States
			1st byte		2nd byte		3rd byte		4th byte	
—	PUSH.L	ERn	0	1	0	0	6	D	F 0:ern	5

#### Notes

1. PUSH.L ERn is identical to MOV.L ERn, @-SP.
2. When PUSH.L ER7 is executed, the value saved on the stack is the ER7 value after effective address calculation (after ER7 is decremented by 4).

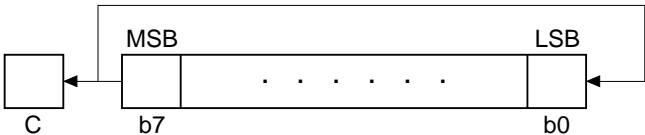
2.2.51 (1) ROTL (B)

ROTL (ROtate Left) Rotate

Operation	Condition Code																
Rd (left rotation) → Rd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>0</td><td>↕</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	0	↕
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	0	↕										
<b>Assembly-Language Format</b>	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 7.																
ROTL.B Rd																	
<b>Operand Size</b>																	
Byte																	

**Description**

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) is rotated to the least significant bit (bit 0), and also copied to the carry flag.



**Available Registers**

Rd: R0L to R7L, R0H to R7H

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.B	Rd	1    2	8    rd			1

**Notes**

2.2.51 (2) ROTL (B)

ROTL (ROtate Left) Rotate

**Operation**

Rd (left rotation) → Rd

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

**Assembly-Language Format**

ROTL.B #2, Rd

(Example)

ROTL.B #2, R1L

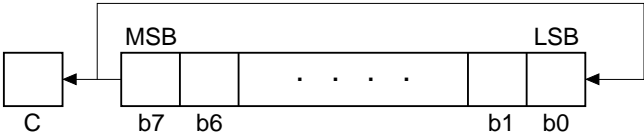
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 6.

**Operand Size**

Byte

**Description**

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the left. The most significant two bits (bits 7 and 6) are rotated to the least significant two bits (bits 1 and 0), and bit 6 is also copied to the carry flag.



**Available Registers**

Rd: R0L to R7L, R0H to R7H

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.B	#2, Rd	1    2	C    rd			1

**Notes**

### 2.2.51 (3) ROTL (W)

#### ROTL (ROtate Left)

Rotate

##### Operation

Rd (left rotation) → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

##### Assembly-Language Format

ROTL.W Rd

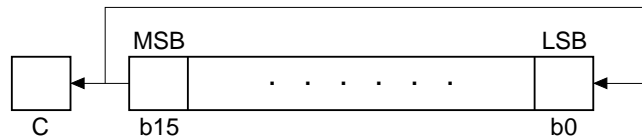
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 15.

##### Operand Size

Word

##### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) is rotated to the least significant bit (bit 0), and also copied to the carry flag.



##### Available Registers

Rd: R0 to R7, E0 to E7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.W	Rd	1	2	9	rd	1

##### Notes

2.2.51 (4) ROTL (W)

ROTL (ROtate Left) Rotate

**Operation**

Rd (left rotation) → Rd

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

**Assembly-Language Format**

ROTL.W #2, Rd

(Example)

ROTL.W #2, R3

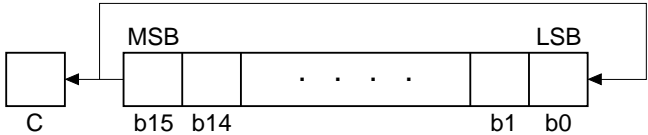
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 14.

**Operand Size**

Word

**Description**

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the left. The most significant two bits (bits 15 and 14) are rotated to the least significant two bits (bits 1 and 0), and bit 14 is also copied to the carry flag.



**Available Registers**

Rd: R0 to R7, E0 to E7

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.W	#2, Rd	1    2	D    rd			1

**Notes**



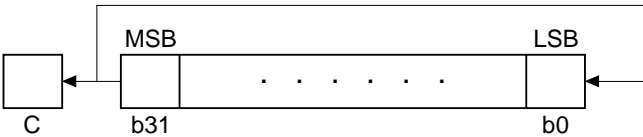
2.2.51 (5) ROTL (L)

ROTL (ROtate Left) Rotate

Operation	Condition Code																
ERd (left rotation) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>0</td><td>↕</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	0	↕
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	0	↕										
<hr/> <b>Assembly-Language Format</b> ROTL.L ERd	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 31.																
<hr/> <b>Operand Size</b> Longword																	

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) is rotated to the least significant bit (bit 0), and also copied to the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.L	ERd	1	2	B 0:erd		1

Notes

2.2.51 (6) ROTL (L)

ROTL (ROtate Left) Rotate

Operation

ERd (left rotation) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTL.L #2, ERd

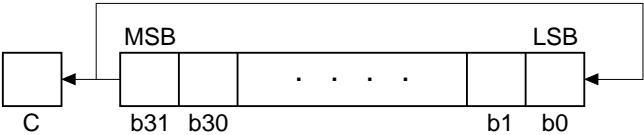
Operand Size

Longword

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 30.

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the left. The most significant two bits (bits 31 and 30) are rotated to the least significant two bits (bits 1 and 0), and bit 30 is also copied to the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.L	#2, ERd	1 2	F 0	erd		1

Notes

## 2.2.52 (1) ROTR (B)

### ROTR (ROtate Right)

Rotate

#### Operation

Rd (right rotation) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

ROTR.B Rd

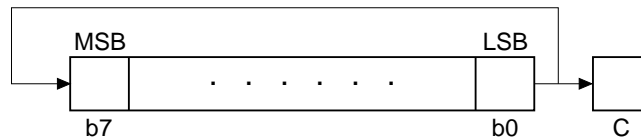
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

#### Operand Size

Byte

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 7), and also copied to the carry flag.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.B	Rd	1 : 3	8 : rd			1

#### Notes

2.2.52 (2) ROTR (B)

ROTR (ROTate Right)

Rotate

Operation

Rd (right rotation) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTR.B #2, Rd

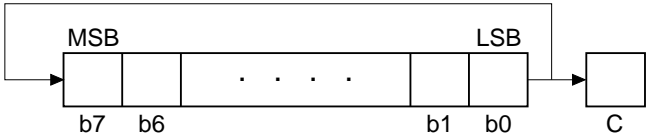
Operand Size

Byte

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 7 and 6), and bit 1 is also copied to the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.B	#2, Rd	1    3	C    rd			1

Notes

## 2.2.52 (3) ROTR (W)

### ROTR (ROTate Right)

Rotate

#### Operation

Rd (right rotation) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

ROTR.W Rd

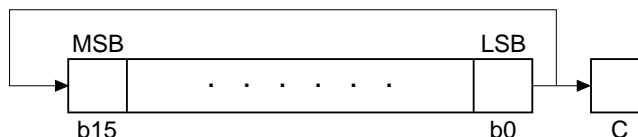
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 0.

#### Operand Size

Word

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 15), and also copied to the carry flag.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.W	Rd	1 3	9 rd			1

#### Notes

2.2.52 (4) ROTR (W)

ROTR (ROtate Right)

Rotate

Operation

Rd (right rotation) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTR.W #2, Rd

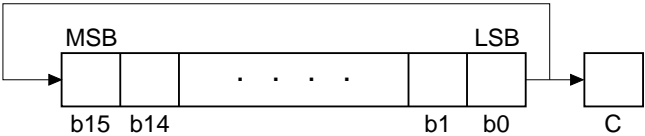
Operand Size

Word

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 15 and 14), and bit 1 is also copied to the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.W	#2, Rd	1    3	D    rd			1

Notes

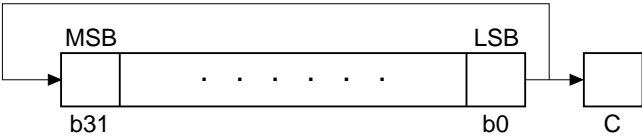
2.2.52 (5) ROTR (L)

ROTR (ROTate Right) Rotate

Operation	Condition Code																
ERd (right rotation) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>0</td><td>↕</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	0	↕
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	0	↕										
<hr/>																	
<b>Assembly-Language Format</b>																	
ROTR.L ERd																	
<hr/>																	
<b>Operand Size</b>																	
Longword																	
	H: Previous value remains unchanged.																
	N: Set to 1 if the result is negative; otherwise cleared to 0.																
	Z: Set to 1 if the result is zero; otherwise cleared to 0.																
	V: Always cleared to 0.																
	C: Receives the previous value in bit 0.																

**Description**

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the right. The least significant bit (bit 0) is rotated to the most significant bit (bit 31), and also copied to the carry flag.



**Available Registers**

ERd: ER0 to ER7

**Operand Format and Number of States Required for Execution**

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.L	ERd	1 3	B 0 erd			1

**Notes**

2.2.52 (6) ROTR (L)

ROTR (ROtate Right)

Rotate

Operation

ERd (right rotation) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTR.L #2, ERd

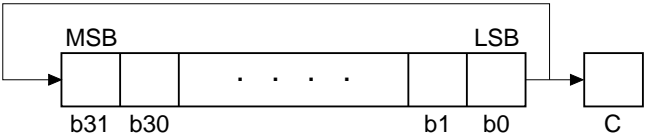
Operand Size

Longword

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the right. The least significant two bits (bits 1 and 0) are rotated to the most significant two bits (bits 31 and 30), and bit 1 is also copied to the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.L	#2, ERd	1 3	F 0:erd			1

Notes



### 2.2.53 (1) ROTXL (B)

**ROTXL (ROtate with eXtend carry Left)**

**Rotate through Carry**

#### Operation

Rd (left rotation through carry flag) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

ROTXL.B Rd

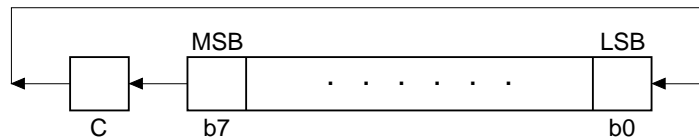
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 7.

#### Operand Size

Byte

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 7) rotates into the carry flag.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.B	Rd	1	2	0	rd	1

#### Notes

2.2.53 (2) ROTXL (B)

ROTXL (ROtate with eXtend carry Left)

Rotate through Carry

Operation

Rd (left rotation through carry flag) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXL.B #2, Rd

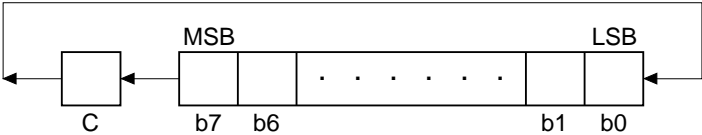
Operand Size

Byte

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 6.

Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 7 rotates into bit 0, and bit 6 rotates into the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.B	#2, Rd	1	2	4	rd	1

Notes

### 2.2.53 (3) ROTXL (W)

#### ROTXL (ROtate with eXtend carry Left)

#### Rotate through Carry

##### Operation

Rd (left rotation through carry flag) → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

##### Assembly-Language Format

ROTXL.W Rd

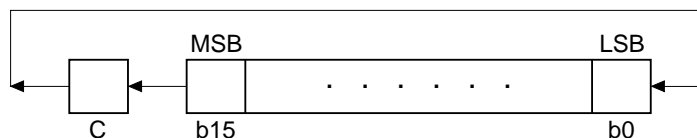
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 15.

##### Operand Size

Word

#### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 15) rotates into the carry flag.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.W	Rd	1	2	1	rd	1

#### Notes

2.2.53 (4) ROTXL (W)

ROTXL (ROtate with eXtend carry Left)

Rotate through Carry

Operation

Rd (left rotation through carry flag) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXL.W #2, Rd

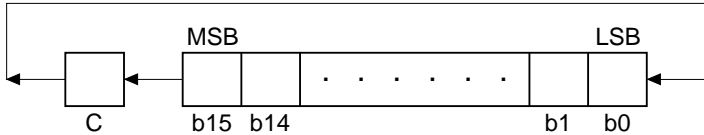
Operand Size

Word

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 14.

Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 15 rotates into bit 0, and bit 14 rotates into the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.W	#2, Rd	1    2	5    rd			1

Notes

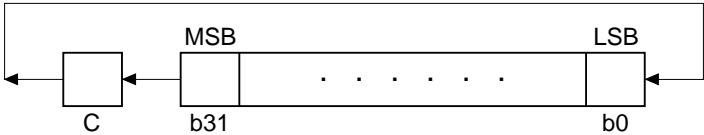
2.2.53 (5) ROTXL (L)

ROTXL (ROtate with eXtend carry Left) Rotate through Carry

Operation	Condition Code																
ERd (left rotation through carry flag) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>0</td><td>↕</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	0	↕
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	0	↕										
<b>Assembly-Language Format</b> ROTXL.L ERd	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 31.																
<b>Operand Size</b> Longword																	

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit (bit 31) rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.L	ERd	1	2	3	0:erd	1

Notes

2.2.53 (6) ROTXL (L)

ROTXL (ROtate with eXtend carry Left)

Rotate through Carry

Operation

ERd (left rotation through carry flag) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXL.L #2, ERd

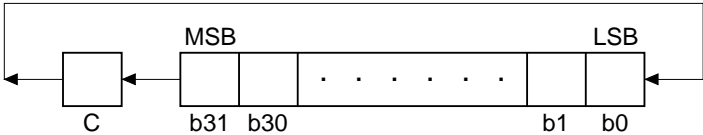
Operand Size

Longword

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 30.

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the left through the carry flag. The carry flag rotates into bit 1, bit 31 rotates into bit 0, and bit 30 rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.L	#2, ERd	1	2	7 : 0 : erd		1

Notes

## 2.2.54 (1) ROTXR (B)

### ROTXR (ROtate with eXtend carry Right)

### Rotate through Carry

#### Operation

Rd (right rotation through carry flag) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

ROTXR.B Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

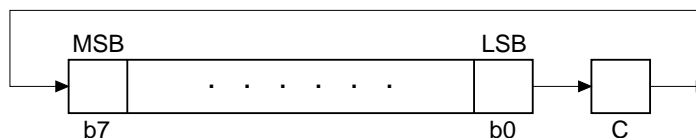
C: Receives the previous value in bit 0.

#### Operand Size

Byte

#### Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 7). The least significant bit (bit 0) rotates into the carry flag.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.B	Rd	1 3	0 rd			1

#### Notes

2.2.54 (2) ROTXR (B)

ROTXR (ROtate with eXtend carry Right)

Rotate through Carry

Operation

Rd (right rotation through carry flag) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXR.B #2, Rd

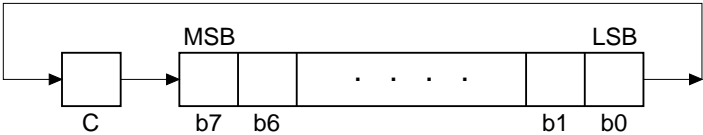
Operand Size

Byte

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in an 8-bit register Rd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 6, bit 0 rotates into bit 7, and bit 1 rotates into the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.B	#2, Rd	1    3	4    rd			1

Notes



### 2.2.54 (3) ROTXR (W)

#### ROTXR (ROtate with eXtend carry Right)

#### Rotate through Carry

##### Operation

Rd (right rotation through carry flag) → Rd

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

##### Assembly-Language Format

ROTXR.W Rd

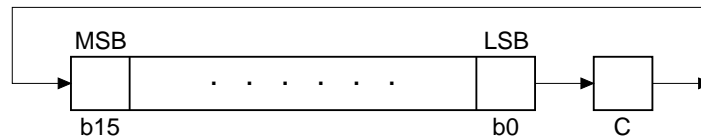
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 0.

##### Operand Size

Word

##### Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 15). The least significant bit (bit 0) rotates into the carry flag.



##### Available Registers

Rd: R0 to R7, E0 to E7

##### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.W	Rd	1 3	1 rd			1

##### Notes

2.2.54 (4) ROTXR (W)

ROTXR (ROtate with eXtend carry Right)

Rotate through Carry

Operation

Rd (right rotation through carry flag) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXR.W #2, Rd

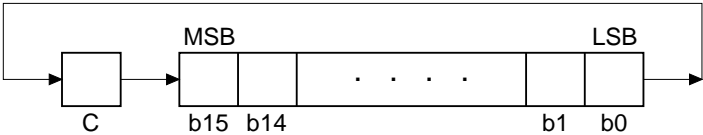
Operand Size

Word

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in a 16-bit register Rd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 14, bit 0 rotates into bit 15, and bit 1 rotates into the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.W	#2, Rd	1    3	5    rd			1

Notes

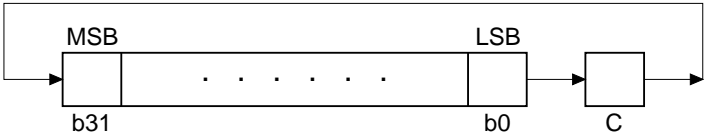
2.2.54 (5) ROTXR (L)

ROTXR (ROtate with eXtend carry Right) Rotate through Carry

Operation	Condition Code																
ERd (right rotation through carry flag) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↑</td><td>↑</td><td>0</td><td>↑</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↑	↑	0	↑
I	UI	H	U	N	Z	V	C										
—	—	—	—	↑	↑	0	↑										
<b>Assembly-Language Format</b> ROTXR.L ERd	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 0.																
<b>Operand Size</b> Longword																	

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 31). The least significant bit (bit 0) rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.L	ERd	1 3	3 0:erd			1

Notes

2.2.54 (6) ROTXR (L)

ROTXR (ROtate with eXtend carry Right)

Rotate through Carry

Operation

ERd (right rotation through carry flag) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

ROTXR.L #2, ERd

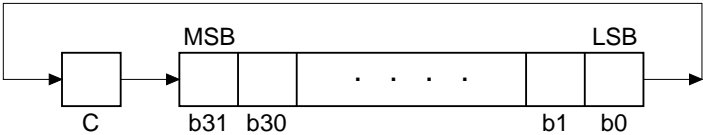
Operand Size

Longword

- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 1.

Description

This instruction rotates the bits in a 32-bit register ERd (destination operand) two bits to the right through the carry flag. The carry flag rotates into bit 30, bit 0 rotates into bit 31, and bit 1 rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.L	#2, ERd	1 3	7 0:erd			1

Notes

## 2.2.55 RTE

### RTE (ReTurn from Exception)

### Return from Exception Handling

#### Operation

- When EXR is invalid  
@SP+ → CCR  
@SP+ → PC
- When EXR is valid  
@SP+ → EXR  
@SP+ → CCR  
@SP+ → PC

#### Assembly-Language Format

RTE

#### Operand Size

—

#### Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

- I: Restored from the corresponding bit on the stack.
- UI: Restored from the corresponding bit on the stack.
- H: Restored from the corresponding bit on the stack.
- U: Restored from the corresponding bit on the stack.
- N: Restored from the corresponding bit on the stack.
- Z: Restored from the corresponding bit on the stack.
- V: Restored from the corresponding bit on the stack.
- C: Restored from the corresponding bit on the stack.

#### Description

This instruction returns from an exception-handling routine by restoring the EXR, condition-code register (CCR) and program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The CCR and PC contents at the time of execution of this instruction are lost. If the extended control register (EXR) is valid, it is also restored (and the existing EXR contents are lost).

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	RTE		5	6	7	0	5*

**Note:**\*Six states when EXR is valid.

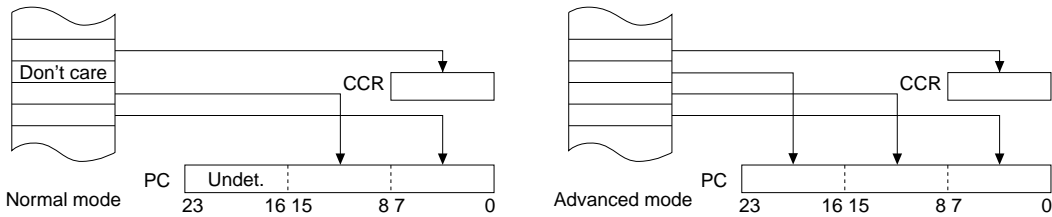
2.2.55 RTE

RTE (ReTurn from Exception)

Return from Exception Handling

Notes

The stack structure differs between normal mode and advanced mode.



2.2.56 RTS

RTS (ReTurn from Subroutine)

Return from Subroutine

<b>Operation</b> @SP+ → PC	<b>Condition Code</b> <table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	—	—	—	—
I	UI	H	U	N	Z	V	C										
—	—	—	—	—	—	—	—										
<b>Assembly-Language Format</b> RTS	H: Previous value remains unchanged. N: Previous value remains unchanged. Z: Previous value remains unchanged. V: Previous value remains unchanged. C: Previous value remains unchanged.																
<b>Operand Size</b> —																	

**Description**

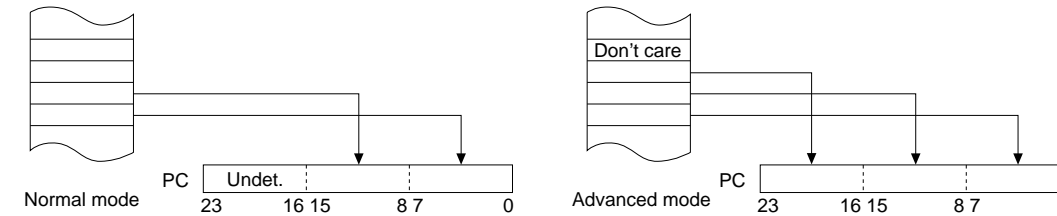
This instruction returns from a subroutine by restoring the program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The PC contents at the time of execution of this instruction are lost.

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st Byte	2nd Byte	3rd Byte	4th Byte	Normal	Advanced
—	RTS		5 4	7 0			4	5

**Notes**

The stack structure and number of states required for execution differ between normal mode and advanced mode. In normal mode, only the lower 16 bits of the program counter are restored.



2.2.57 (1) SHAL (B)

SHAL (SHift Arithmetic Left)

Shift Arithmetic

Operation

Rd (left arithmetic shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

Assembly-Language Format

SHAL.B Rd

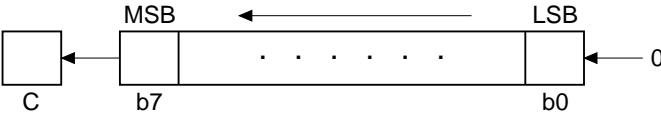
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Set to 1 if an overflow occurs; otherwise cleared to 0.  
C: Receives the previous value in bit 7.

Operand Size

Byte

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.B	Rd	1 : 0	8 : rd			1

Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.



## 2.2.57 (2) SHAL (B)

### SHAL (SHift Arithmetic Left)

### Shift Arithmetic

#### Operation

Rd (left arithmetic shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	↑	↑

#### Assembly-Language Format

SHAL.B #2, Rd

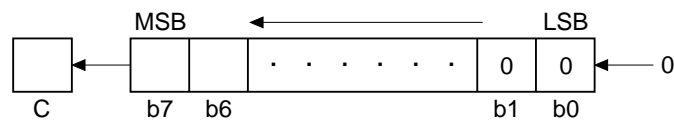
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Set to 1 if an overflow occurs; otherwise cleared to 0.  
 C: Receives the previous value in bit 6.

#### Operand Size

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the left. Bit 6 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.B	#2, Rd	1 0	C rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

## 2.2.57 (3) SHAL (W)

### SHAL (SHift Arithmetic Left)

### Shift Arithmetic

#### Operation

Rd (left arithmetic shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

#### Assembly-Language Format

SHAL.W Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

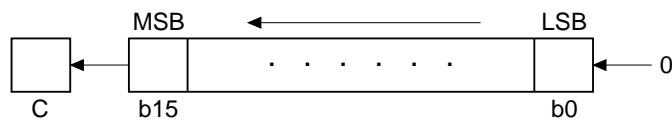
C: Receives the previous value in bit 15.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.W	Rd	1 0	9 rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

## 2.2.57 (4) SHAL (W)

### SHAL (SHift Arithmetic Left)

### Shift Arithmetic

#### Operation

Rd (left arithmetic shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	↑	↑

#### Assembly-Language Format

SHAL.W #2, Rd

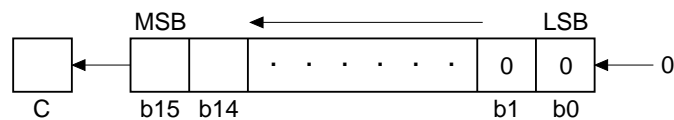
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Set to 1 if an overflow occurs; otherwise cleared to 0.  
 C: Receives the previous value in bit 14.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the left. Bit 14 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.W	#2, Rd	1 : 0	D : rd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

2.2.57 (5) SHAL (L)

SHAL (SHift Arithmetic Left)

Shift Arithmetic

Operation

ERd (left arithmetic shift) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

Assembly-Language Format

SHAL.L ERd

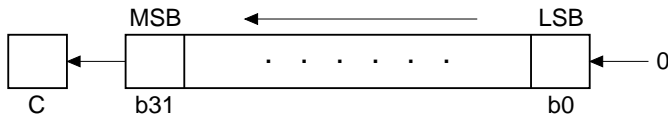
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Set to 1 if an overflow occurs; otherwise cleared to 0.  
C: Receives the previous value in bit 31.

Operand Size

Longword

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.L	ERd	1 0	B 0:erd			1

Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

## 2.2.57 (6) SHAL (L)

### SHAL (SHift Arithmetic Left)

### Shift Arithmetic

#### Operation

ERd (left arithmetic shift) → ERd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

#### Assembly-Language Format

SHAL.L #2, ERd

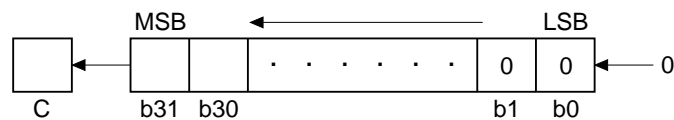
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Set to 1 if an overflow occurs; otherwise cleared to 0.  
 C: Receives the previous value in bit 30.

#### Operand Size

Longword

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the left. Bit 30 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.L	#2, ERd	1 0	F 0:erd			1

#### Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

2.2.58 (1) SHAR (B)

SHAR (SHift Arithmetic Right)

Shift Arithmetic

Operation

Rd (right arithmetic shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHAR.B Rd

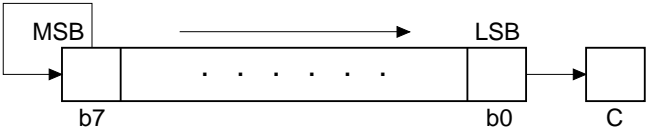
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

Operand Size

Byte

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 7 shifts into itself. Since bit 7 remains unaltered, the sign does not change.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.B	Rd	1	1	8	rd	1

Notes

## 2.2.58 (2) SHAR (B)

### SHAR (SHift Arithmetic Right)

### Shift Arithmetic

#### Operation

Rd (right arithmetic shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

SHAR.B #2, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

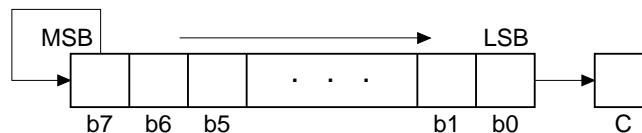
C: Receives the previous value in bit 1.

#### Operand Size

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. The most significant three bits (bits 7, 6, and 5) all receive the previous value of bit 7. Since bit 7 remains unaltered, the sign does not change.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.B	#2, Rd	1	1	C	rd	1

#### Notes

2.2.58 (3) SHAR (W)

SHAR (SHift Arithmetic Right)

Shift Arithmetic

Operation

Rd (right arithmetic shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHAR.W Rd

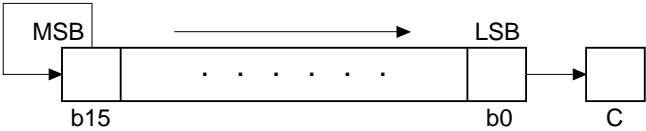
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

Operand Size

Word

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 15 shifts into itself. Since bit 15 remains unaltered, the sign does not change.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.W	Rd	1 : 1	9 : rd			1

Notes



## 2.2.58 (4) SHAR (W)

### SHAR (SHift Arithmetic Right)

### Shift Arithmetic

#### Operation

Rd (right arithmetic shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

SHAR.W #2, Rd

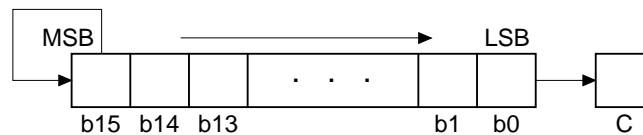
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 1.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. The most significant three bits (bits 15, 14, and 13) all receive the previous value of bit 15. Since bit 15 remains unaltered, the sign does not change.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.W	#2, Rd	1 1	D rd			1

#### Notes

2.2.58 (5) SHAR (L)

SHAR (SHift Arithmetic Right)

Shift Arithmetic

Operation

ERd (right arithmetic shift) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHAR.L ERd

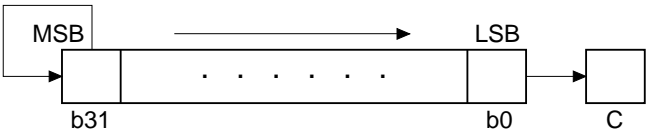
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

Operand Size

Longword

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 31 shifts into itself. Since bit 31 remains unaltered, the sign does not change.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.L	ERd	1	1	B	0:erd	1

Notes

## 2.2.58 (6) SHAR (L)

### SHAR (SHift Arithmetic Right)

### Shift Arithmetic

#### Operation

ERd (right arithmetic shift) → ERd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

SHAR.L #2, ERd

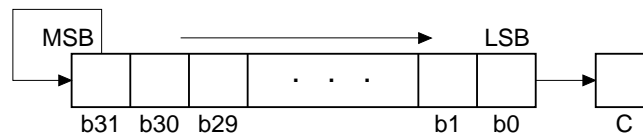
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 1.

#### Operand Size

Longword

#### Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. The most significant three bits (bits 31, 30, and 29) all receive the previous value of bit 31. Since bit 31 remains unaltered, the sign does not change.



#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.L	#2, ERd	1	1	F 0:erd		1

#### Notes

2.2.59 (1) SHLL (B)

SHLL (SHift Logical Left)

Shift Logical

Operation

Rd (left logical shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHLL.B Rd

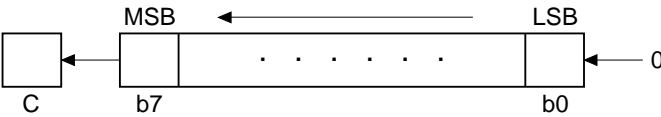
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 7.

Operand Size

Byte

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 7) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.B	Rd	1 0	0 rd			1

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

## 2.2.59 (2) SHLL (B)

### SHLL (SHift Logical Left)

Shift Logical

#### Operation

Rd (left logical shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

SHLL.B #2, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

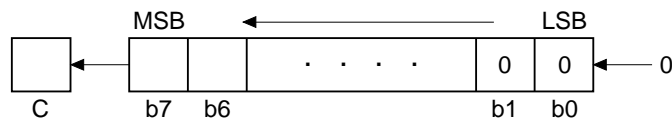
C: Receives the previous value in bit 6.

#### Operand Size

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the left. Bit 6 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.B	#2, Rd	1    0	4    rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.59 (3) SHLL (W)

SHLL (SHift Logical Left)

Shift Logical

Operation

Rd (left logical shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHLL.W Rd

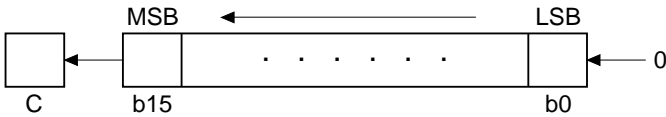
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 15.

Operand Size

Word

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit (bit 15) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.W	Rd	1 : 0	1 : rd			1

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

## 2.2.59 (4) SHLL (W)

### SHLL (SHift Logical Left)

Shift Logical

#### Operation

Rd (left logical shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	↑

#### Assembly-Language Format

SHLL.W #2, Rd

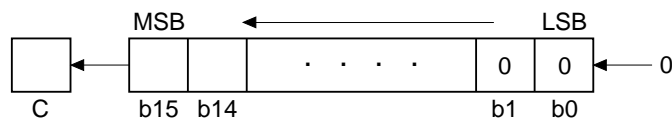
- H: Previous value remains unchanged.  
 N: Set to 1 if the result is negative; otherwise cleared to 0.  
 Z: Set to 1 if the result is zero; otherwise cleared to 0.  
 V: Always cleared to 0.  
 C: Receives the previous value in bit 14.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the left. Bit 14 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.W	#2, Rd	1 0	5 rd			1

#### Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.59 (5) SHLL (L)

SHLL (SHift Logical Left)

Shift Logical

Operation

ERd (left logical shift) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

Assembly-Language Format

SHLL.L ERd

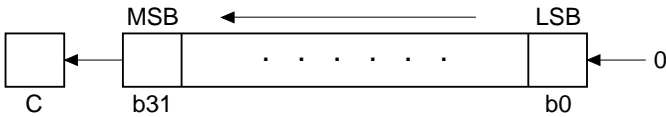
- H: Previous value remains unchanged.  
N: Set to 1 if the result is negative; otherwise cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 31.

Operand Size

Longword

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit (bit 31) shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.L	ERd	1 0	3 0erd			1

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.



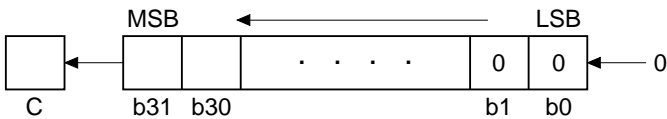
2.2.59 (6) SHLL (L)

SHLL (SHift Logical Left) Shift Logical

Operation	Condition Code																
ERd (left logical shift) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>↕</td><td>↕</td><td>0</td><td>↕</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	↕	↕	0	↕
I	UI	H	U	N	Z	V	C										
—	—	—	—	↕	↕	0	↕										
<hr/> <b>Assembly-Language Format</b> SHLL.L #2, ERd	H: Previous value remains unchanged. N: Set to 1 if the result is negative; otherwise cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 30.																
<hr/> <b>Operand Size</b> Longword																	

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the left. Bit 30 shifts into the carry flag. Bits 0 and 1 are cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.L	#2, ERd	1 0	7 0:erd			1

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.60 (1) SHLR (B)

SHLR (SHift Logical Right)

Shift Logical

Operation

Rd (right logical shift) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

Assembly-Language Format

SHLR.B Rd

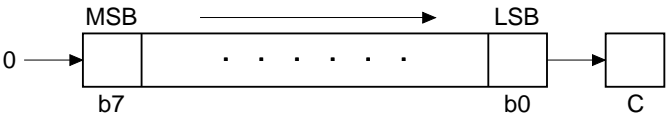
- H: Previous value remains unchanged.  
N: Always cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

Operand Size

Byte

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 7) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.B	Rd	1	1	0	rd	1

Notes

## 2.2.60 (2) SHLR (B)

### SHLR (SHift Logical Right)

Shift Logical

#### Operation

Rd (right logical shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↑	0	↑

#### Assembly-Language Format

SHLR.B #2, Rd

H: Previous value remains unchanged.

N: Always cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

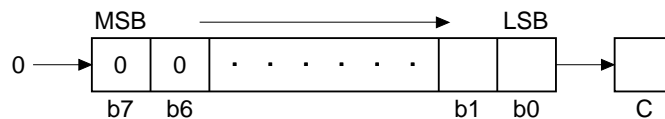
C: Receives the previous value in bit 1.

#### Operand Size

Byte

#### Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 7 and 6 are cleared to 0.



#### Available Registers

Rd: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.B	#2, Rd	1    1	4    rd			1

#### Notes

## 2.2.60 (3) SHLR (W)

### SHLR (SHift Logical Right)

Shift Logical

#### Operation

Rd (right logical shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

#### Assembly-Language Format

SHLR.W Rd

H: Previous value remains unchanged.

N: Always cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

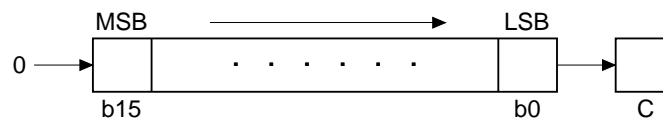
C: Receives the previous value in bit 0.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 15) is cleared to 0.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.W	Rd	1 : 1	1 : rd			1

#### Notes

## 2.2.60 (4) SHLR (W)

### SHLR (SHift Logical Right)

Shift Logical

#### Operation

Rd (right logical shift) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↑	0	↑

#### Assembly-Language Format

SHLR.W #2, Rd

H: Previous value remains unchanged.

N: Always cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

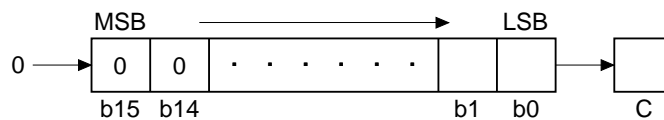
C: Receives the previous value in bit 1.

#### Operand Size

Word

#### Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 15 and 14 are cleared to 0.



#### Available Registers

Rd: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.W	#2, Rd	1    1	5    rd			1

#### Notes

2.2.60 (5) SHLR (L)

SHLR (SHift Logical Right)

Shift Logical

Operation

ERd (right logical shift) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

Assembly-Language Format

SHLR.L ERd

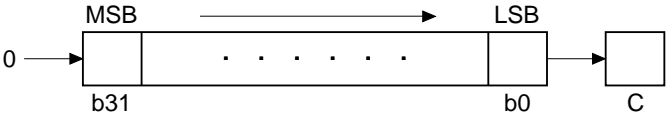
- H: Previous value remains unchanged.  
N: Always cleared to 0.  
Z: Set to 1 if the result is zero; otherwise cleared to 0.  
V: Always cleared to 0.  
C: Receives the previous value in bit 0.

Operand Size

Longword

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. The least significant bit (bit 0) shifts into the carry flag. The most significant bit (bit 31) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.L	ERd	1	3	0:erd		1

Notes

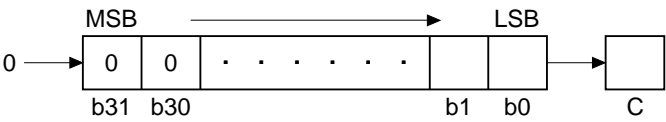
2.2.60 (6) SHLR (L)

SHLR (SHift Logical Right) Shift Logical

Operation	Condition Code																
ERd (right logical shift) → ERd	<table><tr><td>I</td><td>UI</td><td>H</td><td>U</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>0</td><td>↑</td><td>0</td><td>↑</td></tr></table>	I	UI	H	U	N	Z	V	C	—	—	—	—	0	↑	0	↑
I	UI	H	U	N	Z	V	C										
—	—	—	—	0	↑	0	↑										
<hr/> <b>Assembly-Language Format</b> SHLR.L #2, ERd	H: Previous value remains unchanged. N: Always cleared to 0. Z: Set to 1 if the result is zero; otherwise cleared to 0. V: Always cleared to 0. C: Receives the previous value in bit 1.																
<hr/> <b>Operand Size</b> Longword																	

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) two bits to the right. Bit 1 shifts into the carry flag. Bits 31 and 30 are cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.L	#2, ERd	1	1	7	0:erd	1

Notes

## 2.2.61 SLEEP

### SLEEP (SLEEP)

### Power-Down Mode

#### Operation

Program execution state → power-down mode

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

#### Assembly-Language Format

SLEEP

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

#### Operand Size

—

#### Description

When the SLEEP instruction is executed, the CPU enters a power-down mode. Its internal state remains unchanged, but the CPU stops executing instructions and waits for an exception-handling request. When it receives an exception-handling request, the CPU exits the power-down mode and begins the exception-handling sequence. Interrupt requests other than NMI cannot end the power-down mode if they are masked in the CPU.

#### Available Registers

—

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	SLEEP		0 : 1	8 : 0			2

#### Notes

For information about power-down modes, see the relevant microcontroller hardware manual.



## 2.2.62 (1) STC (B)

**STC (STore from Control register)**

**Store CCR**

### Operation

CCR → Rd

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

STC.B CCR, Rd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### Operand Size

Byte

### Description

This instruction copies the CCR contents to an 8-bit register Rd.

### Available Registers

Rd: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	STC.B	CCR, Rd	0    2	0    rd			1

### Notes

2.2.62 (2) STC (B)

STC (STore from Control register)

Store EXR

Operation

EXR → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

Assembly-Language Format

STC.B EXR, Rd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Operand Size

Byte

Description

This instruction copies the EXR contents to an 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	STC.B	EXR, Rd	0 : 2	1 : rd			1

Notes

### 2.2.62 (3) STC (W)

**STC (STore from Control register)**

**Store CCR**

---

**Operation**

CCR → (EAd)

---

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

---

**Assembly-Language Format**

STC.W CCR, <EAd>

---

**Operand Size**

Word

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

**Description**

This instruction copies the CCR contents to a destination location. Although CCR is a byte register, the destination operand is a word operand. The CCR contents are stored at the even address. Undetermined data is stored at the odd address.

---

**Available Registers**

ERd: ER0 to ER7

## Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format												No. of States								
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte			7th byte		8th byte		9th byte		10th byte	
Register indirect	STC.W	CCR, @ERd	0	1	4	0	6	9	1:erd	0													3
Register indirect with displacement	STC.W	CCR, @(d:16, ERd)	0	1	4	0	6	F	1:erd	0	disp												4
	STC.W	CCR, @(d:32, ERd)	0	1	4	0	7	8	0:erd	0	6	B	A	0	disp								6
Register indirect with pre-decrement	STC.W	CCR, @-ERd	0	1	4	0	6	D	1:erd	0													4
Absolute address	STC.W	CCR, @aa:16	0	1	4	0	6	B	8	0	abs											4	
	STC.W	CCR, @aa:32	0	1	4	0	6	B	A	0	abs												5

### Notes

2.2.62 (3) STC (W)

STC (Store from Control register)

Store CCR

## 2.2.62 (4) STC (W)

STC (STore from Control register)

Store EXR

---

### Operation

EXR → (EAd)

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

---

### Assembly-Language Format

STC.W EXR, <EAd>

---

### Operand Size

Word

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

### Description

This instruction copies the EXR contents to a destination location. Although EXR is a byte register, the destination operand is a word operand. The EXR contents are stored at the even address. Undetermined data is stored at the odd address.

---

### Available Registers

ERd: ER0 to ER7

## Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format												No. of States								
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte			7th byte		8th byte		9th byte		10th byte	
Register indirect	STC.W	EXR, @ERd	0	1	4	1	6	9	1:erd	0													3
Register indirect with displacement	STC.W	EXR, @(d:16, ERd)	0	1	4	1	6	F	1:erd	0	disp												4
	STC.W	EXR, @(d:32, ERd)	0	1	4	1	7	8	0:erd	0	6	B	A	0	disp						6		
Register indirect with pre-decrement	STC.W	EXR, @-ERd	0	1	4	1	6	D	1:erd	0													4
Absolute address	STC.W	EXR, @aa:16	0	1	4	1	6	B	8	0	abs												4
	STC.W	EXR, @aa:32	0	1	4	1	6	B	A	0	abs										5		

### Notes

STC (STore from Control register)

Store EXR

2.2.62 (4) STC (W)

### 2.2.63 STM

#### STM (STore from Multiple registers)

#### Store Data on Stack

---

**Operation**

ERn (register list) → @-SP

---

**Condition Code**

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

---

**Assembly-Language Format**

STM.L <register list>, @-SP

---

**Operand Size**

Longword

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

---

**Description**

This instruction saves a group of registers specified by a register list onto the stack. The registers are saved in ascending order of register number.

Two, three, or four registers can be saved by one STM instruction. The following ranges can be specified in the register list.

Two registers: ER0-ER1, ER2-ER3, ER4-ER5, or ER6-ER7

Three registers: ER0-ER2 or ER4-ER6

Four registers: ER0-ER3 or ER4-ER7

---

**Available Registers**

ERn: ER0 to ER7

### 2.2.63 STM

STM (STore from Multiple registers)

Store Data on Stack

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
–	STM.L	(ERn–ERn+1), @_SP	0	1	1	0	6	D	F	0: ern	7
–	STM.L	(ERn–ERn+2), @_SP	0	1	2	0	6	D	F	0: ern	9
–	STM.L	(ERn–ERn+3), @_SP	0	1	3	0	6	D	F	0: ern	11

#### Notes

When ER7 is saved, the value after effective address calculation (after ER7 is decremented by 4) is saved on the stack.



## 2.2.64 STMAC

STMAC (STore from MAC register)

Store Data from MAC Register

### Operation

MACH → ERd

or

MACL → ERd

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕*	↕*	↕*	—

### Assembly-Language Format

STMAC MAC register, ERd

### Operand Size

Longword

- H: Previous value remains unchanged.  
 N: Set to 1 if a MAC instruction resulted in a negative MAC register value; otherwise cleared to 0.  
 Z: Set to 1 if a MAC instruction resulted in a zero MAC register value; otherwise cleared to 0.  
 V: Set to 1 if a MAC instruction resulted in an overflow; otherwise cleared to 0.  
 C: Previous value remains unchanged.

### Description

This instruction moves the contents of a multiply-accumulate register (MACH or MACL) to a general register. If the transfer is from MACH, the upper 22 bits transferred to the general register are a sign extension.

This instruction is supported by the H8S/2600 CPU only.

### Available Registers

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	STMAC	MACH, ERd	0 2	2 0 ers			1
Register direct	STMAC	MACL, ERd	0 2	3 0 ers			1

### Notes

Execution of this instruction copies the N, Z, and V flag values from the multiplier to the condition-code register (CCR). If the STMAC instruction is executed after a CLRMAC or LDMAC instruction with no intervening MAC instruction, the V flag will be 0 and the N and Z flags will have undetermined values.

## 2.2.65 (1) SUB (B)

SUB (SUBtract binary)

Subtract Binary

### Operation

$Rd - Rs \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

### Assembly-Language Format

SUB.B Rs, Rd

H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

### Operand Size

Byte

### Description

This instruction subtracts the contents of an 8-bit register Rs (source operand) from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SUB.B	Rs, Rd	1    8	rs    rd			1

## 2.2.65 (1) SUB (B)

### SUB (SUBtract binary)

### Subtract Binary

---

#### Notes

The SUB.B instruction can operate only on general registers. Immediate data can be subtracted from general register contents by using the SUBX instruction. Before executing SUBX #xx:8, Rd, first set the Z flag to 1 and clear the C flag to 0. The following coding examples can also be used to subtract nonzero immediate data #IMM.

- (1) ORC     #H'05,CCR  
      SUBX   #(IMM-1),Rd
- (2) ADD     #(0-IMM),Rd  
      XORC   #H'01,CCR

## 2.2.65 (2) SUB (W)

### SUB (SUBtract binary)

### Subtract Binary

#### Operation

Rd – (EAs) → Rd

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

#### Assembly-Language Format

SUB.W <EAs>, Rd

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

#### Operand Size

Word

#### Description

This instruction subtracts a source operand from the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States		
			1st byte		2nd byte		3rd byte			4th byte	
Immediate	SUB.W	#xx:16, Rd	7	9	3	rd	IMM				2
Register direct	SUB.W	Rs, Rd	1	9	rs	rd					1

#### Notes

## 2.2.65 (3) SUB (L)

**SUB (SUBtract binary)**

**Subtract Binary**

### Operation

$ERd - (EAs) \rightarrow ERd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↑	—	↑	↑	↑	↑

### Assembly-Language Format

SUB.L <EAs>, ERd

### Operand Size

Longword

- H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

### Description

This instruction subtracts a source operand from the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	SUB.L	#xx:32, ERd	7 : A	3 : 0:erd	IMM				3
Register direct	SUB.L	ERs, ERd	1 : A	1:ers:0:erd					1

### Notes

## 2.2.66 SUBS

SUBS (SUBtract with Sign extension)

Subtract Binary Address Data

### Operation

Rd – 1 → ERd

Rd – 2 → ERd

Rd – 4 → ERd

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

SUBS #1, ERd

SUBS #2, ERd

SUBS #4, ERd

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### Operand Size

Longword

### Description

This instruction subtracts the immediate value 1, 2, or 4 from the contents of a 32-bit register ERd (destination operand). Unlike the SUB instruction, it does not affect the condition-code flags.

### Available Registers

ERd: ER0 to ER7

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SUBS	#1, ERd	1 B	0 0:erd			1
Register direct	SUBS	#2, ERd	1 B	8 0:erd			1
Register direct	SUBS	#4, ERd	1 B	9 0:erd			1

### Notes

## 2.2.67 SUBX

SUBX (SUBtract with eXtend carry)

Subtract with Borrow

### Operation

$Rd - (EAs) - C \rightarrow Rd$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

### Assembly-Language Format

SUBX <EAs>, Rd

- H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
- N: Set to 1 if the result is negative; otherwise cleared to 0.
- Z: Set to 1 if the result is zero; otherwise cleared to 0.
- V: Set to 1 if an overflow occurs; otherwise cleared to 0.
- C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

### Operand Size

Byte

### Description

This instruction subtracts the source operand and carry flag from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

### Available Registers

Rd: R0L to R7L, R0H to R7H

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	SUBX	#xx:8, Rd	B    rd	IMM			1
Register direct	SUBX	Rs, Rd	1    E	rs    rd			1

### Notes

## 2.2.68 TAS

### TAS (Test And Set)

### Test and Set

#### Operation

@ERd – 0 → set/clear CCR  
1 → (<bit 7> of @ERd)

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

TAS @ERd

#### Operand Size

Byte

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Description

This instruction tests a memory operand by comparing it with zero, and sets the condition-code register according to the result. Then it sets the most significant bit (bit 7) of the operand to 1.

#### Available Registers

ERd: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format							No. of States
			1st byte		2nd byte		3rd byte		4th byte	
Register indirect	TAS	@ERd	0	1	E	0	7	B	0 : erd : C	4

#### Notes



## 2.2.69 TRAPA

**TRAPA (TRAP Always)**

**Trap Unconditionally**

### Operation

- When EXR is invalid

PC → @-SP

CCR → @-SP

<Vector> → PC

- When EXR is valid

PC → @-SP

CCR → @-SP

EXR → @-SP

<Vector> → PC

### Condition Code

I	UI	H	U	N	Z	V	C
1	*	—	—	—	—	—	—

\* See instruction set table.

I: Always set to 1.

UI: See note.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### Assembly-Language Format

TRAPA #x:2

### Operand Size

—

### Description

This instruction pushes the program counter (PC) and condition-code register (CCR) onto the stack, then sets the I bit to 1. If the extended control register (EXR) is valid, EXR is also saved onto the stack, but bits I2 to I0 are not modified. Next execution branches to a new address given by the contents of the vector address corresponding to the specified vector number. The PC value pushed onto the stack is the starting address of the next instruction after the TRAPA instruction.

#x	Vector Address	
	Normal Mode	Advanced Mode
0	H'0010 to H'0011	H'000020 to H'000023
1	H'0012 to H'0013	H'000024 to H'000027
2	H'0014 to H'0015	H'000028 to H'00002B
3	H'0016 to H'0017	H'00002C to H'00002F

## 2.2.69 TRAPA

TRAPA (TRAP Always)

Trap Unconditionally

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	TRAPA	#x:2	5    7	00:IMM: 0			7*

\*Eight states when EXR is valid.

### Notes

The UI bit is set to 1 when used as an interrupt mask bit, but retains its previous value when used as a user bit. For details, see the relevant microcontroller hardware manual.

The stack and vector structure differ between normal mode and advanced mode, and depending on whether EXR is valid or invalid.

## 2.2.70 (1) XOR (B)

### XOR (eXclusive OR logical)

### Exclusive Logical OR

#### Operation

$Rd \oplus (EAs) \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

#### Assembly-Language Format

XOR.B <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Byte

#### Description

This instruction exclusively ORs the source operand with the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

#### Available Registers

Rd: R0L to R7L, R0H to R7H

Rs: R0L to R7L, R0H to R7H

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	XOR.B	#xx:8, Rd	D	rd	IMM		1
Register direct	XOR.B	Rs, Rd	1	5	rs	rd	1

#### Notes

## 2.2.70 (2) XOR (W)

### XOR (eXclusive OR logical)

### Exclusive Logical OR

#### Operation

$Rd \oplus (EAs) \rightarrow Rd$

#### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

#### Assembly-Language Format

XOR.W <EAs>, Rd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

#### Operand Size

Word

#### Description

This instruction exclusively ORs the source operand with the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

#### Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States		
			1st byte		2nd byte		3rd byte			4th byte	
Immediate	XOR.W	#xx:16, Rd	7	9	5	rd	IMM				2
Register direct	XOR.W	Rs, Rd	6	5	rs	rd					1

#### Notes

### 2.2.70 (3) XOR (L)

#### XOR (eXclusive OR logical)

#### Exclusive Logical OR

##### Operation

$ERd \oplus (EAs) \rightarrow ERd$

##### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↑	↑	0	—

##### Assembly-Language Format

XOR.L <EAs>, ERd

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Previous value remains unchanged.

##### Operand Size

Longword

#### Description

This instruction exclusively ORs the source operand with the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

#### Available Registers

ERd: ER0 to ER7

ERs: ER0 to ER7

#### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States				
			1st byte		2nd byte		3rd byte		4th byte			5th byte		6th byte	
Immediate	XOR.L	#xx:32, ERd	7	A	5	0:erd	IMM								3
Register direct	XOR.L	ERs, ERd	0	1	F	0	6	5	0:ers	0:erd					2

#### Notes

## 2.2.71 (1) XORC

**XORC (eXclusive OR Control register)**

**Exclusive Logical OR with CCR**

### Operation

$\text{CCR} \oplus \text{\#IMM} \rightarrow \text{CCR}$

### Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

### Assembly-Language Format

XORC #xx:8, CCR

I: Stores the corresponding bit of the result.

UI: Stores the corresponding bit of the result.

H: Stores the corresponding bit of the result.

U: Stores the corresponding bit of the result.

N: Stores the corresponding bit of the result.

Z: Stores the corresponding bit of the result.

V: Stores the corresponding bit of the result.

C: Stores the corresponding bit of the result.

### Operand Size

Byte

### Description

This instruction exclusively ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	XORC	#xx:8, CCR	0 : 5	IMM			1

### Notes

## 2.2.71 (2) XORC

**XORC (eXclusive OR Control register)**

**Exclusive Logical OR with EXR**

### Operation

$\text{EXR} \oplus \# \text{IMM} \rightarrow \text{EXR}$

### Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

### Assembly-Language Format

XORC #xx:8, EXR

### Operand Size

Byte

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

### Description

This instruction exclusively ORs the contents of the extended control register (EXR) with immediate data and stores the result in the extended control register. No interrupt requests, including NMI, are accepted for three states after execution of this instruction.

### Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	XORC	#xx:8, EXR	0 : 1	4 : 1	0 : 5	IMM	2

### Notes

## 2.3 Instruction Set Summary

### 2.3.1 Instructions and Addressing Modes

Table 2-1 Instruction Set Summary

Function	Instruction	Addressing Mode													
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:32,ERn)	@-ERn/@ERn+	@aa:8	@aa:16	@aa:24	@aa:32	@(d:8,PC)	@(d:16,PC)	@@aa:8	
Data transfer	MOV	BWL	BWL	BWL	BWL	BWL	BWL	B	BWL	—	BWL	—	—	—	—
	POP, PUSH	—	—	—	—	—	—	—	—	—	—	—	—	—	WL
	LDM, STM	—	—	—	—	—	—	—	—	—	—	—	—	—	L
	MOVEPE, MOVTPE	—	—	—	—	—	—	—	B	—	—	—	—	—	—
Arithmetic operations	ADD, CMP	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	SUB	WL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	ADDX, SUBX	B	B	—	—	—	—	—	—	—	—	—	—	—	—
	ADDS, SUBS	—	L	—	—	—	—	—	—	—	—	—	—	—	—
	INC, DEC	WL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	DAA, DAS	—	B	—	—	—	—	—	—	—	—	—	—	—	—
	MULXU, DIVXU, MULXS, DIVXS	—	BW	—	—	—	—	—	—	—	—	—	—	—	—
	NEG	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	EXTU, EXTS	—	WL	—	—	—	—	—	—	—	—	—	—	—	—
	TAS	—	—	B	—	—	—	—	—	—	—	—	—	—	—
	MAC*	—	—	—	—	—	○	—	—	—	—	—	—	—	—
	CLRMAC*	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDMAC*, STMAC*	—	L	—	—	—	—	—	—	—	—	—	—	—	—

Note: \*These instructions are supported only by the H8S/2600 CPU.



**Table 2-1 Instruction Set Summary (cont)**

Function	Instruction	Addressing Mode													
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:32,ERn)	@-ERn/@ERn+	@aa:8	@aa:16	@aa:24	@aa:32	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Logic operations	AND, OR, XOR	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—	—
	NOT	—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Shift operations		—	BWL	—	—	—	—	—	—	—	—	—	—	—	—
Bit manipulation		—	B	B	—	—	—	B	B	—	B	—	—	—	—
Branch	Bcc, BSR	—	—	—	—	—	—	—	—	—	—	○	○	—	—
	JMP, JSR	—	—	—	—	—	—	—	—	○	—	—	—	○	—
	RTS	—	—	—	—	—	—	—	—	—	—	—	—	—	○
System control	TRAPA	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	RTE	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	SLEEP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDC	B	B	W	W	W	W	—	W	—	W	—	—	—	—
	STC	—	B	W	W	W	W	—	W	—	W	—	—	—	—
	ANDC, ORC, XORC	B	—	—	—	—	—	—	—	—	—	—	—	—	—
	NOP	—	—	—	—	—	—	—	—	—	—	—	—	—	○
Block data transfer		—	—	—	—	—	—	—	—	—	—	—	—	—	BW

**Legend**

B: Byte

W: Word

L: Longword

### 2.3.2 Instruction Set

**Table 2-2 Instruction Set**

**(1) Data Transfer Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code					No. of States		
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advance
MOV	MOV.B #xx:8,Rd	B	2									#xx:8→Rd8	—	—	↑	↑	0	—	1	
	MOV.B Rs,Rd	B		2								Rs8→Rd8	—	—	↑	↑	0	—	1	
	MOV.B @ERs,Rd	B			2							@ERs→Rd8	—	—	↑	↑	0	—	2	
	MOV.B @(d:16, ERs), Rd	B				4						@(d:16,ERs)→Rd8	—	—	↑	↑	0	—	3	
	MOV.B @(d:32,ERs),Rd	B				8						@(d:32,ERs)→Rd8	—	—	↑	↑	0	—	5	
	MOV.B @ERs+,Rd	B					2					@ERs→Rd8,ERs32+1→ERs32	—	—	↑	↑	0	—	3	
	MOV.B @aa:8,Rd	B						2				@aa:8→Rd8	—	—	↑	↑	0	—	2	
	MOV.B @aa:16,Rd	B						4				@aa:16→Rd8	—	—	↑	↑	0	—	3	
	MOV.B @aa:32,Rd	B						6				@aa:32→Rd8	—	—	↑	↑	0	—	4	
	MOV.B Rs, @ERd	B			2							Rs8→@ERd	—	—	↑	↑	0	—	2	
	MOV.B Rs, @(d:16,ERd)	B				4						Rd8→@(d:16,ERd)	—	—	↑	↑	0	—	3	
	MOV.B Rs, @(d:32,ERd)	B				8						Rd8→@(d:32,ERd)	—	—	↑	↑	0	—	5	
	MOV.B Rs, @-ERd	B					2					ERd32-1→ERd32,Rs8→@ERd	—	—	↑	↑	0	—	3	
	MOV.B Rs, @aa:8	B						2				Rs8→@aa:8	—	—	↑	↑	0	—	2	
	MOV.B Rs, @aa:16	B						4				Rs8→@aa:16	—	—	↑	↑	0	—	3	
	MOV.B Rs, @aa:32	B						6				Rs8→@aa:32	—	—	↑	↑	0	—	4	
	MOV.W #xx:16,Rd	W	4									#xx:16→Rd16	—	—	↑	↑	0	—	2	
	MOV.W Rs,Rd	W		2								Rs16→Rd16	—	—	↑	↑	0	—	1	
	MOV.W @ERs,Rd	W			2							@ERs→Rd16	—	—	↑	↑	0	—	2	
	MOV.W @(d:16,ERs),Rd	W				4						@(d:16,ERs)→Rd16	—	—	↑	↑	0	—	3	
	MOV.W @(d:32,ERs),Rd	W				8						@(d:32,ERs)→Rd16	—	—	↑	↑	0	—	5	
	MOV.W @ERs+,Rd	W					2					ERs→Rd16,ERs32+2→@ERd32	—	—	↑	↑	0	—	3	
	MOV.W @aa:16,Rd	W						4				@aa:16→Rd16	—	—	↑	↑	0	—	3	
	MOV.W @aa:32,Rd	W						6				@aa:32→Rd16	—	—	↑	↑	0	—	4	
	MOV.W Rs, @ERd	W			2							Rs16→@ERd	—	—	↑	↑	0	—	2	
	MOV.W Rs, @(d:16,ERd)	W				4						Rs16→@(d:16,ERd)	—	—	↑	↑	0	—	3	
	MOV.W Rs, @(d:32,ERd)	W				8						Rs16→@(d:32,ERd)	—	—	↑	↑	0	—	5	
	MOV.W Rs, @-ERd	W					2					ERd32-2→ERd32,Rs16→@ERd	—	—	↑	↑	0	—	3	
	MOV.W Rs, @aa:16	W						4				Rs16→@aa:16	—	—	↑	↑	0	—	3	
	MOV.W Rs, @aa:32	W						6				Rs16→@aa:32	—	—	↑	↑	0	—	4	

**Table 2-2 Instruction Set (cont)**

**(1) Data Transfer Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)								Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@-ERn/@ERn+	@aa	@(d,PC)	@@aa		I	H	N	Z	V	C	Normal	Advanced
MOV	MOV.L #xx:32,Rd	L	6								#xx:32→Rd32	—	—	↑	↑	0	—	3	
	MOV.L ERs,ERd	L		2							ERs32→ERd32	—	—	↑	↑	0	—	1	
	MOV.L @ERs,ERd	L			4						@ERs→ERd32	—	—	↑	↑	0	—	4	
	MOV.L @(d:16,ERs),ERd	L				6					@(d:16,ERs)→ERd32	—	—	↑	↑	0	—	5	
	MOV.L @(d:32,ERs),ERd	L				10					@(d:32,ERs)→ERd32	—	—	↑	↑	0	—	7	
	MOV.L @ERs+,ERd	L					4				@ERs→ERd32,ERs32+4→@ERs32	—	—	↑	↑	0	—	5	
	MOV.L @aa:16,ERd	L						6			@aa:16→ERd32	—	—	↑	↑	0	—	5	
	MOV.L @aa:32,ERd	L						8			@aa:32→ERd32	—	—	↑	↑	0	—	6	
	MOV.L ERs,@ERd	L			4						ERs32→@ERd	—	—	↑	↑	0	—	4	
	MOV.L ERs,@(d:16,ERd)	L				6					ERs32→@(d:16,ERd)	—	—	↑	↑	0	—	5	
	MOV.L ERs,@(d:32,ERd)	L				10					ERs32→@(d:32,ERd)	—	—	↑	↑	0	—	7	
	MOV.L ERs,@-ERd	L					4				ERd32-4→ERd32,ERs32→@ERd	—	—	↑	↑	0	—	5	
	MOV.L ERs,@aa:16	L						6			ERs32→@aa:16	—	—	↑	↑	0	—	5	
	MOV.L ERs,@aa:32	L						8			ERs32→@aa:32	—	—	↑	↑	0	—	6	
POP	POP.W Rn	W								2	@SP→Rn16,SP+2→SP	—	—	↑	↑	0	—	3	
	POP.L ERn	L								4	@SP→ERn32,SP+4→SP	—	—	↑	↑	0	—	5	
PUSH	PUSH.W Rn	W								2	SP-2→SP,Rn16→@SP	—	—	↑	↑	0	—	3	
	PUSH.L ERn	L								4	SP-4→SP,ERn32→@SP	—	—	↑	↑	0	—	5	
LDM	LDM.L @SP+,(ERm-ERn)	L								4	(@SP→ERn32,SP+4→SP) repeat until end of register list	—	—	—	—	—	—	7/9/11 ①	
STM	STM.L (ERm-ERn),@-SP	L								4	(SP-4→SP,ERn32→@SP) repeat until end of register list	—	—	—	—	—	—	7/9/11 ①	
MOVFP	MOVFP @aa:16,Rd	B						4			@aa:16→Rd (synchronized with E clock)	—	—	↑	↑	0	—	②	
MOVTP	MOVTP Rs,@aa:16	B						4			Rs→@aa:16 (synchronized with E clock)	—	—	↑	↑	0	—	②	

Table 2-2 Instruction Set (cont)

## (2) Arithmetic Operation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)										Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa		I		H	N	Z	V	C	Normal		
ADD	ADD.B #xx:8,Rd	B	2									Rd8+#xx:8→Rd8	—	↑	↑	↑	↑	↑		1	
	ADD.B Rs,Rd	B		2								Rd8+Rs8→Rd8	—	↑	↑	↑	↑	↑		1	
	ADD.W #xx:16,Rd	W	4									Rd16+#xx:16→Rd16	—	③	↑	↑	↑	↑		2	
	ADD.W Rs,Rd	W		2								Rd16+Rs16→Rd16	—	③	↑	↑	↑	↑		1	
	ADD.L #xx:32,ERd	L	6									ERd32+#xx:32→ERd32	—	④	↑	↑	↑	↑		3	
	ADD.L ERs,ERd	L		2								ERd32+ERs32→ERd32	—	④	↑	↑	↑	↑		1	
ADDX	ADDX #xx:8,Rd	B	2									Rd8+#xx:8+C→Rd8	—	↑	↑	⑤	↑	↑		1	
	ADDX Rs,Rd	B		2								Rd8+Rs8+C→Rd8	—	↑	↑	⑤	↑	↑		1	
ADDS	ADDS #1,ERd	L		2								ERd32+1→ERd32	—	—	—	—	—	—		1	
	ADDS #2,ERd	L		2								ERd32+2→ERd32	—	—	—	—	—	—		1	
	ADDS #4,ERd	L		2								ERd32+4→ERd32	—	—	—	—	—	—		1	
INC	INC.B Rd	B		2								Rd8+1→Rd8	—	—	↑	↑	↑	↑		1	
	INC.W #1,Rd	W		2								Rd16+1→Rd16	—	—	↑	↑	↑	↑		1	
	INC.W #2,Rd	W		2								Rd16+2→Rd16	—	—	↑	↑	↑	↑		1	
	INC.L #1,ERd	L		2								ERd32+1→ERd32	—	—	↑	↑	↑	↑		1	
	INC.L #2,ERd	L		2								ERd32+2→ERd32	—	—	↑	↑	↑	↑		1	
DAA	DAA Rd	B		2								Rd8 decimal adjust → Rd8	—	*	↑	↑	*	—		1	
SUB	SUB.B Rs,Rd	B		2								Rd8-Rs8→Rd8	—	↑	↑	↑	↑	↑		1	
	SUB.W #xx:16,Rd	W	4									Rd16-#xx:16→Rd16	—	③	↑	↑	↑	↑		2	
	SUB.W Rs,Rd	W		2								Rd16-Rs16→Rd16	—	③	↑	↑	↑	↑		1	
	SUB.L #xx:32,ERd	L	6									ERd32-#xx:32→ERd32	—	④	↑	↑	↑	↑		3	
	SUB.L ERs,ERd	L		2								ERd32-ERs32→ERd32	—	④	↑	↑	↑	↑		1	
SUBX	SUBX #xx:8,Rd	B	2									Rd8-#xx:8-C→Rd8	—	↑	↑	⑤	↑	↑		1	
	SUBX Rs,Rd	B		2								Rd8-Rs8-C→Rd8	—	↑	↑	⑤	↑	↑		1	
SUBS	SUBS #1,ERd	L		2								ERd32-1→ERd32	—	—	—	—	—	—		1	
	SUBS #2,ERd	L		2								ERd32-2→ERd32	—	—	—	—	—	—		1	
	SUBS #4,ERd	L		2								ERd32-4→ERd32	—	—	—	—	—	—		1	
DEC	DEC.B Rd	B		2								Rd8-1→Rd8	—	—	↑	↑	↑	↑		1	
	DEC.W #1,Rd	W		2								Rd16-1→Rd16	—	—	↑	↑	↑	↑		1	
	DEC.W #2,Rd	W		2								Rd16-2→Rd16	—	—	↑	↑	↑	↑		1	
	DEC.L #1,ERd	L		2								ERd32-1→ERd32	—	—	↑	↑	↑	↑		1	
	DEC.L #2,ERd	L		2								ERd32-2→ERd32	—	—	↑	↑	↑	↑		1	

Table 2-2 Instruction Set (cont)

## (2) Arithmetic Operation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
DAS	DAS Rd	B		2								Rd8 decimal adjust →Rd8	—	*	↑	↑	*	—		1
MULXU	MULXU.B Rs,Rd	B		2								Rd8×Rs8→Rd16 (unsigned operation)	—	—	—	—	—	—		3 (12*)
	MULXU.W Rs,ERd	W		2								Rd16×Rs16→ERd32 (unsigned operation)	—	—	—	—	—	—		4 (20*)
MULXS	MULXS.B Rs,Rd	B		4								Rd8×Rs8→Rd16 (signed operation)	—	—	↑	↑	—	—		4 (13*)
	MULXS.W Rs,ERd	W		4								Rd16×Rs16→ERd32 (signed multiplication)	—	—	↑	↑	—	—		5 (21*)
DIVXU	DIVXU.B Rs,Rd	B		2								Rd16÷Rs8→Rd16 (RdH: remainder, RdL: quotient) (unsigned division)	—	—	⑥	⑦	—	—		12
	DIVXU.W Rs,ERd	W		2								ERd32÷Rs16→ERd32 (Ed: remainder, Rd: quotient) (unsigned division)	—	—	⑥	⑦	—	—		20
DIVXS	DIVXS.B Rs,Rd	B		4								Rd16÷Rs8→Rd16 (RdH: remainder, RdL: quotient) (signed division)	—	—	⑧	⑦	—	—		13
	DIVXS.W Rs,ERd	W		4								ERd32÷Rs16→ERd32 (Ed: remainder, Rd: quotient) (signed division)	—	—	⑧	⑦	—	—		21
CMP	CMP.B #xx:8,Rd	B	2									Rd8-#xx:8	—	↑	↑	↑	↑	↑		1
	CMP.B Rs,Rd	B		2								Rd8-Rs8	—	↑	↑	↑	↑	↑		1
	CMP.W #xx:16,Rd	W	4									Rd16-#xx:16	—	③	↑	↑	↑	↑		2
	CMP.W Rs,Rd	W		2								Rd16-Rs16	—	③	↑	↑	↑	↑		1
	CMP.L #xx:32,ERd	L	6									ERd32-#xx:32	—	④	↑	↑	↑	↑		3
	CMP.L ERs,ERd	L		2								ERd32-ERs32	—	④	↑	↑	↑	↑		1
NEG	NEG.B Rd	B		2								0-Rd8→Rd8	—	↑	↑	↑	↑	↑		1
	NEG.W Rd	W		2								0-Rd16→Rd16	—	↑	↑	↑	↑	↑		1
	NEG.L ERd	L		2								0-ERd32→ERd32	—	↑	↑	↑	↑	↑		1
EXTU	EXTU.W Rd	W		2								0→(<bits 15 to 8> of Rd16)	—	—	0	↑	0	—		1
	EXTU.L ERd	L		2								0→(<bits 31 to 16> of ERd32)	—	—	0	↑	0	—		1
EXTS	EXTS.W Rd	W		2								(<bit 7> of Rd16)→(<bits 15 to 8> of Rd16)	—	—	↑	↑	0	—		1
	EXTS.L ERd	L		2								(<bit 15> of ERd32)→(<bits 31 to 16> of ERd32)	—	—	↑	↑	0	—		1
TAS	TAS @ERd	B			4							@ERd-0→set CCR, 1→(<bit 7> of @ERd)	—	—	↑	↑	0	—		4

Note: \*For the H8S/2000 CPU.

**Table 2-2 Instruction Set (cont)**

**(2) Arithmetic Operation Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
MAC*	MAC @ERn+,@ERm+	—				4						@ERn×@ERm+MAC→MAC (signed multiplication) ERn+2→ERn,ERm+2→ERm	—	—	— ⑨	— ⑨	— ⑨	—		4
CLRMAC*	CLRMAC	—								2		0→MACH, MACL	—	—	—	—	—	—		2
LDMAC*	LDMAC ERs,MACH	L		2								ERs→MACH	—	—	—	—	—	—		2
	LDMAC ERs,MACL	L		2								ERs→MACL	—	—	—	—	—	—		2
STMAC*	STMAC MACH,ERd	L		2								MACH→ERd	—	—	↑	↑	↑	—		1
	STMAC MACL,ERd	L		2								MACL→ERd	—	—	↑	↑	↑	—		1

Note: \*These instructions are supported only by the H8S/2600 CPU.

**Table 2-2 Instruction Set (cont)**

**(3) Logic Operation Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@-ERn/@ERn+	@aa	@(d,PC)	@@aa			I	H	N	Z	V	C	Normal	Advanced
AND	AND.B #xx:8,Rd	B	2									Rd8^#xx:8→Rd8	—	—	↑	↑	0	—	1	
	AND.B Rs,Rd	B		2								Rd8^Rs8→Rd8	—	—	↑	↑	0	—	1	
	AND.W #xx:16,Rd	W	4									Rd16^#xx:16→Rd16	—	—	↑	↑	0	—	2	
	AND.W Rs,Rd	W		2								Rd16^Rs16→Rd16	—	—	↑	↑	0	—	1	
	AND.L #xx:32,ERd	L	6									ERd32^#xx:32→ERd32	—	—	↑	↑	0	—	3	
	AND.L ERs,ERd	L		4								ERd32^ERs32→ERd32	—	—	↑	↑	0	—	2	
OR	OR.B #xx:8,Rd	B	2									Rd8v#xx:8→Rd8	—	—	↑	↑	0	—	1	
	OR.B Rs,Rd	B		2								Rd8vRs8→Rd8	—	—	↑	↑	0	—	1	
	OR.W #xx:16,Rd	W	4									Rd16v#xx:16→Rd16	—	—	↑	↑	0	—	2	
	OR.W Rs,Rd	W		2								Rd16vRs16→Rd16	—	—	↑	↑	0	—	1	
	OR.L #xx:32,ERd	L	6									ERd32v#xx:32→ERd32	—	—	↑	↑	0	—	3	
	OR.L ERs,ERd	L		4								ERd32vERs32→ERd32	—	—	↑	↑	0	—	2	
XOR	XOR.B #xx:8,Rd	B	2									Rd8@#xx:8→Rd8	—	—	↑	↑	0	—	1	
	XOR.B Rs,Rd	B		2								Rd8@Rs8→Rd8	—	—	↑	↑	0	—	1	
	XOR.W #xx:16,Rd	W	4									Rd16@#xx:16→Rd16	—	—	↑	↑	0	—	2	
	XOR.W Rs,Rd	W		2								Rd16@Rs16→Rd16	—	—	↑	↑	0	—	1	
	XOR.L #xx:32,ERd	L	6									ERd32@#xx:32→ERd32	—	—	↑	↑	0	—	3	
	XOR.L ERs,ERd	L		4								ERd32@ERs32→ERd32	—	—	↑	↑	0	—	2	
NOT	NOT.B Rd	B		2								¬ Rd8→Rd8	—	—	↑	↑	0	—	1	
	NOT.W Rd	W		2								¬ Rd16→Rd16	—	—	↑	↑	0	—	1	
	NOT.L ERd	L		2								¬ Rd32→Rd32	—	—	↑	↑	0	—	1	

**Table 2-2 Instruction Set (cont)**

**(4) Shift Instructions**

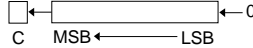
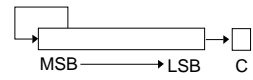
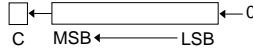
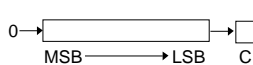
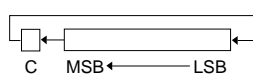
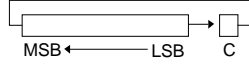
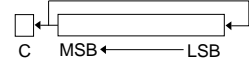
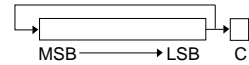
Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States		
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa	I		I	H	N	Z	V	C			Normal
SHAL	SHAL.B Rd	B		2									—	—	↑	↑	↑	↑	↑	1	
	SHAL.B #2,Rd	B		2									—	—	↑	↑	↑	↑	↑	1	
	SHAL.W Rd	W		2									—	—	↑	↑	↑	↑	↑	1	
	SHAL.W #2,Rd	W		2									—	—	↑	↑	↑	↑	↑	1	
	SHAL.L ERd	L		2									—	—	↑	↑	↑	↑	↑	1	
	SHAL.L #2,ERd	L		2									—	—	↑	↑	↑	↑	↑	1	
SHAR	SHAR.B Rd	B		2									—	—	↑	↑	0	↑	1		
	SHAR.B #2,Rd	B		2									—	—	↑	↑	0	↑	1		
	SHAR.W Rd	W		2									—	—	↑	↑	0	↑	1		
	SHAR.W #2,Rd	W		2									—	—	↑	↑	0	↑	1		
	SHAR.L ERd	L		2									—	—	↑	↑	0	↑	1		
	SHAR.L #2,ERd	L		2									—	—	↑	↑	0	↑	1		
SHLL	SHLL.B Rd	B		2									—	—	↑	↑	0	↑	1		
	SHLL.B #2,Rd	B		2									—	—	↑	↑	0	↑	1		
	SHLL.W Rd	W		2									—	—	↑	↑	0	↑	1		
	SHLL.W #2,Rd	W		2									—	—	↑	↑	0	↑	1		
	SHLL.L ERd	L		2									—	—	↑	↑	0	↑	1		
	SHLL.L #2,ERd	L		2									—	—	↑	↑	0	↑	1		
SHLR	SHLR.B Rd	B		2									—	—	0	↑	0	↑	1		
	SHLR.B #2,Rd	B		2									—	—	0	↑	0	↑	1		
	SHLR.W Rd	W		2									—	—	0	↑	0	↑	1		
	SHLR.W #2,Rd	W		2									—	—	0	↑	0	↑	1		
	SHLR.L ERd	L		2									—	—	0	↑	0	↑	1		
	SHLR.L #2,ERd	L		2									—	—	0	↑	0	↑	1		
ROTXL	ROTXL.B Rd	B		2									—	—	↑	↑	0	↑	1		
	ROTXL.B #2,Rd	B		2									—	—	↑	↑	0	↑	1		
	ROTXL.W Rd	W		2									—	—	↑	↑	0	↑	1		
	ROTXL.W #2,Rd	W		2									—	—	↑	↑	0	↑	1		
	ROTXL.L ERd	L		2									—	—	↑	↑	0	↑	1		
	ROTXL.L #2,ERd	L		2									—	—	↑	↑	0	↑	1		



Table 2-2 Instruction Set (cont)

(4) Shift Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn++	@aa	@ (d,PC)	@ @aa	I		I	H	N	Z	V	C	Normal	Advanced
ROTXR	ROTXR.B Rd	B		2									—	—	↑	↑	0	↑		1
	ROTXR.B #2,Rd	B		2									—	—	↑	↑	0	↑		1
	ROTXR.W Rd	W		2									—	—	↑	↑	0	↑		1
	ROTXR.W #2,Rd	W		2									—	—	↑	↑	0	↑		1
	ROTXR.L ERd	L		2									—	—	↑	↑	0	↑		1
	ROTXR.L #2,ERd	L		2									—	—	↑	↑	0	↑		1
ROTL	ROTL.B Rd	B		2									—	—	↑	↑	0	↑		1
	ROTL.B #2,Rd	B		2									—	—	↑	↑	0	↑		1
	ROTL.W Rd	W		2									—	—	↑	↑	0	↑		1
	ROTL.W #2,Rd	W		2									—	—	↑	↑	0	↑		1
	ROTL.L ERd	L		2									—	—	↑	↑	0	↑		1
	ROTL.L #2,ERd	L		2									—	—	↑	↑	0	↑		1
ROTR	ROTR.B Rd	B		2									—	—	↑	↑	0	↑		1
	ROTR.B #2,Rd	B		2									—	—	↑	↑	0	↑		1
	ROTR.W Rd	W		2									—	—	↑	↑	0	↑		1
	ROTR.W #2,Rd	W		2									—	—	↑	↑	0	↑		1
	ROTR.L ERd	L		2									—	—	↑	↑	0	↑		1
	ROTR.L #2,ERd	L		2									—	—	↑	↑	0	↑		1

**Table 2-2 Instruction Set (cont)**

**(5) Bit Manipulation Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa	I		I	H	N	Z	V	C	Normal	Advanced
BSET	BSET #xx:3,Rd	B		2								(#xx:3 of Rd8)←1	—	—	—	—	—	—	1	
	BSET #xx:3,@ERd	B			4							(#xx:3 of @ERd)←1	—	—	—	—	—	—	4	
	BSET #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)←1	—	—	—	—	—	—	4	
	BSET #xx:3,@aa:16	B						6				(#xx:3 of @aa:16)←1	—	—	—	—	—	—	5	
	BSET #xx:3,@aa:32	B						8				(#xx:3 of @aa:32)←1	—	—	—	—	—	—	6	
	BSET Rn,Rd	B		2								(Rn8 of Rd8)←1	—	—	—	—	—	—	1	
	BSET Rn,@ERd	B			4							(Rn8 of @ERd)←1	—	—	—	—	—	—	4	
	BSET Rn,@aa:8	B						4				(Rn8 of @aa:8)←1	—	—	—	—	—	—	4	
BSET Rn,@aa:16	B						6				(Rn8 of @aa:16)←1	—	—	—	—	—	—	5		
BSET Rn,@aa:32	B						8				(Rn8 of @aa:32)←1	—	—	—	—	—	—	6		
BCLR	BCLR #xx:3,Rd	B		2								(#xx:3 of Rd8)←0	—	—	—	—	—	—	1	
	BCLR #xx:3,@ERd	B			4							(#xx:3 of @ERd)←0	—	—	—	—	—	—	4	
	BCLR #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)←0	—	—	—	—	—	—	4	
	BCLR #xx:3,@aa:16	B						6				(#xx:3 of @aa:16)←0	—	—	—	—	—	—	5	
	BCLR #xx:3,@aa:32	B						8				(#xx:3 of @aa:32)←0	—	—	—	—	—	—	6	
	BCLR Rn,Rd	B		2								(Rn8 of Rd8)←0	—	—	—	—	—	—	1	
	BCLR Rn,@ERd	B			4							(Rn8 of @ERd)←0	—	—	—	—	—	—	4	
	BCLR Rn,@aa:8	B						4				(Rn8 of @aa:8)←0	—	—	—	—	—	—	4	
BCLR Rn,@aa:16	B						6				(Rn8 of @aa:16)←0	—	—	—	—	—	—	5		
BCLR Rn,@aa:32	B						8				(Rn8 of @aa:32)←0	—	—	—	—	—	—	6		
BNOT	BNOT #xx:3,Rd	B		2								(#xx:3 of Rd8)← [¬ (#xx:3 of Rd8)]	—	—	—	—	—	—	1	
	BNOT #xx:3,@ERd	B			4							(#xx:3 of @ERd)← [¬ (#xx:3 of @ERd)]	—	—	—	—	—	—	4	
	BNOT #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)← [¬ (#xx:3 of @aa:8)]	—	—	—	—	—	—	4	
	BNOT #xx:3,@aa:16	B						6				(#xx:3 of @aa:16)← [¬ (#xx:3 of @aa:16)]	—	—	—	—	—	—	5	
	BNOT #xx:3,@aa:32	B						8				(#xx:3 of @aa:32)← [¬ (#xx:3 of @aa:32)]	—	—	—	—	—	—	6	
	BNOT Rn,Rd	B		2								(Rn8 of Rd8)← [¬ (Rn8 of Rd8)]	—	—	—	—	—	—	1	
	BNOT Rn,@ERd	B			4							(Rn8 of @ERd)← [¬ (Rn8 of @ERd)]	—	—	—	—	—	—	4	
	BNOT Rn,@aa:8	B						4				(Rn8 of @aa:8)← [¬ (Rn8 of @aa:8)]	—	—	—	—	—	—	4	
BNOT Rn,@aa:16	B						6				(Rn8 of @aa:16)← [¬ (Rn8 of @aa:16)]	—	—	—	—	—	—	5		
BNOT Rn,@aa:32	B						8				(Rn8 of @aa:32)← [¬ (Rn8 of @aa:32)]	—	—	—	—	—	—	6		

Table 2-2 Instruction Set (cont)

## (5) Bit Manipulation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code					No. of States		
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn++	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
BTST	BTST #xx:3,Rd	B		2								(#xx:3 of Rd8)→Z	—	—	—	↑	—	—	1	
	BTST #xx:3,@ERd	B				4						(#xx:3 of @ERd)→Z	—	—	—	↑	—	—	3	
	BTST #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)→Z	—	—	—	↑	—	—	3	
	BTST #xx:3,@aa:16	B						6				(#xx:3 of @aa:16)→Z	—	—	—	↑	—	—	4	
	BTST #xx:3,@aa:32	B						8				(#xx:3 of @aa:32)→Z	—	—	—	↑	—	—	5	
	BTST Rn,Rd	B		2								(Rn8 of Rd8)→Z	—	—	—	↑	—	—	1	
	BTST Rn,@ERd	B			4							(Rn8 of @ERd)→Z	—	—	—	↑	—	—	3	
	BTST Rn,@aa:8	B						4				(Rn8 of @aa:8)→Z	—	—	—	↑	—	—	3	
BTST Rn,@aa:16	B							6			(Rn8 of @aa:16)→Z	—	—	—	↑	—	—	4		
BTST Rn,@aa:32	B							8			(Rn8 of @aa:32)→Z	—	—	—	↑	—	—	5		
BLD	BLD #xx:3,Rd	B		2								(#xx:3 of Rd8)→C	—	—	—	—	↑	—	1	
	BLD #xx:3,@ERd	B			4							(#xx:3 of @ERd)→C	—	—	—	—	↑	—	3	
	BLD #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)→C	—	—	—	—	↑	—	3	
	BLD #xx:3,@aa:16	B						6				(#xx:3 of @aa:16)→C	—	—	—	—	↑	—	4	
	BLD #xx:3,@aa:32	B						8				(#xx:3 of @aa:32)→C	—	—	—	—	↑	—	5	
BILD	BILD #xx:3,Rd	B		2								¬ (#xx:3 of Rd8)→C	—	—	—	—	↑	—	1	
	BILD #xx:3,@ERd	B			4							¬ (#xx:3 of @ERd)→C	—	—	—	—	↑	—	3	
	BILD #xx:3,@aa:8	B						4				¬ (#xx:3 of @aa:8)→C	—	—	—	—	↑	—	3	
	BILD #xx:3,@aa:16	B						6				¬ (#xx:3 of @aa:16)→C	—	—	—	—	↑	—	4	
	BILD #xx:3,@aa:32	B						8				¬ (#xx:3 of @aa:32)→C	—	—	—	—	↑	—	5	
BST	BST #xx:3,Rd	B		2								C→(#xx:3 of Rd8)	—	—	—	—	—	—	1	
	BST #xx:3,@ERd	B			4							C→(#xx:3 of @ERd24)	—	—	—	—	—	—	4	
	BST #xx:3,@aa:8	B						4				C→(#xx:3 of @aa:8)	—	—	—	—	—	—	4	
	BST #xx:3,@aa:16	B						6				C→(#xx:3 of @aa:16)	—	—	—	—	—	—	5	
	BST #xx:3,@aa:32	B						8				C→(#xx:3 of @aa:32)	—	—	—	—	—	—	6	
BIST	BIST #xx:3,Rd	B		2								¬ C→(#xx:3 of Rd8)	—	—	—	—	—	—	1	
	BIST #xx:3,@ERd	B			4							¬ C→(#xx:3 of @ERd24)	—	—	—	—	—	—	4	
	BIST #xx:3,@aa:8	B						4				¬ C→(#xx:3 of @aa:8)	—	—	—	—	—	—	4	
	BIST #xx:3,@aa:16	B						6				¬ C→(#xx:3 of @aa:16)	—	—	—	—	—	—	5	
	BIST #xx:3,@aa:32	B						8				¬ C→(#xx:3 of @aa:32)	—	—	—	—	—	—	6	

Table 2-2 Instruction Set (cont)

## (5) Bit Manipulation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
BAND	BAND #xx:3,Rd	B		2								C^(#xx:3 of Rd8)→C	—	—	—	—	—	↑		1
	BAND #xx:3,@ERd	B			4							C^(#xx:3 of @ERd24)→C	—	—	—	—	—	↑		3
	BAND #xx:3,@aa:8	B						4				C^(#xx:3 of @aa:8)→C	—	—	—	—	—	↑		3
	BAND #xx:3,@aa:16	B						6				C^(#xx:3 of @aa:16)→C	—	—	—	—	—	↑		4
	BAND #xx:3,@aa:32	B						8				C^(#xx:3 of @aa:32)→C	—	—	—	—	—	↑		5
BIAND	BIAND #xx:3,Rd	B		2								C^ [¬ (#xx:3 of Rd8)]→C	—	—	—	—	—	↑		1
	BIAND #xx:3,@ERd	B			4							C^ [¬ (#xx:3 of @ERd24)]→C	—	—	—	—	—	↑		3
	BIAND #xx:3,@aa:8	B						4				C^ [¬ (#xx:3 of @aa:8)]→C	—	—	—	—	—	↑		3
	BIAND #xx:3,@aa:16	B						6				C^ [¬ (#xx:3 of @aa:16)]→C	—	—	—	—	—	↑		4
	BIAND #xx:3,@aa:32	B						8				C^ [¬ (#xx:3 of @aa:32)]→C	—	—	—	—	—	↑		5
BOR	BOR #xx:3,Rd	B		2								Cv(#xx:3 of Rd8)→C	—	—	—	—	—	↑		1
	BOR #xx:3,@ERd	B			4							Cv(#xx:3 of @ERd24)→C	—	—	—	—	—	↑		3
	BOR #xx:3,@aa:8	B						4				Cv(#xx3: of @aa:8)→C	—	—	—	—	—	↑		3
	BOR #xx:3,@aa:16	B						6				Cv(#xx3: of @aa:16)→C	—	—	—	—	—	↑		4
	BOR #xx:3,@aa:32	B						8				Cv(#xx3: of @aa:32)→C	—	—	—	—	—	↑		5
BIOR	BIOR #xx:3,Rd	B		2								Cv [¬ (#xx:3 of Rd8)]→C	—	—	—	—	—	↑		1
	BIOR #xx:3,@ERd	B			4							Cv [¬ (#xx:3 of @ERd24)]→C	—	—	—	—	—	↑		3
	BIOR #xx:3,@aa:8	B						4				Cv [¬ (#xx:3 of @aa:8)]→C	—	—	—	—	—	↑		3
	BIOR #xx:3,@aa:16	B						6				Cv [¬ (#xx:3 of @aa:16)]→C	—	—	—	—	—	↑		4
	BIOR #xx:3,@aa:32	B						8				Cv [¬ (#xx:3 of @aa:32)]→C	—	—	—	—	—	↑		5
BXOR	BXOR #xx:3,Rd	B		2								C^(#xx:3 of Rd8)→C	—	—	—	—	—	↑		1
	BXOR #xx:3,@ERd	B			4							C^(#xx:3 of @ERd24)→C	—	—	—	—	—	↑		3
	BXOR #xx:3,@aa:8	B						4				C^(#xx:3 of @aa:8)→C	—	—	—	—	—	↑		3
	BXOR #xx:3,@aa:16	B						6				C^(#xx:3 of @aa:16)→C	—	—	—	—	—	↑		4
	BXOR #xx:3,@aa:32	B						8				C^(#xx:3 of @aa:32)→C	—	—	—	—	—	↑		5
BIXOR	BIXOR #xx:3,Rd	B		2								C^( [¬ (#xx:3 of Rd8)]→C	—	—	—	—	—	↑		1
	BIXOR #xx:3,@ERd	B			4							C^( [¬ (#xx:3 of @ERd24)]→C	—	—	—	—	—	↑		3
	BIXOR #xx:3,@aa:8	B						4				C^( [¬ (#xx:3 of @aa:8)]→C	—	—	—	—	—	↑		3
	BIXOR #xx:3,@aa:16	B						6				C^( [¬ (#xx:3 of @aa:16)]→C	—	—	—	—	—	↑		4
	BIXOR #xx:3,@aa:32	B						8				C^( [¬ (#xx:3 of @aa:32)]→C	—	—	—	—	—	↑		5

**Table 2-2 Instruction Set (cont)**

**(6) Branch Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)								Operation	Condition Code						No. of States		
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
Bcc	BRA d:8(BT d:8)	—							2		if condition is true then PC←PC+d else next;	Always	—	—	—	—	—	—	2	
	BRA d:16(BT d:16)	—							4				—	—	—	—	—	—	3	
	BRN d:8(BF d:8)	—							2			Never	—	—	—	—	—	—	2	
	BRN d:16(BF d:16)	—							4				—	—	—	—	—	—	3	
	BHI d:8	—							2			Cvz=0	—	—	—	—	—	—	2	
	BHI d:16	—							4				—	—	—	—	—	—	3	
	BLS d:8	—							2			Cvz=1	—	—	—	—	—	—	2	
	BLS d:16	—							4				—	—	—	—	—	—	3	
	BCC d:8(BHS d:8)	—							2			C=0	—	—	—	—	—	—	2	
	BCC d:16(BHS d:16)	—							4				—	—	—	—	—	—	3	
	BCS d:8(BLO d:8)	—							2			C=1	—	—	—	—	—	—	2	
	BCS d:16(BLO d:16)	—							4				—	—	—	—	—	—	3	
	BNE d:8	—							2			Z=0	—	—	—	—	—	—	2	
	BNE d:16	—							4				—	—	—	—	—	—	3	
	BEQ d:8	—							2			Z=1	—	—	—	—	—	—	2	
	BEQ d:16	—							4				—	—	—	—	—	—	3	
	BVC d:8	—							2			V=0	—	—	—	—	—	—	2	
	BVC d:16	—							4				—	—	—	—	—	—	3	
	BVS d:8	—							2			V=1	—	—	—	—	—	—	2	
	BVS d:16	—							4				—	—	—	—	—	—	3	
	BPL d:8	—							2			N=0	—	—	—	—	—	—	2	
	BPL d:16	—							4				—	—	—	—	—	—	3	
	BMI d:8	—							2			N=1	—	—	—	—	—	—	2	
	BMI d:16	—							4				—	—	—	—	—	—	3	
	BGE d:8	—							2			N⊕V=0	—	—	—	—	—	—	2	
	BGE d:16	—							4				—	—	—	—	—	—	3	
	BLT d:8	—							2			N⊕V=1	—	—	—	—	—	—	2	
	BLT d:16	—							4				—	—	—	—	—	—	3	
	BGT d:8	—							2			Zv(N⊕V)=0	—	—	—	—	—	—	2	
	BGT d:16	—							4				—	—	—	—	—	—	3	
	BLE d:8	—							2			Zv(N⊕V)=1	—	—	—	—	—	—	2	
	BLE d:16	—							4				—	—	—	—	—	—	3	

**Table 2-2 Instruction Set (cont)**

**(6) Branch Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@-ERn/@ERn+	@aa	@(d,PC)	@@aa										
													Branch Condition	I	H	N	Z	V	C	Normal
JMP	JMP @ERn	—			2						PC←Ern	—	—	—	—	—	—	2		
	JMP @aa:24	—						4			PC←aa:24	—	—	—	—	—	—	3		
	JMP @@aa:8	—								2	PC←@aa:8	—	—	—	—	—	—	4	5	
BSR	BSR d:8	—							2		PC→@-SP,PC←PC+d:8	—	—	—	—	—	—	3	4	
	BSR d:16	—							4		PC→@-SP,PC←PC+d:16	—	—	—	—	—	—	4	5	
JSR	JSR @ERn	—			2						PC→@-SP,PC←ERn	—	—	—	—	—	—	3	4	
	JSR @aa:24	—						4			PC→@-SP,PC←aa:24	—	—	—	—	—	—	4	5	
	JSR @@aa:8	—								2	PC→@-SP,PC←aa:8	—	—	—	—	—	—	4	6	
RTS	RTS	—								2	PC←@SP+	—	—	—	—	—	—	4	5	

**Table 2-2 Instruction Set (cont)**

**(7) System Control Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@-ERn/@ERn+	@aa	@(d,PC)	@@aa	I		I	H	N	Z	V	C	Normal	Advanced
TRAPA	TRAPA #x:2	—									2	PC→@-SP,CCR→@-SP, EXR→@-SP,<vector>→PC	1	—	—	—	—	—	7 10	8 10
RTE	RTE	—										EXR←@SP+,CCR←@SP+, PC←@SP+	↑	↑	↑	↑	↑	↑	5 10	
SLEEP	SLEEP	—										Transition to power-down state	—	—	—	—	—	—	2	
LDC	LDC #xx:8,CCR	B	2									#xx:8→CCR	↑	↑	↑	↑	↑	↑	1	
	LDC #xx:8,EXR	B	4									#xx:8→EXR	—	—	—	—	—	—	2	
	LDC Rs,CCR	B		2								Rs8→CCR	↑	↑	↑	↑	↑	↑	1	
	LDC Rs,EXR	B		2								Rs8→EXR	—	—	—	—	—	—	1	
	LDC @ERs,CCR	W			4							@ERs→CCR	↑	↑	↑	↑	↑	↑	3	
	LDC @ERs,EXR	W			4							@ERs→EXR	—	—	—	—	—	—	3	
	LDC @(d:16,ERs),CCR	W				6						@(d:16,ERs)→CCR	↑	↑	↑	↑	↑	↑	4	
	LDC @(d:16,ERs),EXR	W				6						@(d:16,ERs)→EXR	—	—	—	—	—	—	4	
	LDC @(d:32,ERs),CCR	W				10						@(d:32,ERs)→CCR	↑	↑	↑	↑	↑	↑	6	
	LDC @(d:32,ERs),EXR	W				10						@(d:32,ERs)→EXR	—	—	—	—	—	—	6	
	LDC @ERs+,CCR	W					4					@ERs→CCR,ERs32+2→ERs32	↑	↑	↑	↑	↑	↑	4	
	LDC @ERs+,EXR	W					4					@ERs→EXR,ERs32+2→ERs32	—	—	—	—	—	—	4	
	LDC @aa:16,CCR	W						6				@aa:16→CCR	↑	↑	↑	↑	↑	↑	4	
	LDC @aa:16,EXR	W						6				@aa:16→EXR	—	—	—	—	—	—	4	
LDC @aa:32,CCR	W							8			@aa:32→CCR	↑	↑	↑	↑	↑	↑	5		
LDC @aa:32,EXR	W							8			@aa:32→EXR	—	—	—	—	—	—	5		

**Table 2-2 Instruction Set (cont)**

**(7) System Control Instructions**

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
STC	STC CCR,Rd	B		2								CCR→Rd8	—	—	—	—	—	—	1	
	STC EXR,Rd	B		2								EXR→Rd8	—	—	—	—	—	—	1	
	STC CCR,@ERd	W			4							CCR→@ERd	—	—	—	—	—	—	3	
	STC EXR,@ERd	W			4							EXR→@ERd	—	—	—	—	—	—	3	
	STC CCR,@ (d:16,ERd)	W				6						CCR→@ (d:16,ERd)	—	—	—	—	—	—	4	
	STC EXR,@ (d:16,ERd)	W				6						EXR→@ (d:16,ERd)	—	—	—	—	—	—	4	
	STC CCR,@ (d:32,ERd)	W				10						CCR→@ (d:32,ERd)	—	—	—	—	—	—	6	
	STC EXR,@ (d:32,ERd)	W				10						EXR→@ (d:32,ERd)	—	—	—	—	—	—	6	
	STC CCR,@-ERd	W					4					ERd32-2→ERd32,CCR→@ERd	—	—	—	—	—	—	4	
	STC EXR,@-ERd	W					4					ERd32-2→ERd32,EXR→@ERd	—	—	—	—	—	—	4	
	STC CCR,@aa:16	W						6				CCR→@aa:16	—	—	—	—	—	—	4	
	STC EXR,@aa:16	W						6				EXR→@aa:16	—	—	—	—	—	—	4	
STC CCR,@aa:32	W						8				CCR→@aa:32	—	—	—	—	—	—	5		
STC EXR,@aa:32	W						8				EXR→@aa:32	—	—	—	—	—	—	5		
ANDC	ANDC #xx:8,CCR	B	2									CCR^#xx:8→CCR	↑	↑	↑	↑	↑	↑	1	
	ANDC #xx:8,EXR	B	4									EXR^#xx:8→EXR	—	—	—	—	—	—	2	
ORC	ORC #xx:8,CCR	B	2									CCRv#xx:8→CCR	↑	↑	↑	↑	↑	↑	1	
	ORC #xx:8,EXR	B	4									EXRv#xx:8→EXR	—	—	—	—	—	—	2	
XORC	XORC #xx:8,CCR	B	2									CCR@#xx:8→CCR	↑	↑	↑	↑	↑	↑	1	
	XORC #xx:8,EXR	B	4									EXR@#xx:8→EXR	—	—	—	—	—	—	2	
NOP	NOP	—									2	PC←PC+2	—	—	—	—	—	—	1	



Table 2-2 Instruction Set (cont)

## (8) Block Transfer Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (Bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@ (d,ERn)	@-ERn/@ERn+	@aa	@ (d,PC)	@ @aa			I	H	N	Z	V	C	Normal	Advanced
EEPMOV	EEPMOV.B	—									4	if R4L ≠ 0 Repeat @ER5+ →@ER6+ R5+1→R5 R6+1→R6 R4L-1→R4L Until R4L=0 else next;	—	—	—	—	—	—	4+2n*2	
	EEPMOV.W	—									4	if R4 ≠ 0 Repeat @ER5+ →@ER6+ R5+1→R5 R6+1→R6 R4-1→R4 Until R4=0 else next;	—	—	—	—	—	—	4+2n*2	

Notes: 1. The number of states is the number of states required for execution when the instruction and its operands are located in on-chip memory.

2. n is the initial setting of R4L or R4.

① Seven states for saving or restoring two registers, nine states for three registers, or eleven states for four registers.

② The number of states required for execution of an instruction that transfers data in synchronization with the E clock is variable.

③ Set to 1 when a carry or borrow occurs at bit 11; otherwise cleared to 0.

④ Set to 1 when a carry or borrow occurs at bit 27; otherwise cleared to 0.

⑤ Retains its previous value when the result is zero; otherwise cleared to 0.

⑥ Set to 1 when the divisor is negative; otherwise cleared to 0.

⑦ Set to 1 when the divisor is zero; otherwise cleared to 0.

⑧ Set to 1 when the quotient is negative; otherwise cleared to 0.

⑨ MAC instruction results are indicated in the flags when the STMAC instruction is executed.

⑩ One additional state is required for execution when EXR is valid.

## 2.4 Instruction Codes

Table 2-3 Instruction Codes

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
ADD	ADD.B #xx:8,Rd	B	8	rd	IMM							
	ADD.B Rs,Rd	B	0	8	rs	rd						
	ADD.W #xx:16,Rd	W	7	9	1	rd	IMM					
	ADD.W Rs,Rd	W	0	9	rs	rd						
	ADD.L #xx:32,ERd	L	7	A	1	0:erd	IMM					
	ADD.L ERs,ERd	L	0	A	1	ers:0:erd						
ADDS	ADDS #1,ERd	L	0	B	0	0:erd						
	ADDS #2,ERd	L	0	B	8	0:erd						
	ADDS #4,ERd	L	0	B	9	0:erd						
ADDX	ADDX #xx:8,Rd	B	9	rd	IMM							
	ADDX Rs,Rd	B	0	E	rs	rd						
AND	AND.B #xx:8,Rd	B	E	rd	IMM							
	AND.B Rs,Rd	B	1	6	rs	rd						
	AND.W #xx:16,Rd	W	7	9	6	rd	IMM					
	AND.W Rs,Rd	W	6	6	rs	rd						
	AND.L #xx:32,ERd	L	7	A	6	0:erd	IMM					
	AND.L ERs,ERd	L	0	1	F	0	6	6	0:ers:0:erd			
ANDC	ANDC #xx:8,CCR	B	0	6	IMM							
	ANDC #xx:8,EXR	B	0	1	4	1	0	6	IMM			
BAND	BAND #xx:3,Rd	B	7	6	0:IMM	rd						
	BAND #xx:3,@ERd	B	7	C	0:erd	0	7	6	0:IMM	0		
	BAND #xx:3,@aa:8	B	7	E	abs	7	6	0:IMM	0			
	BAND #xx:3,@aa:16	B	6	A	1	0	abs	7	6	0:IMM	0	
	BAND #xx:3,@aa:32	B	6	A	3	0	abs		7	6	0:IMM	0
Bcc	BRA d:8 (BT d:8)	—	4	0	disp							
	BRA d:16 (BT d:16)	—	5	8	0	0	disp					
	BRN d:8 (BF d:8)	—	4	1	disp							
	BRN d:16 (BF d:16)	—	5	8	1	0	disp					
	BHI d:8	—	4	2	disp							
	BHI d:16	—	5	8	2	0	disp					
	BLS d:8	—	4	3	disp							
	BLS d:16	—	5	8	3	0	disp					
	BCC d:8 (BHS d:8)	—	4	4	disp							

Table 2-3 Instruction Codes (cont)

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
Bcc	BCC d:16 (BHS d:16)	—	5	8	4	0	disp					
	BCS d:8 (BLO d:8)	—	4	5	disp							
	BCS d:16 (BLO d:16)	—	5	8	5	0	disp					
	BNE d:8	—	4	6	disp							
	BNE d:16	—	5	8	6	0	disp					
	BEQ d:8	—	4	7	disp							
	BEQ d:16	—	5	8	7	0	disp					
	BVC d:8	—	4	8	disp							
	BVC d:16	—	5	8	8	0	disp					
	BVS d:8	—	4	9	disp							
	BVS d:16	—	5	8	9	0	disp					
	BPL d:8	—	4	A	disp							
	BPL d:16	—	5	8	A	0	disp					
	BMI d:8	—	4	B	disp							
	BMI d:16	—	5	8	B	0	disp					
	BGE d:8	—	4	C	disp							
	BGE d:16	—	5	8	C	0	disp					
	BLT d:8	—	4	D	disp							
	BLT d:16	—	5	8	D	0	disp					
	BGT d:8	—	4	E	disp							
	BGT d:16	—	5	8	E	0	disp					
	BLE d:8	—	4	F	disp							
	BLE d:16	—	5	8	F	0	disp					
BCLR	BCLR #xx:3,Rd	B	7	2	0:IMM	rd						
	BCLR #xx:3,@ERd	B	7	D	0:erd	0	7	2	0:IMM	0		
	BCLR #xx:3,@aa:8	B	7	F	abs	7	2	0:IMM	0			
	BCLR #xx:3,@aa:16	B	6	A	1	8	abs	7	2	0:IMM	0	
	BCLR #xx:3,@aa:32	B	6	A	3	8	abs		7	2	0:IMM	0
	BCLR Rn,Rd	B	6	2	rn	rd						
	BCLR Rn,@ERd	B	7	D	0:erd	0	6	2	rn	0		
	BCLR Rn,@aa:8	B	7	F	abs	6	2	rn	0			
	BCLR Rn,@aa:16	B	6	A	1	8	abs	6	2	rn	0	
	BCLR Rn,@aa:32	B	6	A	3	8	abs		6	2	rn	0

### Table 2-3 Instruction Codes (cont)

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BIAND	BIAND #xx:3,Rd	B	7	6	1:IMM: rd							
	BIAND #xx:3,@ERd	B	7	C	0:erd: 0	7	6	1:IMM: 0				
	BIAND #xx:3,@aa:8	B	7	E	abs	7	6	1:IMM: 0				
	BIAND #xx:3,@aa:16	B	6	A	1	0	abs	7	6	1:IMM: 0		
	BIAND #xx:3,@aa:32	B	6	A	3	0	abs		7	6	1:IMM: 0	
BILD	BILD #xx:3,Rd	B	7	7	1:IMM: rd							
	BILD #xx:3,@ERd	B	7	C	0:erd: 0	7	7	1:IMM: 0				
	BILD #xx:3,@aa:8	B	7	E	abs	7	7	1:IMM: 0				
	BILD #xx:3,@aa:16	B	6	A	1	0	abs	7	7	1:IMM: 0		
	BILD #xx:3,@aa:32	B	6	A	3	0	abs		7	7	1:IMM: 0	
BIOR	BIOR #xx:3,Rd	B	7	4	1:IMM: rd							
	BIOR #xx:3,@ERd	B	7	C	0:erd: 0	7	4	1:IMM: 0				
	BIOR #xx:3,@aa:8	B	7	E	abs	7	4	1:IMM: 0				
	BIOR #xx:3,@aa:16	B	6	A	1	0	abs	7	4	1:IMM: 0		
	BIOR #xx:3,@aa:32	B	6	A	3	0	abs		7	4	1:IMM: 0	
BIST	BIST #xx:3,Rd	B	6	7	1:IMM: rd							
	BIST #xx:3,@ERd	B	7	D	0:erd: 0	6	7	1:IMM: 0				
	BIST #xx:3,@aa:8	B	7	F	abs	6	7	1:IMM: 0				
	BIST #xx:3,@aa:16	B	6	A	1	8	abs	6	7	1:IMM: 0		
	BIST #xx:3,@aa:32	B	6	A	3	8	abs		6	7	1:IMM: 0	
BIXOR	BIXOR #xx:3,Rd	B	7	5	1:IMM: rd							
	BIXOR #xx:3,@ERd	B	7	C	0:erd: 0	7	5	1:IMM: 0				
	BIXOR #xx:3,@aa:8	B	7	E	abs	7	5	1:IMM: 0				
	BIXOR #xx:3,@aa:16	B	6	A	1	0	abs	7	5	1:IMM: 0		
	BIXOR #xx:3,@aa:32	B	6	A	3	0	abs		7	5	1:IMM: 0	
BLD	BLD #xx:3,Rd	B	7	7	0:IMM: rd							
	BLD #xx:3,@ERd	B	7	C	0:erd: 0	7	7	0:IMM: 0				
	BLD #xx:3,@aa:8	B	7	E	abs	7	7	0:IMM: 0				
	BLD #xx:3,@aa:16	B	6	A	1	0	abs	7	7	0:IMM: 0		
	BLD #xx:3,@aa:32	B	6	A	3	0	abs		7	7	0:IMM: 0	
BNOT	BNOT #xx:3,Rd	B	7	1	0:IMM: rd							
	BNOT #xx:3,@ERd	B	7	D	0:erd: 0	7	1	0:IMM: 0				
	BNOT #xx:3,@aa:8	B	7	F	abs	7	1	0:IMM: 0				
	BNOT #xx:3,@aa:16	B	6	A	1	8	abs	7	1	0:IMM: 0		
	BNOT #xx:3,@aa:32	B	6	A	3	8	abs		7	1	0:IMM: 0	
	BNOT Rn,Rd	B	6	1	m:rd							

**Table 2-3 Instruction Codes (cont)**

Instruction	Mnemonic	Size	Instruction Format																		
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte		9th byte		10th byte
BNOT	BNOT Rn,@ERd	B	7	D	0:erd	0	6	1	rn	0											
	BNOT Rn,@aa:8	B	7	F	abs		6	1	rn	0											
	BNOT Rn,@aa:16	B	6	A	1	8	abs				6	1	rn	0							
	BNOT Rn,@aa:32	B	6	A	3	8	abs					6	1	rn	0						
BOR	BOR #xx:3,Rd	B	7	4	0:IMM	rd									6	1	rn	0			
	BOR #xx:3,@ERd	B	7	C	0:erd	0	7	4	0:IMM	0											
	BOR #xx:3,@aa:8	B	7	E	abs		7	4	0:IMM	0											
	BOR #xx:3,@aa:16	B	6	A	1	0	abs				7	4	0:IMM	0							
	BOR #xx:3,@aa:32	B	6	A	3	0	abs					7	4	0:IMM	0						
	BSET #xx:3,Rd	B	7	0	0:IMM	rd															
	BSET #xx:3,@ERd	B	7	D	0:erd	0	7	0	0:IMM	0											
	BSET #xx:3,@aa:8	B	7	F	abs		7	0	0:IMM	0											
	BSET #xx:3,@aa:16	B	6	A	1	8	abs				7	0	0:IMM	0							
	BSET #xx:3,@aa:32	B	6	A	3	8	abs					7	0	0:IMM	0						
	BSET Rn,Rd	B	6	0	rn	rd															
	BSET Rn,@ERd	B	7	D	0:erd	0	6	0	rn	0											
	BSET Rn,@aa:8	B	7	F	abs		6	0	rn	0											
	BSET Rn,@aa:16	B	6	A	1	8	abs				6	0	rn	0							
	BSET Rn,@aa:32	B	6	A	3	8	abs					6	0	rn	0						
	BSR d:8	—	5	5	disp																
	BSR d:16	—	5	C	0	0	disp														
BST	BST #xx:3,Rd	B	6	7	0:IMM	rd															
	BST #xx:3,@ERd	B	7	D	0:erd	0	6	7	0:IMM	0											
	BST #xx:3,@aa:8	B	7	F	abs		6	7	0:IMM	0											
	BST #xx:3,@aa:16	B	6	A	1	8	abs				6	7	0:IMM	0							
	BST #xx:3,@aa:32	B	6	A	3	8	abs					6	7	0:IMM	0						
BTST	BTST #xx:3,Rd	B	7	3	0:IMM	rd															
	BTST #xx:3,@ERd	B	7	C	0:erd	0	7	3	0:IMM	0											
	BTST #xx:3,@aa:8	B	7	E	abs		7	3	0:IMM	0											
	BTST #xx:3,@aa:16	B	6	A	1	0	abs				7	3	0:IMM	0							
	BTST #xx:3,@aa:32	B	6	A	3	0	abs					7	3	0:IMM	0						
	BTST Rn,Rd	B	6	3	rn	rd															
	BTST Rn,@ERd	B	7	C	0:erd	0	6	3	rn	0											
	BTST Rn,@aa:8	B	7	E	abs		6	3	rn	0											
	BTST Rn,@aa:16	B	6	A	1	0	abs				6	3	rn	0							
	BTST Rn,@aa:32	B	6	A	3	0	abs					6	3	rn	0						

### Table 2-3 Instruction Codes (cont)

Instruction	Mnemonic	Size	Instruction Format																			
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte		9th byte		10th byte	
BXOR	BXOR #xx:3,Rd	B	7	5	0:IMM: rd																	
	BXOR #xx:3,@ERd	B	7	C	0:erd: 0		7	5	0:IMM: 0													
	BXOR #xx:3,@aa:8	B	7	E	abs		7	5	0:IMM: 0													
	BXOR #xx:3,@aa:16	B	6	A	1	0	abs			7	5	0:IMM: 0										
	BXOR #xx:3,@aa:32	B	6	A	3	0	abs						7	5	0:IMM: 0							
CLRMAC <sup>*</sup>	CLRMAC	—	0	1	A	0									7	5	0:IMM: 0					
CMP	CMP.B #xx:8,Rd	B	A	rd	IMM																	
	CMP.B Rs,Rd	B	1	C	rs	rd																
	CMP.W #xx:16,Rd	W	7	9	2	rd	IMM															
	CMP.W Rd,Rd	W	1	D	rs	rd																
	CMP.L #xx:32,ERd	L	7	A	2	0:erd	IMM															
	CMP.L ERs,ERd	L	1	F	1:ers	0:erd																
DAA	DAA Rd	B	0	F	0	rd																
DAS	DAS Rd	B	1	F	0	rd																
DEC	DEC.B Rd	B	1	A	0	rd																
	DEC.W #1,Rd	W	1	B	5	rd																
	DEC.W #2,Rd	W	1	B	D	rd																
	DEC.L #1,ERd	L	1	B	7	0:erd																
	DEC.L #2,ERd	L	1	B	F	0:erd																
DIVXS	DIVXS.B Rs,Rd	B	0	1	D	0	5	1	rs	rd												
	DIVXS.W Rs,ERd	W	0	1	D	0	5	3	rs	0:erd												
DIVXU	DIVXU.B Rs,Rd	B	5	1	rs	rd																
	DIVXU.W Rs,ERd	W	5	3	rs	0:erd																
EEPMOV	EEPMOV.B	—	7	B	5	C	5	9	8	F												
	EEPMOV.W	—	7	B	D	4	5	9	8	F												
EXTS	EXTS.W Rd	W	1	7	D	rd																
	EXTS.L ERd	L	1	7	F	0:erd																
XTU	XTU.W Rd	W	1	7	5	rd																
	XTU.L ERd	L	1	7	7	0:erd																
INC	INC.B Rd	B	0	A	0	rd																
	INC.W #1,Rd	W	0	B	5	rd																
	INC.W #2,Rd	W	0	B	D	rd																
	INC.L #1,ERd	L	0	B	7	0:erd																
	INC.L #2,ERd	L	0	B	F	0:erd																

Table 2-3 Instruction Codes (cont)

Instruction	Mnemonic	Size	Instruction Format																	
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte		9th byte	
JMP	JMP @ERn	—	5	9	0:ern		0													
	JMP @aa:24	—	5	A	abs															
	JMP @aa:8	—	5	B	abs															
JSR	JSR @ERn	—	5	D	0:ern		0													
	JSR @aa:24	—	5	E	abs															
	JSR @ @aa:8	—	5	F	abs															
LDC	LDC #xx:8,CCR	B	0	7	IMM															
	LDC #xx:8,EXR	B	0	1	4	1	0	7	IMM											
	LDC Rs,CCR	B	0	3	0	rs														
	LDC Rs,EXR	B	0	3	1	rs														
	LDC @ERs,CCR	W	0	1	4	0	6	9	0:ers	0										
	LDC @ERs,EXR	W	0	1	4	1	6	9	0:ers	0										
	LDC @(d:16,ERs),CCR	W	0	1	4	0	6	F	0:ers	0	disp									
	LDC @(d:16,ERs),EXR	W	0	1	4	1	6	F	0:ers	0	disp									
	LDC @(d:32,ERs),CCR	W	0	1	4	0	7	8	0:ers	0	6	B	2	0	disp					
	LDC @(d:32,ERs),EXR	W	0	1	4	1	7	8	0:ers	0	6	B	2	0	disp					
	LDC @ERs+,CCR	W	0	1	4	0	6	D	0:ers	0										
	LDC @ERs+,EXR	W	0	1	4	1	6	D	0:ers	0										
	LDC @aa:16,CCR	W	0	1	4	0	6	B	0	0	abs									
	LDC @aa:16,EXR	W	0	1	4	1	6	B	0	0	abs									
	LDC @aa:32,CCR	W	0	1	4	0	6	B	2	0	abs									
LDC @aa:32,EXR	W	0	1	4	1	6	B	2	0	abs										
LDM	LDM.L @SP+,(ERn-ERn+1	L	0	1	1	0	6	D	7	0:ern+1										
	LDM.L @SP+,(ERn-ERn+2	L	0	1	2	0	6	D	7	0:ern+2										
	LDM.L @SP+,(ERn-ERn+3	L	0	1	3	0	6	D	7	0:ern+3										
LDMAC*	LDMAC ERS,MACH	L	0	3	2	0:ers														
	LDMAC ERs,MACL	L	0	3	3	0:ers														
MAC*	MAC @ERn+,@ERm+	—	0	1	6	0	6	D	0:ern	0:erm										
MOV	MOV.B #xx:8,Rd	B	F	rd	IMM															
	MOV.B Rs,Rd	B	0	C	rs	rd														
	MOV.B @ERs,Rd	B	6	8	0:ers	rd														
	MOV.B @(d:16,ERs),Rd	B	6	E	0:ers	rd	disp													
	MOV.B @(d:32,ERs),Rd	B	7	8	0:ers	0	6	A	2	rd	disp									
	MOV.B @ERs+,Rd	B	6	C	0:ers	rd														
	MOV.B @aa:8,Rd	B	2	rd	abs															
	MOV.B @aa:16,Rd	B	6	A	0	rd	abs													

**Table 2-3 Instruction Codes (cont)**

Instruction	Mnemonic	Size	Instruction Format																		
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte		9th byte		10th byte
MOV	MOV.B @aa:32,Rd	B	6	A	2	rd	abs														
	MOV.B Rs,@ERd	B	6	8	1	erd	rs														
	MOV.B Rs,@(d:16,ERd)	B	6	E	1	erd	rs	disp													
	MOV.B Rs,@(d:32,ERd)	B	7	8	0	erd	0	6	A	A	rs	disp									
	MOV.B Rs,@-ERd	B	6	C	1	erd	rs														
	MOV.B Rs,@aa:8	B	3	rs	abs																
	MOV.B Rs,@aa:16	B	6	A	8	rs	abs														
	MOV.B Rs,@aa:32	B	6	A	A	rs	abs														
	MOV.W #xx:16,Rd	W	7	9	0	rd	IMM														
	MOV.W Rs,Rd	W	0	D	rs	rd															
	MOV.W @ERs,Rd	W	6	9	0	ers	rd														
	MOV.W @(d:16,ERs),Rd	W	6	F	0	ers	rd	disp													
	MOV.W @(d:32,ERs),Rd	W	7	8	0	ers	0	6	B	2	rd	disp									
	MOV.W @ERs+,Rd	W	6	D	0	ers	rd														
	MOV.W @aa:16,Rd	W	6	B	0	rd	abs														
	MOV.W @aa:32,Rd	W	6	B	2	rd	abs														
	MOV.W Rs,@ERd	W	6	9	1	erd	rs														
	MOV.W Rs,@(d:16,ERd)	W	6	F	1	erd	rs	disp													
	MOV.W Rs,@(d:32,ERd)	W	7	8	0	erd	0	6	B	A	rs	disp									
	MOV.W Rs,@-ERd	W	6	D	1	erd	rs														
	MOV.W Rs,@aa:16	W	6	B	8	rs	abs														
	MOV.W Rs,@aa:32	W	6	B	A	rs	abs														
	MOV.L #xx:32,Rd	L	7	A	0	0	erd	IMM													
	MOV.L ERs,ERd	L	0	F	1	ers	0	erd													
	MOV.L @ERs,ERd	L	0	1	0	0	0	6	9	0	ers	0	erd								
	MOV.L @(d:16,ERs),ERd	L	0	1	0	0	0	6	F	0	ers	0	erd	disp							
	MOV.L @(d:32,ERs),ERd*	L	0	1	0	0	0	7	8	0	ers	0	6	B	2	0	erd	disp			
	MOV.L @ERs+,ERd	L	0	1	0	0	0	6	D	0	ers	0	erd								
	MOV.L @aa:16,ERd	L	0	1	0	0	0	6	B	0	0	erd	abs								
	MOV.L @aa:32,ERd	L	0	1	0	0	0	6	B	2	0	erd	abs								
	MOV.L ERs,@ERd	L	0	1	0	0	0	6	9	1	erd	0	ers								
	MOV.L ERs,@(d:16,ERd)	L	0	1	0	0	0	6	F	1	erd	0	ers	disp							
	MOV.L ERs,@(d:32,ERd)	L	0	1	0	0	0	7	8	1	erd	0	6	B	A	0	ers	disp			
	MOV.L ERs,@-ERd	L	0	1	0	0	0	6	D	1	erd	0	ers								
	MOV.L ERs,@aa:16	L	0	1	0	0	0	6	B	8	0	ers	abs								
	MOV.L ERs,@aa:32	L	0	1	0	0	0	6	B	A	0	ers	abs								

Note: \* The MOV.L ERs,@ (d: 32, ERd) instruction will operate with bit 7 of the 4th byte set to either 1 or 0.



Table 2-3 Instruction Codes (cont)

Instruction	Mnemonic	Size	Instruction Format													
			1st byte		2nd byte		3rd byte		4th byte		5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
MOVFPE	MOVFPE @aa:16,Rd	B	6	A	4	rd	abs									
MOVTPE	MOVTPE Rs,@aa:16	B	6	A	C	rs	abs									
MULXS	MULXS.B Rs,Rd	B	0	1	C	0	5	0	rs	rd						
	MULXS.W Rs,ERd	W	0	1	C	0	5	2	rs	0:erd						
MULXU	MULXU.B Rs,Rd	B	5	0	rs	rd										
	MULXU.W Rs,ERd	W	5	2	rs	0:erd										
NEG	NEG.B Rd	B	1	7	8	rd										
	NEG.W Rd	W	1	7	9	rd										
	NEG.L ERd	L	1	7	B	0:erd										
NOP	NOP	—	0	0	0	0										
NOT	NOT.B Rd	B	1	7	0	rd										
	NOT.W Rd	W	1	7	1	rd										
	NOT.L ERd	L	1	7	3	0:erd										
OR	OR.B #xx:8,Rd	B	C	rd	IMM											
	OR.B Rs,Rd	B	1	4	rs	rd										
	OR.W #xx:16,Rd	W	7	9	4	rd	IMM									
	OR.W Rs,Rd	W	6	4	rs	rd										
	OR.L #xx:32,ERd	L	7	A	4	0:erd	IMM									
	OR.L ERs,ERd	L	0	1	F	0	6	4	0:ers	0:erd						
ORC	ORC #xx:8,CCR	B	0	4	IMM											
	ORC #xx:8,EXR	B	0	1	4	1	0	4	IMM							
POP	POP.W Rn	W	6	D	7	m										
	POP.L ERn	L	0	1	0	0	6	D	7	0:ern						
PUSH	PUSH.W Rn	W	6	D	F	m										
	PUSH.L ERn	L	0	1	0	0	6	D	F	0:ern						
ROTL	ROTL.B Rd	B	1	2	8	rd										
	ROTL.B #2,Rd	B	1	2	C	rd										
	ROTL.W Rd	W	1	2	9	rd										
	ROTL.W #2,Rd	W	1	2	D	rd										
	ROTL.L ERd	L	1	2	B	0:erd										
	ROTL.L #2,ERd	L	1	2	F	0:erd										
ROTR	ROTR.B Rd	B	1	3	8	rd										
	ROTR.B #2,Rd	B	1	3	C	rd										
	ROTR.W Rd	W	1	3	9	rd										
	ROTR.W #2,Rd	W	1	3	D	rd										

**Table 2-3 Instruction Codes (cont)**

Instruction	Mnemonic	Size	Instruction Format											
			1st byte		2nd byte		3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
ROTR	ROTR.L ERd	L	1	3	B	0: erd								
	ROTR.L #2,ERd	L	1	3	F	0: erd								
ROTXL	ROTXL.B Rd	B	1	2	0	rd								
	ROTXL.B #2,Rd	B	1	2	4	rd								
	ROTXL.W Rd	W	1	2	1	rd								
	ROTXL.W #2,Rd	W	1	2	5	rd								
	ROTXL.L ERd	L	1	2	3	0: erd								
	ROTXL.L #2,ERd	L	1	2	7	0: erd								
ROTXR	ROTXR.B Rd	B	1	3	0	rd								
	ROTXR.B #2,Rd	B	1	3	4	rd								
	ROTXR.W Rd	W	1	3	1	rd								
	ROTXR.W #2,Rd	W	1	3	5	rd								
	ROTXR.L ERd	L	1	3	3	0: erd								
	ROTXR.L #2,ERd	L	1	3	7	0: erd								
RTE	RTE	—	5	6	7	0								
RTS	RTS	—	5	4	7	0								
SHAL	SHAL.B Rd	B	1	0	8	rd								
	SHAL.B #2,Rd	B	1	0	C	rd								
	SHAL.W Rd	W	1	0	9	rd								
	SHAL.W #2,Rd	W	1	0	D	rd								
	SHAL.L ERd	L	1	0	B	0: erd								
	SHAL.L #2,ERd	L	1	0	F	0: erd								
SHAR	SHAR.B Rd	B	1	1	8	rd								
	SHAR.B #2,Rd	B	1	1	C	rd								
	SHAR.W Rd	W	1	1	9	rd								
	SHAR.W #2,Rd	W	1	1	D	rd								
	SHAR.L ERd	L	1	1	B	0: erd								
	SHAR.L #2,ERd	L	1	1	F	0: erd								
SHLL	SHLL.B Rd	B	1	0	0	rd								
	SHLL.B #2,Rd	B	1	0	4	rd								
	SHLL.W Rd	W	1	0	1	rd								
	SHLL.W #2,Rd	W	1	0	5	rd								
	SHLL.L ERd	L	1	0	3	0: erd								
	SHLL.L #2,ERd	L	1	0	7	0: erd								
SHLR	SHLR.B Rd	B	1	1	0	rd								
	SHLR.B #2,Rd	B	1	1	4	rd								

### Table 2-3 Instruction Codes (cont)

[illegible]

**Table 2-3 Instruction Codes (cont)**

Instruction	Mnemonic	Size	Instruction Format																			
			1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte		9th byte		10th byte	
XOR	XOR.B #xx:8,Rd	B	D	rd	IMM																	
	XOR.B Rs,Rd	B	1	5	rs	rd																
	XOR.W #xx:16,Rd	W	7	9	5	rd	IMM															
	XOR.W Rs,Rd	W	6	5	rs	rd																
	XOR.L #xx:32,ERd	L	7	A	5	0: erd	IMM															
	XOR.L ERs,ERd	L	0	1	F	0	6	5	0: ers	0: erd												
XORC	XORC #xx:8,CCR	B	0	5	IMM																	
	XORC #xx:8,EXR	B	0	1	4	1	0	5	IMM													

Note: \*These instructions are supported by the H8S/2600 CPU only.

**Legend**

IMM: Immediate data (2, 3, 8, 16, or 32 bits)

abs: Absolute address (8, 16, 24, or 32 bits)

disp: Displacement (8, 16, or 32 bits)

rs, rd, rn: Register field (4 bits specifying an 8-bit or 16-bit register. The symbols rs, rd, and rn correspond to operand symbols Rs, Rd, and Rn.)

ers, erd, ern, erm: Register field (3 bits specifying an address register or 32-bit register. The symbols ers, erd, ern, and erm correspond to operand symbols ERs, ERd, ERn, and ERm.)

The register fields specify general registers as follows.

Address Register 32-Bit Register		16-Bit Register		8-Bit Register	
Register Field	General Register	Register Field	General Register	Register Field	General Register
000	ER0	0000	R0	0000	R0H
001	ER1	0001	R1	0001	R1H
⋮	⋮	⋮	⋮	⋮	⋮
111	ER7	0111	R7	0111	R7H
		1000	E0	1000	R0L
		1001	E1	1001	R1L
		⋮	⋮	⋮	⋮
		1111	E7	1111	R7L

2.5 Operation Code Map

Table 2-4 shows an operation code map.

Table 2-4 Operation Code Map (1)

Operation Code:

1st byte		2nd byte	
AH	AL	BH	BL

AL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
AH	0	NOP	Table 2-4 (2)	STC	LDC	ORC	XORC	ANDC	LDC	ADD	Table 2-4 (2)	Table 2-4 (2)	MOV	ADDX	Table 2-4 (2)		
	1	Table 2-4 (2)	Table 2-4 (2)	STMAC*	LDMAC*	OR	XOR	AND	Table 2-4 (2)	SUB	Table 2-4 (2)	Table 2-4 (2)	CMP	SUBX	Table 2-4 (2)		
	2	MOV.B															
	3																
	4	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
	5	MULXU	DIVXU	MULXU	DIVXU	RTS	BSR	RTE	TRAPA	Table 2-4 (2)	JMP	BSR	JSR				
	6	BSET	BNOT	BCLR	BTST	OR	XOR	AND	BST	MOV	Table 2-4 (2)	MOV					
	7					BOR	BXOR	BAND	BLD								BIST
	8	ADD															
	9	ADDX															
	A	CMP															
	B	SUBX															
	C	OR															
	D	XOR															
	E	AND															
	F	MOV															

Note: \*These instructions are supported by the H8S/2600 CPU only.

**Table 2-4 Operation Code Map (2)**

**Operation Code:**

1st byte		2nd byte	
AH	AL	BH	BL

BH AH AL		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F					
01	MOV	LDM			STM		LDC		STC			MAC*			SLEEP		CLRMAC*		Table 2-4 (3)	Table 2-4 (3)	TAS	Table 2-4 (3)
0A	INC									ADD												
0B	ADDS						INC		INC	ADDS					INC		INC					
0F	DAA									MOV												
10	SHLL		SHLL				SHLL	SHAL		SHAL			SHAL									
11	SHLR		SHLR				SHLR	SHAR		SHAR			SHAR									
12	ROTXL		ROTXL				ROTXL	ROTL		ROTL			ROTL									
13	ROTXR		ROTXR				ROTXR	ROTR		ROTR			ROTR									
17	NOT		NOT		EXTU		EXTU	NEG		NEG		EXTS		EXTS								
1A	DEC									SUB												
1B	SUBS						DEC		DEC	SUBS					DEC		DEC					
1F	DAS	BNE								CMP												
58	BRA	BRN	BHI	BLS	BCC	BCS		BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE						
6A	MOV	Table 2-4 (4)	MOV	Table 2-4 (4)	MOVFP				MOV		MOV		MOVTPE									
79	MOV	ADD	CMP	SUB	OR	XOR	AND															
7A	MOV	ADD	CMP	SUB	OR	XOR																

Note: \*These instructions are supported by the H8S/2600 CPU only.

**Table 2-4 Operation Code Map (3)****Operation Code:**

1st byte		2nd byte		3rd byte		4th byte	
AH	AL	BH	BL	CH	CL	DH	DL



Instruction when most significant bit of DH is 0.

Instruction when most significant bit of DH is 1.

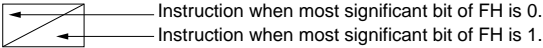
CL AHALBHLCH	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01C05	MULXS		MULXS													
01D05		DIVXS		DIVXS												
01F06					OR	XOR	AND									
7Cr06*1				BTST												
7Cr07*1				BTST	BOR BIOR	BXOR BIXOR	BAND BIAND	BLD BILD								
7Dr06*1	BSET	BNOT	BCLR					BST BIST								
7Dr07*1	BSET	BNOT	BCLR													
7Eaa6*2				BTST												
7Eaa7*2				BTST	BOR BIOR	BXOR BIXOR	BAND BIAND	BLD BILD								
7Faa6*2	BSET	BNOT	BCLR					BST BIST								
7Faa7*2	BSET	BNOT	BCLR													

Notes: 1. The letter "r" indicates a register field.  
 2. The letters "aa" indicate an absolute address field.

Table 2-4 Operation Code Map (4)

Operation Code:

1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte	
AH	AL	BH	BL	CH	CL	DH	DL	EH	EL	FH	FL

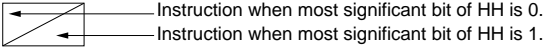


<div>EL</div> <div>AHALBHLCHCLDHDLEH</div>	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
6A10aaaa6*				BTST												
6A10aaaa7*					BOR BIOR	BXOR BIXOR	BAND BIAND	BLD BILD								
6A18aaaa6*	BSET	BNOT	BCLR					BST BIST								
6A18aaaa7*																

278

Operation Code:

1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte	
AH	AL	BH	BL	CH	CL	DH	DL	EH	EL	FH	FL	GH	GL	HH	HL



<div>EL</div> <div>AHALBHL ... FHFLGH</div>	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
6A30aaaaaaaa6*				BTST												
6A30aaaaaaaa7*					BOR BIOR	BXOR BIXOR	BAND BIAND	BLD BILD								
6A38aaaaaaaa6*	BSET	BNOT	BCLR					BST BIST								
6A38aaaaaaaa7*																

Note: \* The letters “aa” indicate an absolute address field.



## 2.6 Number of States Required for Instruction Execution

The tables in this section can be used to calculate the number of states required for instruction execution by the CPU. Table 2-6 indicates the number of instruction fetch, data read/write, and other cycles occurring in each instruction. Table 2-5 indicates the number of states required for each cycle, depending on its size. The number of states required for each cycle depends on the product. See the hardware manuals for the relevant product for details. The number of states required for execution of an instruction can be calculated from these two tables as follows:

$$\text{Execution states} = I \times S_I + J \times S_J + K \times S_K + L \times S_L + M \times S_M + N \times S_N$$

**Examples:** Advanced mode, program code and stack located in external memory, on-chip supporting modules accessed in two states with 8-bit bus width, external devices accessed in three states with one wait state and 16-bit bus width.

1. BSET #0, @FFFC7:8

From table 2-6:

$$I = L = 2, \quad J = K = M = N = 0$$

From table 2-5:

$$S_I = 4, \quad S_L = 2$$

$$\text{Number of states required for execution} = 2 \times 4 + 2 \times 2 = 12$$

2. JSR @@30

From table 2-6:

$$I = J = K = 2, \quad L = M = N = 0$$

From table 2-5:

$$S_I = S_J = S_K = 4$$

$$\text{Number of states required for execution} = 2 \times 4 + 2 \times 4 + 2 \times 4 = 24$$

**Table 2-5 Number of States per Cycle**

			Access Conditions					
			On-Chip Supporting Module		External Device			
					8-Bit Bus		16-Bit Bus	
			8-Bit Bus	16-Bit Bus	2-State Access	3-State Access	2-State Access	3-State Access
Instruction fetch	$S_I$	1	2n	n	4	6 + 2m	2	3 + m*
Branch address read	$S_J$							
Stack operation	$S_K$							
Byte data access	$S_L$		n		2	3 + m		
Word data access	$S_M$		2n		4	6 + 2m		
Internal operation	$S_N$	1	1	1	1	1	1	1

Note: \* For the MOVFPE and MOVTPPE instructions, refer to the relevant microcontroller hardware manual.

**Legend**

m: Number of wait states inserted into external device access

n: Number of states required for access to an on-chip supporting module. For the specific number, refer to the relevant microcontroller hardware manual.

**Table 2-6 Number of Cycles in Instruction Execution**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
ADD	ADD.B #xx:8,Rd	1					
	ADD.B Rs,Rd	1					
	ADD.W #xx:16,Rd	2					
	ADD.W Rs,Rd	1					
	ADD.L #xx:32,ERd	3					
	ADD.L ERs,ERd	1					
ADDS	ADDS #1/2/4,ERd	1					
ADDX	ADDX #xx:8,Rd	1					
	ADDX Rs,Rd	1					
AND	AND.B #xx:8,Rd	1					
	AND.B Rs,Rd	1					
	AND.W #xx:16,Rd	2					
	AND.W Rs,Rd	1					
	AND.L #xx:32,ERd	3					
	AND.L ERs,ERd	2					
ANDC	ANDC #xx:8,CCR	1					
	ANDC #xx:8,EXR	2					
BAND	BAND #xx:3,Rd	1					
	BAND #xx:3,@ERd	2			1		
	BAND #xx:3,@aa:8	2			1		
	BAND #xx:3,@aa:16	3			1		
	BAND #xx:3,@aa:32	4			1		
Bcc	BRA d:8 (BT d:8)	2					
	BRN d:8 (BF d:8)	2					
	BHI d:8	2					
	BLS d:8	2					
	BCC d:8 (BHS d:8)	2					
	BCS d:8 (BLO d:8)	2					
	BNE d:8	2					
	BEQ d:8	2					
	BVC d:8	2					
	BVS d:8	2					
	BPL d:8	2					
	BMI d:8	2					
	BGE d:8	2					
	BLT d:8	2					
	BGT d:8	2					
	BLE d:8	2					
	BRA d:16 (BT d:16)	2					1
	BRN d:16 (BF d:16)	2					1

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
Bcc	BHI d:16	2					1
	BLS d:16	2					1
	BCC d:16 (BHS d:16)	2					1
	BCS d:16 (BLO d:16)	2					1
	BNE d:16	2					1
	BEQ d:16	2					1
	BVC d:16	2					1
	BVS d:16	2					1
	BPL d:16	2					1
	BMI d:16	2					1
	BGE d:16	2					1
	BLT d:16	2					1
	BGT d:16	2					1
	BLE d:16	2					1
BCLR	BCLR #xx:3,Rd	1					
	BCLR #xx:3,@ERd	2			2		
	BCLR #xx:3,@aa:8	2			2		
	BCLR #xx:3,@aa:16	3			2		
	BCLR #xx:3,@aa:32	4			2		
	BCLR Rn,Rd	1					
	BCLR Rn,@ERd	2			2		
	BCLR Rn,@aa:8	2			2		
	BCLR Rn,@aa:16	3			2		
	BCLR Rn,@aa:32	4			2		
BIAND	BIAND #xx:3,Rd	1					
	BIAND #xx:3,@ERd	2			1		
	BIAND #xx:3,@aa:8	2			1		
	BIAND #xx:3,@aa:16	3			1		
	BIAND #xx:3,@aa:32	4			1		
BILD	BILD #xx:3,Rd	1					
	BILD #xx:3,@ERd	2			1		
	BILD #xx:3,@aa:8	2			1		
	BILD #xx:3,@aa:16	3			1		
	BILD #xx:3,@aa:32	4			1		
BIOR	BIOR #xx:8,Rd	1					
	BIOR #xx:8,@ERd	2			1		
	BIOR #xx:8,@aa:8	2			1		
	BIOR #xx:8,@aa:16	3			1		
	BIOR #xx:8,@aa:32	4			1		

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
BIST	BIST #xx:3,Rd	1					
	BIST #xx:3,@ERd	2			2		
	BIST #xx:3,@aa:8	2			2		
	BIST #xx:3,@aa:16	3			2		
	BIST #xx:3,@aa:32	4			2		
BIXOR	BIXOR #xx:3,Rd	1					
	BIXOR #xx:3,@ERd	2			1		
	BIXOR #xx:3,@aa:8	2			1		
	BIXOR #xx:3,@aa:16	3			1		
	BIXOR #xx:3,@aa:32	4			1		
BLD	BLD #xx:3,Rd	1					
	BLD #xx:3,@ERd	2			1		
	BLD #xx:3,@aa:8	2			1		
	BLD #xx:3,@aa:16	3			1		
	BLD #xx:3,@aa:32	4			1		
BNOT	BNOT #xx:3,Rd	1					
	BNOT #xx:3,@ERd	2			2		
	BNOT #xx:3,@aa:8	2			2		
	BNOT #xx:3,@aa:16	3			2		
	BNOT #xx:3,@aa:32	4			2		
	BNOT Rn,Rd	1					
	BNOT Rn,@ERd	2			2		
	BNOT Rn,@aa:8	2			2		
	BNOT Rn,@aa:16	3			2		
	BNOT Rn,@aa:32	4			2		
BOR	BOR #xx:3,Rd	1					
	BOR #xx:3,@ERd	2			1		
	BOR #xx:3,@aa:8	2			1		
	BOR #xx:3,@aa:16	3			1		
	BOR #xx:3,@aa:32	4			1		
BSET	BSET #xx:3,Rd	1					
	BSET #xx:3,@ERd	2			2		
	BSET #xx:3,@aa:8	2			2		
	BSET #xx:3,@aa:16	3			2		
	BSET #xx:3,@aa:32	4			2		
	BSET Rn,Rd	1					
	BSET Rn,@ERd	2			2		
	BSET Rn,@aa:8	2			2		
	BSET Rn,@aa:16	3			2		
	BSET Rn,@aa:32	4			2		

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
			I	J	K	L	M	N
BSR	BSR d:8	Normal	2		1			
		Advanced	2		2			
	BSR d:16	Normal	2		1			1
		Advanced	2		2			1
BST	BST #xx:3,Rd		1					
	BST #xx:3,@ERd		2			2		
	BST #xx:3,@aa:8		2			2		
	BST #xx:3,@aa:16		3			2		
	BST #xx:3,@aa:32		4			2		
BTST	BTST #xx:3,Rd		1					
	BTST #xx:3,@ERd		2			1		
	BTST #xx:3,@aa:8		2			1		
	BTST #xx:3,@aa:16		3			1		
	BTST #xx:3,@aa:32		4			1		
	BTST Rn,Rd		1					
	BTST Rn,@ERd		2			1		
	BTST Rn,@aa:8		2			1		
	BTST Rn,@aa:16		3			1		
	BTST Rn,@aa:32		4			1		
BXOR	BXOR #xx:3,Rd		1					
	BXOR #xx:3,@ERd		2			1		
	BXOR #xx:3,@aa:8		2			1		
	BXOR #xx:3,@aa:16		3			1		
	BXOR #xx:3,@aa:32		4			1		
CLRMAC	CLRMAC		1					1
CMP	CMP.B #xx:8,Rd		1					
	CMP.B Rs,Rd		1					
	CMP.W #xx:16,Rd		2					
	CMP.W Rs,Rd		1					
	CMP.L #xx:32,ERd		3					
	CMP.L ERs,ERd		1					
DAA	DAA Rd		1					
DAS	DAS Rd		1					
DEC	DEC.B Rd		1					
	DEC.W #1/2,Rd		1					
	DEC.L #1/2,ERd		1					
DIVXS	DIVXS.B Rs,Rd		2					11
	DIVXS.W Rs,ERd		2					19
DIVXU	DIVXU.B Rs,Rd		1					11
	DIVXU.W Rs,ERd		1					19

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic		Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
			I	J	K	L	M	N
EEPMOV	EEPMOV.B		2			2n + 2 <sup>+1</sup>		
	EEPMOV.W		2			2n + 2 <sup>+1</sup>		
EXTS	EXTS.W Rd		1					
	EXTS.L ERd		1					
EXTU	EXTU.W Rd		1					
	EXTU.L ERd		1					
INC	INC.B Rd		1					
	INC.W #1/2,Rd		1					
	INC.L #1/2,ERd		1					
JMP	JMP @ERn		2					
	JMP @aa:24		2					1
	JMP @aa:8	Normal	2	1				1
		Advanced	2	2				1
JSR	JSR @ERn	Normal	2		1			
		Advanced	2		2			
	JSR @aa:24	Normal	2		1			1
		Advanced	2		2			1
	JSR @aa:8	Normal	2	1	1			
		Advanced	2	2	2			
LDC	LDC #xx:8,CCR		1					
	LDC #xx:8,EXR		2					
	LDC Rs,CCR		1					
	LDC Rs,EXR		1					
	LDC @ERs,CCR		2				1	
	LDC @ERs,EXR		2				1	
	LDC @(d:16,ERs),CCR		3				1	
	LDC @(d:16,ERs),EXR		3				1	
	LDC @(d:32,ERs),CCR		5				1	
	LDC @(d:32,ERs),EXR		5				1	
	LDC @ERs+,CCR		2				1	1
	LDC @ERs+,EXR		2				1	1
	LDC @aa:16,CCR		3				1	
	LDC @aa:16,EXR		3				1	
	LDC @aa:32,CCR		4				1	
	LDC @aa:32,EXR		4				1	
LDM	LDM.L @SP+,(ERn-ERn+1)		2		4			1
	LDM.L @SP+,(ERn-ERn+2)		2		6			1
	LDM.L @SP+,(ERn-ERn+3)		2		8			1
LDMAC*	LDMAC ERs,MACH		1					1
	LDMAC ERs,MACL		1					1

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
MAC*	MAC @ERn+,@ERm+	2				2	
MOV	MOV.B #xx:8,Rd	1					
	MOV.B Rs,Rd	1					
	MOV.B @ERs,Rd	1			1		
	MOV.B @(d:16,ERs),Rd	2			1		
	MOV.B @(d:32,ERs),Rd	4			1		
	MOV.B @ERs+,Rd	1			1		1
	MOV.B @aa:8,Rd	1			1		
	MOV.B @aa:16,Rd	2			1		
	MOV.B @aa:32,Rd	3			1		
	MOV.B Rs,@ERd	1			1		
	MOV.B Rs,@(d:16,ERd)	2			1		
	MOV.B Rs,@(d:32,ERd)	4			1		
	MOV.B Rs,@-ERd	1			1		1
	MOV.B Rs,@aa:8	1			1		
	MOV.B Rs,@aa:16	2			1		
	MOV.B Rs,@aa:32	3			1		
	MOV.W #xx:16,Rd	2					
	MOV.W Rs,Rd	1					
	MOV.W @ERs,Rd	1				1	
	MOV.W @(d:16,ERs),Rd	2				1	
	MOV.W @(d:32,ERs),Rd	4				1	
	MOV.W @ERs+,Rd	1				1	1
	MOV.W @aa:16,Rd	2				1	
	MOV.W @aa:32,Rd	3				1	
	MOV.W Rs,@ERd	1				1	
	MOV.W Rs,@(d:16,ERd)	2				1	
	MOV.W Rs,@(d:32,ERd)	4				1	
	MOV.W Rs,@-ERd	1				1	1
	MOV.W Rs,@aa:16	2				1	
	MOV.W Rs,@aa:32	3				1	
	MOV.L #xx:32,ERd	3					
	MOV.L ERs,ERd	1					
	MOV.L @ERs,ERd	2				2	
	MOV.L @(d:16,ERs),ERd	3				2	
	MOV.L @(d:32,ERs),ERd	5				2	
	MOV.L @ERs+,ERd	2				2	1
	MOV.L @aa:16,ERd	3				2	
	MOV.L @aa:32,ERd	4				2	
	MOV.L ERs,@ERd	2				2	
	MOV.L ERs,@(d:16,ERd)	3				2	



**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
MOV	MOV.L ERs, @(d:32,ERd)	5				2	
	MOV.L ERs, @-ERd	2				2	1
	MOV.L ERs, @aa:16	3				2	
	MOV.L ERs, @aa:32	4				2	
MOVFPE	MOVFPE @:aa:16,Rd	2			1*2		
MOVTPE	MOVTPE Rs,@:aa:16	2			1*2		
MULXS	MULXS.B Rs,Rd H8S/2600	2					2
	H8S/2000	2					10
	MULXS.W Rs,ERd H8S/2600	2					3
	H8S/2000	2					18
MULXU	MULXU.B Rs,Rd H8S/2600	1					2
	H8S/2000	1					10
	MULXU.W Rs,ERd H8S/2600	1					3
	H8S/2000	1					18
NEG	NEG.B Rd	1					
	NEG.W Rd	1					
	NEG.L ERd	1					
NOP	NOP	1					
NOT	NOT.B Rd	1					
	NOT.W Rd	1					
	NOT.L ERd	1					
OR	OR.B #xx:8,Rd	1					
	OR.B Rs,Rd	1					
	OR.W #xx:16,Rd	2					
	OR.W Rs,Rd	1					
	OR.L #xx:32,ERd	3					
	OR.L ERs,ERd	2					
ORC	ORC #xx:8,CCR	1					
	ORC #xx:8,EXR	2					
POP	POP.W Rn	1				1	1
	POP.L ERn	2				2	1
PUSH	PUSH.W Rn	1				1	1
	PUSH.L ERn	2				2	1
ROTL	ROTL.B Rd	1					
	ROTL.B #2,Rd	1					
	ROTL.W Rd	1					
	ROTL.W #2,Rd	1					
	ROTL.L ERd	1					
	ROTL.L #2,ERd	1					
ROTR	ROTR.B Rd	1					
	ROTR.B #2,Rd	1					
	ROTR.W Rd	1					
	ROTR.W #2,Rd	1					
	ROTR.L ERd	1					
	ROTR.L #2,ERd	1					

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction					
		Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
ROTXL	ROTXL.B Rd	1					
	ROTXL.B #2,Rd	1					
	ROTXL.W Rd	1					
	ROTXL.W #2,Rd	1					
	ROTXL.L ERd	1					
	ROTXL.L #2,ERd	1					
ROTXR	ROTXR.B Rd	1					
	ROTXR.B #2,Rd	1					
	ROTXR.W Rd	1					
	ROTXR.W #2,Rd	1					
	ROTXR.L ERd	1					
	ROTXR.L #2,ERd	1					
RTE	RTE	2		2/3 *1			1
RTS	RTS	Normal	2	1			1
		Advanced	2	2			1
SHAL	SHAL.B Rd	1					
	SHAL.B #2,Rd	1					
	SHAL.W Rd	1					
	SHAL.W #2,Rd	1					
	SHAL.L ERd	1					
	SHAL.L #2,ERd	1					
SHAR	SHAR.B Rd	1					
	SHAR.B #2,Rd	1					
	SHAR.W Rd	1					
	SHAR.W #2,Rd	1					
	SHAR.L ERd	1					
	SHAR.L #2,ERd	1					
SHLL	SHLL.B Rd	1					
	SHLL.B #2,Rd	1					
	SHLL.W Rd	1					
	SHLL.W #2,Rd	1					
	SHLL.L ERd	1					
	SHLL.L #2,ERd	1					
SHLR	SHLR.B Rd	1					
	SHLR.B #2,Rd	1					
	SHLR.W Rd	1					
	SHLR.W #2,Rd	1					
	SHLR.L ERd	1					
	SHLR.L #2,ERd	1					
SLEEP	SLEEP	1					1

**Table 2-8 Number of Cycles in Instruction Execution (cont)**

Instruction	Mnemonic	Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
STC	STC.B CCR,Rd	1					
	STC.B EXR,Rd	1					
	STC.W CCR,@ERd	2				1	
	STC.W EXR,@ERd	2				1	
	STC.W CCR,@(d:16,ERd)	3				1	
	STC.W EXR,@(d:16,ERd)	3				1	
	STC.W CCR,@(d:32,ERd)	5				1	
	STC.W EXR,@(d:32,ERd)	5				1	
	STC.W CCR,@-ERd	2				1	1
	STC.W EXR,@-ERd	2				1	1
	STC.W CCR,@aa:16	3				1	
	STC.W EXR,@aa:16	3				1	
	STC.W CCR,@aa:32	4				1	
	STC.W EXR,@aa:32	4				1	
STM	STM.L (ERn-ERn+1),@-SP	2		4			1
	STM.L(ERn-ERn+2),@-SP	2		6			1
	STM.L(ERn-ERn+3),@-SP	2		8			1
STMAC*	STMAC MACH,ERd	1					
	STMAC MACL,ERd	1					
SUB	SUB.B Rs,Rd	1					
	SUB.W #xx:16,Rd	2					
	SUB.W Rs,Rd	1					
	SUB.L #xx:32,ERd	3					
	SUB.L ERs,ERd	1					
SUBS	SUBS #1/2/4,ERd	1					
SUBX	SUBX #xx:8,Rd	1					
	SUBX Rs,Rd	1					
TAS	TAS @ERd	2			2		
TRAPA	TRAPA #x:2	Normal	2	1	2/3 *1		2
		Advanced	2	2	2/3 *1		2
XOR	XOR.B #xx:8,Rd	1					
	XOR.B Rs,Rd	1					
	XOR.W #xx:16,Rd	2					
	XOR.W Rs,Rd	1					
	XOR.L #xx:32,ERd	3					
	XOR.L ERs,ERd	2					
XORC	XORC #xx:8,CCR	1					
XORC	XORC #xx:8,EXR	2					

Notes: \*These instructions are supported by the H8S/2600 CPU only.

1. 2 when EXR is invalid, 3 when EXR is valid.
2. 5 for concatenated execution, 4 otherwise.

## 2.7 Condition Code Modification

This section indicates the effect of each CPU instruction on the condition code. The notation used in the table is defined below.

$$m = \begin{cases} 31 & \text{for longword operands} \\ 15 & \text{for word operands} \\ 7 & \text{for byte operands} \end{cases}$$

Si	The i-th bit of the source operand
Di	The i-th bit of the destination operand
Ri	The i-th bit of the result
Dn	The specified bit in the destination operand
—	Not affected
↑	Modified according to the result of the instruction (see definition)
0	Always cleared to 0
1	Always set to 1
*	Undetermined (no guaranteed value)
Z'	Z flag before instruction execution
C'	C flag before instruction execution

**Table 2-7 Condition Code Modification**

Instruction	H	N	Z	V	C	Definition
ADD	↑	↑	↑	↑	↑	$H = Sm-4 \cdot Dm-4 + Dm-4 \cdot \overline{Rm-4} + Sm-4 \cdot \overline{Rm-4}$ $N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + Dm \cdot \overline{Rm} + Sm \cdot \overline{Rm}$
ADDS	—	—	—	—	—	
ADDX	↑	↑	↑	↑	↑	$H = Sm-4 \cdot Dm-4 + Dm-4 \cdot \overline{Rm-4} + Sm-4 \cdot \overline{Rm-4}$ $N = Rm$ $Z = Z' \cdot \overline{Rm} \cdot \dots \cdot \overline{R0}$ $V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + Dm \cdot \overline{Rm} + Sm \cdot \overline{Rm}$
AND	—	↑	↑	0	—	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
ANDC	↑	↑	↑	↑	↑	Stores the corresponding bits of the result. No flags change when the operand is EXR.
BAND	—	—	—	—	↑	$C = C' \cdot Dn$
Bcc	—	—	—	—	—	
BCLR	—	—	—	—	—	
BIAND	—	—	—	—	↑	$C = C' \cdot \overline{Dn}$
BILD	—	—	—	—	↑	$C = \overline{Dn}$
BIOR	—	—	—	—	↑	$C = C' + \overline{Dn}$
BIST	—	—	—	—	—	
BIXOR	—	—	—	—	↑	$C = C' \cdot Dn + \overline{C'} \cdot \overline{Dn}$
BLD	—	—	—	—	↑	$C = Dn$
BNOT	—	—	—	—	—	
BOR	—	—	—	—	↑	$C = C' + Dn$
BSET	—	—	—	—	—	
BSR	—	—	—	—	—	
BST	—	—	—	—	—	
BTST	—	—	↑	—	—	$Z = \overline{Dn}$
BXOR	—	—	—	—	↑	$C = C' \cdot \overline{Dn} + \overline{C'} \cdot Dn$
CLRMAC*	—	—	—	—	—	
CMP	↑	↑	↑	↑	↑	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$ $N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$

**Table 2-7 Condition Code Modification (cont)**

Instruction	H	N	Z	V	C	Definition
DAA	*	↑	↑	*	↑	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$ C: decimal arithmetic carry
DAS	*	↑	↑	*	↑	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$ C: decimal arithmetic borrow
DEC	—	↑	↑	↑	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$ $V = D_m \cdot \overline{R_m}$
DIVXS	—	↑	↑	—	—	$N = S_m \cdot \overline{D_m} + \overline{S_m} \cdot D_m$ $Z = \overline{S_m} \cdot \overline{S_{m-1}} \cdot \dots \cdot \overline{S_0}$
DIVXU	—	↑	↑	—	—	$N = S_m$ $Z = \overline{S_m} \cdot \overline{S_{m-1}} \cdot \dots \cdot \overline{S_0}$
EEPMOV	—	—	—	—	—	
EXTS	—	↑	↑	0	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$
EXTU	—	0	↑	0	—	$Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$
INC	—	↑	↑	↑	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$ $V = \overline{D_m} \cdot R_m$
JMP	—	—	—	—	—	
JSR	—	—	—	—	—	
LDC	↑	↑	↑	↑	↑	Stores the corresponding bits of the result. No flags change when the operand is EXR.
LDM	—	—	—	—	—	
LDMAC*	—	—	—	—	—	
MAC*	—	—	—	—	—	
MOV	—	↑	↑	0	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$
MOVFPPE	—	↑	↑	0	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$
MOVTPPE	—	↑	↑	0	—	$N = R_m$ $Z = \overline{R_m} \cdot \overline{R_{m-1}} \cdot \dots \cdot \overline{R_0}$
MULXS	—	↑	↑	—	—	$N = R_{2m}$ $Z = \overline{R_{2m}} \cdot \overline{R_{2m-1}} \cdot \dots \cdot \overline{R_0}$

**Table 2-7 Condition Code Modification (cont)**

Instruction	H	N	Z	V	C	Definition
MULXU	—	—	—	—	—	
NEG	↑	↑	↑	↑	↑	$H = Dm-4 + Rm-4$ $N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $V = Dm \cdot Rm$ $C = Dm + Rm$
NOP	—	—	—	—	—	
NOT	—	↑	↑	0	—	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
OR	—	↑	↑	0	—	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
ORC	↑	↑	↑	↑	↑	Stores the corresponding bits of the result. No flags change when the operand is EXR.
POP	—	↑	↑	0	—	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
PUSH	—	↑	↑	0	—	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
ROTL	—	↑	↑	0	↑	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $C = Dm$ (1-bit shift) or $C = Dm-1$ (2-bit shift)
ROTR	—	↑	↑	0	↑	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $C = D0$ (1-bit shift) or $C = D1$ (2-bit shift)
ROTXL	—	↑	↑	0	↑	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $C = Dm$ (1-bit shift) or $C = Dm-1$ (2-bit shift)
ROTXR	—	↑	↑	0	↑	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $C = D0$ (1-bit shift) or $C = D1$ (2-bit shift)
RTE	↑	↑	↑	↑	↑	Stores the corresponding bits of the result.
RTS	—	—	—	—	—	
SHAL	—	↑	↑	↑	↑	$N = Rm$ $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $V = Dm \cdot Dm-1 + \overline{Dm} \cdot \overline{Dm-1}$ (1-bit shift) $V = \overline{Dm} \cdot \overline{Dm-1} \cdot \overline{Dm-2} + \overline{Dm} \cdot \overline{Dm-1} \cdot \overline{Dm-2}$ (2-bit shift) $C = Dm$ (1-bit shift) or $C = Dm-1$ (2-bit shift)

**Table 2-7 Condition Code Modification (cont)**

Instruction	H	N	Z	V	C	Definition
SHAR	—	↓	↓	0	↓	N = Rm $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ C = D0 (1-bit shift) or C = D1 (2-bit shift)
SHLL	—	↓	↓	0	↓	N = Rm $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ C = Dm (1-bit shift) or C = Dm-1 (2-bit shift)
SHLR	—	0	↓	0	↓	N = Rm $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ C = D0 (1-bit shift) or C = D1 (2-bit shift)
SLEEP	—	—	—	—	—	
STC	—	—	—	—	—	
STM	—	—	—	—	—	
STMAC*	—	↓	↓	↓	—	N = 1 if MAC instruction resulted in negative value in MAC register Z = 1 if MAC instruction resulted in zero value in MAC register V = 1 if MAC instruction resulted in overflow
SUB	↓	↓	↓	↓	↓	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$ N = Rm $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$ $V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$
SUBS	—	—	—	—	—	
SUBX	↓	↓	↓	↓	↓	$H = Sm-4 \cdot \overline{Dm-4} + \overline{Dm-4} \cdot Rm-4 + Sm-4 \cdot Rm-4$ N = Rm $Z = Z' \cdot \overline{Rm} \cdot \dots \cdot \overline{R0}$ $V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + \overline{Dm} \cdot Rm + Sm \cdot Rm$
TAS	—	↓	↓	0	—	N = Dm $Z = \overline{Dm} \cdot \overline{Dm-1} \cdot \dots \cdot \overline{D0}$
TRAPA	—	—	—	—	—	
XOR	—	↓	↓	0	—	N = Rm $Z = \overline{Rm} \cdot \overline{Rm-1} \cdot \dots \cdot \overline{R0}$
XORC	↓	↓	↓	↓	↓	Stores the corresponding bits of the result. No flags change when the operand is EXR.

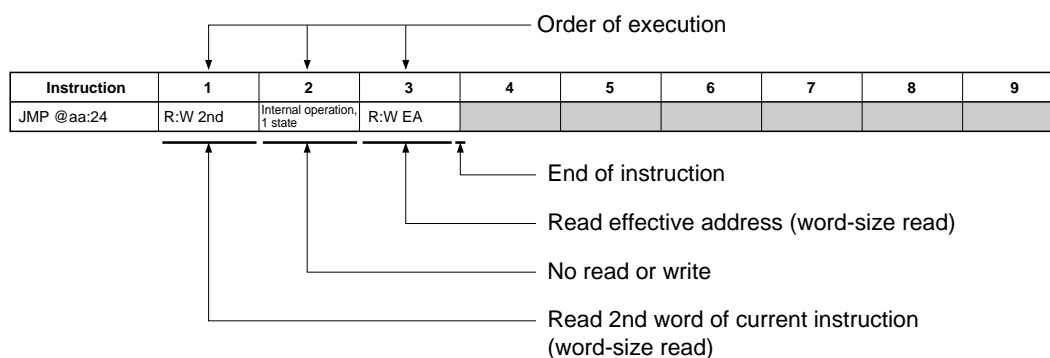
Note: \*These instructions are supported by the H8S/2600 CPU only.



## 2.8 Bus States During Instruction Execution

Table 2-8 indicates the types of cycles that occur during instruction execution by the CPU. See table 2-5 for the number of states per cycle.

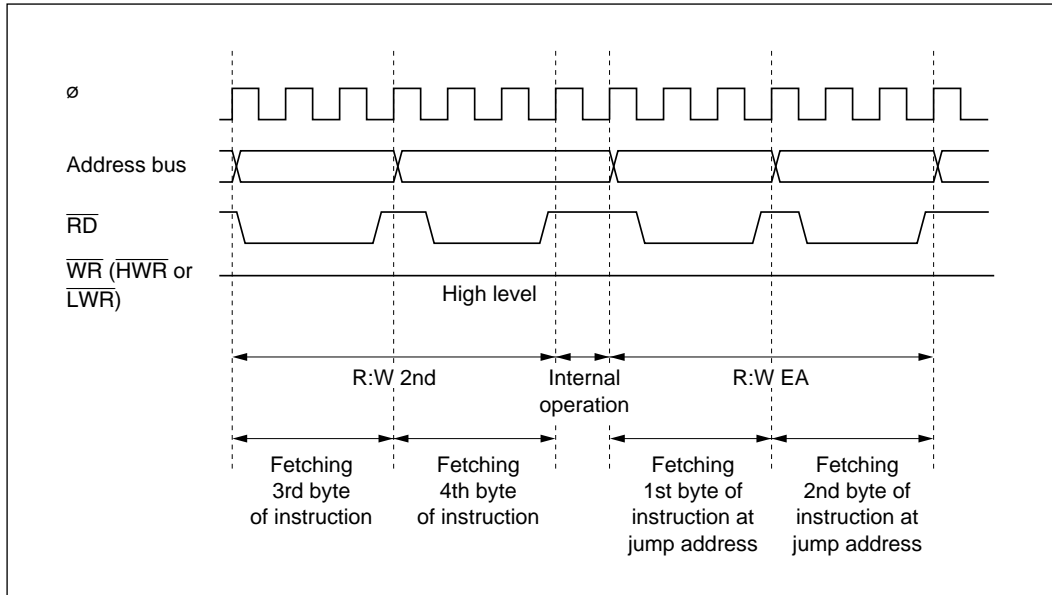
### How to Read the Table:



### Legend

R:B	Byte-size read
R:W	Word-size read
W:B	Byte-size write
W:W	Word-size write
2nd	Address of 2nd word (3rd and 4th bytes)
3rd	Address of 3rd word (5th and 6th bytes)
4th	Address of 4th word (7th and 8th bytes)
5th	Address of 5th word (9th and 10th bytes)
NEXT	Address of next instruction
EA	Effective address
VEC	Vector address

Figure 2-1 shows timing waveforms for the address bus and the  $\overline{RD}$  and  $\overline{WR}$  ( $\overline{HWR}$  or  $\overline{LWR}$ ) signals during execution of the above instruction with an 8-bit bus, using three-state access with no wait states.



**Figure 2-1 Address Bus,  $\overline{RD}$ , and  $\overline{WR}$  ( $\overline{HWR}$  or  $\overline{LWR}$ ) Timing (8-Bit Bus, Three-State Access, No Wait States)**

**Table 2-8 Instruction Execution Cycles**

Instruction	1	2	3	4	5	6	7	8	9
ADD.B #xx:8,Rd	R:W NEXT								
ADD.B Rs,Rd	R:W NEXT								
ADD.W #xx:16,Rd	R:W 2nd	R:W NEXT							
ADD.W Rs,Rd	R:W NEXT								
ADD.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
ADD.L ERs,ERd	R:W NEXT								
ADDS #1/2/4,ERd	R:W NEXT								
ADDX #xx:8,Rd	R:W NEXT								
ADDX Rs,Rd	R:W NEXT								
AND.B #xx:8,Rd	R:W NEXT								
AND.B Rs,Rd	R:W NEXT								
AND.W #xx:16,Rd	R:W 2nd	R:W NEXT							
AND.W Rs,Rd	R:W NEXT								
AND.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
AND.L ERs,ERd	R:W 2nd	R:W NEXT							
ANDC #xx:8,CCR	R:W NEXT								
ANDC #xx:8,EXR	R:W 2nd	R:W NEXT							
BAND #xx:3,Rd	R:W NEXT								
BAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BAND #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BAND #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BRA d:8 (BT d:8)	R:W NEXT	R:W EA							
BRN d:8 (BF d:8)	R:W NEXT	R:W EA							
BHI d:8	R:W NEXT	R:W EA							
BLS d:8	R:W NEXT	R:W EA							
BCC d:8 (BHS d:8)	R:W NEXT	R:W EA							
BCS d:8 (BLO d:8)	R:W NEXT	R:W EA							
BNE d:8	R:W NEXT	R:W EA							
BEQ d:8	R:W NEXT	R:W EA							
BVC d:8	R:W NEXT	R:W EA							
BVS d:8	R:W NEXT	R:W EA							
BPL d:8	R:W NEXT	R:W EA							
BMI d:8	R:W NEXT	R:W EA							
BGE d:8	R:W NEXT	R:W EA							
BLT d:8	R:W NEXT	R:W EA							
BGT d:8	R:W NEXT	R:W EA							

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
BLE d:8	R:W NEXT	R:W EA							
BRA d:16 (BT d:16)	R:W 2nd	Internal operation, 1 state	R:W EA						
BRN d:16 (BF d:16)	R:W 2nd	Internal operation, 1 state	R:W EA						
BHI d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BLS d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BCC d:16 (BHS d:16)	R:W 2nd	Internal operation, 1 state	R:W EA						
BCS d:16 (BLO d:16)	R:W 2nd	Internal operation, 1 state	R:W EA						
BNE d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BEQ d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BVC d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BVS d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BPL d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BMI d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BGE d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BLT d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BGT d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BLE d:16	R:W 2nd	Internal operation, 1 state	R:W EA						
BCLR #xx:3,Rd	R:W NEXT								
BCLR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
BCLR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BCLR Rn,Rd	R:W NEXT								
BCLR Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BCLR Rn,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BCLR Rn,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BIAND #xx:3,Rd	R:W NEXT								
BIAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIAND #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIAND #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BILD #xx:3,Rd	R:W NEXT								
BILD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BILD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BILD #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BILD #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BIOR #xx:3,Rd	R:W NEXT								
BIOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIOR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BIST #xx:3,Rd	R:W NEXT								
BIST #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BIST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BIST #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BIST #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BIXOR #xx:3,Rd	R:W NEXT								
BIXOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BIXOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BIXOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BIXOR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BLD #xx:3,Rd	R:W NEXT								
BLD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT						
BLD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BLD #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BLD #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BNOT #xx:3,Rd	R:W NEXT								

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction		1	2	3	4	5	6	7	8	9
BNOT #xx:3, @ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT #xx:3, @aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT #xx:3, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BNOT #xx:3, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BNOT Rn,Rd		R:W NEXT								
BNOT Rn, @ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT Rn, @aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BNOT Rn, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BNOT Rn, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BOR #xx:3, Rd		R:W NEXT								
BOR #xx:3, @ERd		R:W 2nd	R:B EA	R:W NEXT						
BOR #xx:3, @aa:8		R:W 2nd	R:B EA	R:W NEXT						
BOR #xx:3, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BOR #xx:3, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BSET #xx:3, Rd		R:W NEXT								
BSET #xx:3, @ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET #xx:3, @aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET #xx:3, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BSET #xx:3, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BSET Rn,Rd		R:W NEXT								
BSET Rn, @ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET Rn, @aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BSET Rn, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BSET Rn, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BSR d:8	Normal	R:W NEXT	R:W EA	W:W stack						
	Advanced	R:W NEXT	R:W EA	W:W stack (H)	W:W stack (L)					
BSR d:16	Normal	R:W 2nd	Internal operation, 1 state	R:W EA	W:W stack					
	Advanced	R:W 2nd	Internal operation, 1 state	R:W EA	W:W stack (H)	W:W stack (L)				
BST #xx:3, Rd		R:W NEXT								
BST #xx:3, @ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BST #xx:3, @aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA					
BST #xx:3, @aa:16		R:W 2nd	R:W 3rd	R:B EA	R:W NEXT	W:B EA				
BST #xx:3, @aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT	W:B EA			
BTST #xx:3, Rd		R:W NEXT								
BTST #xx:3, @ERd		R:W 2nd	R:B EA	R:W NEXT						

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
BTST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BTST #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BTST #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BTST Rn,Rd	R:W NEXT								
BTST Rn,ERd	R:W 2nd	R:B EA	R:W NEXT						
BTST Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BTST Rn,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BTST Rn,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
BXOR #xx:3,Rd	R:W NEXT								
BXOR #xx:3,ERd	R:W 2nd	R:B EA	R:W NEXT						
BXOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT						
BXOR #xx:3,@aa:16	R:W 2nd	R:W 3rd	R:B EA	R:W NEXT					
BXOR #xx:3,@aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:B EA	R:W NEXT				
CLRMAC*	R:W NEXT	Internal operation, 1 state							
CMP.B #xx:8,Rd	R:W NEXT								
CMP.B Rs,Rd	R:W NEXT								
CMP.W #xx:16,Rd	R:W 2nd	R:W NEXT							
CMP.W Rs,Rd	R:W NEXT								
CMPL #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
CMPL ERs,ERd	R:W NEXT								
DAA Rd	R:W NEXT								
DAS Rd	R:W NEXT								
DEC.B Rd	R:W NEXT								
DEC.W #1/2,Rd	R:W NEXT								
DEC.L #1/2,ERd	R:W NEXT								
DIVXS.B Rs,Rd	R:W 2nd	R:W NEXT	Internal operation, 11 states						
DIVXS.W Rs,ERd	R:W 2nd	R:W NEXT	Internal operation, 19 states						
DIVXU.B Rs,Rd	R:W NEXT	Internal operation, 11 states							
DIVXU.W Rs,ERd	R:W NEXT	Internal operation, 19 states							
EEPMOV.B	R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT			
EEPMOV.W	R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT			
EXTS.W Rd	R:W NEXT			← Repeated n times*3 →					
EXTS.L ERd	R:W NEXT								
EXTU.W Rd	R:W NEXT								
EXTU.L ERd	R:W NEXT								
INC.B Rd	R:W NEXT								

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction		1	2	3	4	5	6	7	8	9
INC.W #1/2,Rd		R:W NEXT								
INC.L #1/2,ERd		R:W NEXT								
JMP @ERn		R:W NEXT	R:W EA							
JMP @aa:24		R:W 2nd	Internal operation, 1 state	R:W EA						
JMP @@aa:8	Normal	R:W NEXT	R:W aa:8	Internal operation, 1 state	R:W EA					
	Advanced	R:W NEXT	R:W aa:8	R:W aa:8	Internal operation, 1 state	R:W EA				
JSR @ERn	Normal	R:W NEXT	R:W EA	W:W stack						
	Advanced	R:W NEXT	R:W EA	W:W stack (H)	W:W stack (L)					
JSR @aa:24	Normal	R:W 2nd	Internal operation, 1 state	R:W EA	W:W stack					
	Advanced	R:W 2nd	Internal operation, 1 state	R:W EA	W:W stack (H)	W:W stack (L)				
JSR @@aa:8	Normal	R:W NEXT	R:W aa:8	W:W stack	R:W EA					
	Advanced	R:W NEXT	R:W aa:8	R:W aa:8	W:W stack (H)	W:W stack (L)	R:W EA			
LDC #xx:8,CCR		R:W NEXT								
LDC #xx:8,EXR		R:W 2nd	R:W NEXT							
LDC Rs,CCR		R:W NEXT								
LDC Rs,EXR		R:W NEXT								
LDC @ERs,CCR		R:W 2nd	R:W NEXT	R:W EA						
LDC @ERs,EXR		R:W 2nd	R:W NEXT	R:W EA						
LDC @(d:16,ERs),CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @(d:16,ERs),EXR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @(d:32,ERs),CCR		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA			
LDC @(d:32,ERs),EXR		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA			
LDC @ERs+,CCR		R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W EA					
LDC @ERs+,EXR		R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W EA					
LDC @aa:16,CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @aa:16,EXR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA					
LDC @aa:32,CCR		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA				
LDC @aa:32,EXR		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA				
LDML @SP+,(ERn-ERn+1)		R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W stack (H) * <sup>3</sup>	R:W stack (L) * <sup>3</sup>				



**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
LDM.L @SP+,(ERn-ERn+2)	R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W stack (H) *3	R:W stack (L) *3				
LDM.L @SP+,(ERn-ERn+3)	R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W stack (H) *3	R:W stack (L) *3				
LDMAC ERs,MACH*	R:W NEXT	Internal operation, 1 state		← Repeated n times*3 →					
LDMAC ERs,MACL*	R:W NEXT	Internal operation, 1 state							
MAC @ERn+,@ERm+*	R:W 2nd	R:W NEXT	R:W EAn	R:W EAm					
MOV.B #xx:8,Rd	R:W NEXT								
MOV.B Rs,Rd	R:W NEXT								
MOV.B @ERs,Rd	R:W NEXT	R:B EA							
MOV.B @(d:16,ERs),Rd	R:W 2nd	R:W NEXT	R:B EA						
MOV.B @(d:32,ERs),Rd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:B EA				
MOV.B @ERs+,Rd	R:W NEXT	Internal operation, 1 state	R:B EA						
MOV.B @aa:8,Rd	R:W NEXT	R:B EA							
MOV.B @aa:16,Rd	R:W 2nd	R:W NEXT	R:B EA						
MOV.B @aa:32,Rd	R:W 2nd	R:W 3rd	R:W NEXT	R:B EA					
MOV.B Rs,@ERd	R:W NEXT	W:B EA							
MOV.B Rs,@(d:16,ERd)	R:W 2nd	R:W NEXT	W:B EA						
MOV.B Rs,@(d:32,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:B EA				
MOV.B Rs,@-ERd	R:W NEXT	Internal operation, 1 state	W:B EA						
MOV.B Rs,@aa:8	R:W NEXT	W:B EA							
MOV.B Rs,@aa:16	R:W 2nd	R:W NEXT	W:B EA						
MOV.B Rs,@aa:32	R:W 2nd	R:W 3rd	R:W NEXT	W:B EA					
MOV.W #xx:16,Rd	R:W 2nd	R:W NEXT							
MOV.W Rs,Rd	R:W NEXT								
MOV.W @ERs,Rd	R:W NEXT	R:W EA							
MOV.W @(d:16,ERs),Rd	R:W 2nd	R:W NEXT	R:W EA						
MOV.W @(d:32,ERs),Rd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA				
MOV.W @ERs+,Rd	R:W NEXT	Internal operation, 1 state	R:W EA						
MOV.W @aa:16,Rd	R:W 2nd	R:W NEXT	R:W EA						
MOV.W @aa:32,Rd	R:W 2nd	R:W 3rd	R:W NEXT	R:B EA					
MOV.W Rs,@ERd	R:W NEXT	W:W EA							

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
MOV.W Rs, @(d:16,ERd)	R:W 2nd	R:W NEXT	W:W EA						
MOV.W Rs, @(d:32,ERd)	R:W 2nd	R:W 3rd	R:E 4th	R:W NEXT	W:W EA				
MOV.W Rs, @-ERd	R:W NEXT	Internal operation, 1 state	W:W EA						
MOV.W Rs, @aa:16	R:W 2nd	R:W NEXT	W:W EA						
MOV.W Rs, @aa:32	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
MOV.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
MOV.L ERs,ERd	R:W NEXT								
MOV.L @ERs,ERd	R:W 2nd	R:W NEXT	R:W EA	R:W EA+2					
MOV.L @(d:16,ERs),ERd	R:W 2nd	R:W 3rd	R:W NEXT	R:W EA	R:W EA+2				
MOV.L @(d:32,ERs),ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA	R:W EA+2		
MOV.L @ERs+,ERd	R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W EA	R:W EA+2				
MOV.L @aa:16,ERd	R:W 2nd	R:W 3rd	R:W NEXT	R:W EA	R:W EA+2				
MOV.L @aa:32,ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA	R:W EA+2			
MOV.L ERs, @ERd	R:W 2nd	R:W NEXT	W:W EA	W:W EA+2					
MOV.L ERs, @(d:16,ERd)	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA	W:W EA+2				
MOV.L ERs, @(d:32,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA	W:W EA+2		
MOV.L ERs, @-ERd	R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W EA	W:W EA+2				
MOV.L ERs, @aa:16	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA	W:W EA+2				
MOV.L ERs, @aa:32	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA	W:W EA+2			
MOVFPPE @aa:16,Rd	R:W 2nd	Internal operation, 1 state	R:W *4 EA						
MOVTPPE Rs, @aa:16	R:W 2nd	Internal operation, 1 state	W:B *4 EA						
MULXS.B Rs,Rd H8S/2600	R:W 2nd	R:W NEXT	Internal operation, 2 states						
H8S/2000	R:W 2nd	R:W NEXT	Internal operation, 11 states						
MULXS.W Rs,ERd H8S/2600	R:W 2nd	R:W NEXT	Internal operation, 3 states						
H8S/2000	R:W 2nd	R:W NEXT	Internal operation, 19 states						
MULXU.B Rs,Rd H8S/2600	R:W NEXT	Internal operation, 2 states							
H8S/2000	R:W NEXT	Internal operation, 11 states							
MULXU.W Rs,ERd H8S/2600	R:W NEXT	Internal operation, 3 states							
H8S/2000	R:W NEXT	Internal operation, 19 states							
NEG.B Rd	R:W NEXT								
NEG.W Rd	R:W NEXT								
NEG.L ERd	R:W NEXT								
NOP	R:W NEXT								
NOT.B Rd	R:W NEXT								
NOT.W Rd	R:W NEXT								
NOT.L ERd	R:W NEXT								

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction	1	2	3	4	5	6	7	8	9
OR.B #xx:8,Rd	R:W NEXT								
OR.B Rs,Rd	R:W NEXT								
OR.W #xx:16,Rd	R:W 2nd	R:W NEXT							
OR.W Rs,Rd	R:W NEXT								
OR.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT						
OR.L ERs,ERd	R:W 2nd	R:W NEXT							
ORC #xx:8,CCR	R:W NEXT								
ORC #xx:8,EXR	R:W 2nd	R:W NEXT							
POP.W Rn	R:W NEXT	Internal operation, 1 state	R:W EA						
POP.L ERn	R:W 2nd	R:W NEXT	Internal operation, 1 state	R:W EA	R:W EA+2				
PUSH.W Rn	R:W NEXT	Internal operation, 1 state	W:W EA						
PUSH.L ERn	R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W EA	W:W EA+2				
ROTL.B Rd	R:W NEXT								
ROTL.B #2,Rd	R:W NEXT								
ROTL.W Rd	R:W NEXT								
ROTL.W #2,Rd	R:W NEXT								
ROTL.L ERd	R:W NEXT								
ROTL.L #2,ERd	R:W NEXT								
ROTR.B Rd	R:W NEXT								
ROTR.B #2,Rd	R:W NEXT								
ROTR.W Rd	R:W NEXT								
ROTR.W #2,Rd	R:W NEXT								
ROTR.L ERd	R:W NEXT								
ROTR.L #2,ERd	R:W NEXT								
ROTXL.B Rd	R:W NEXT								
ROTXL.B #2,Rd	R:W NEXT								
ROTXL.W Rd	R:W NEXT								
ROTXL.W #2,Rd	R:W NEXT								
ROTXL.L ERd	R:W NEXT								
ROTXL.L #2,ERd	R:W NEXT								

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction		1	2	3	4	5	6	7	8	9
ROTXR.B Rd		R:W NEXT								
ROTXR.B #2,Rd		R:W NEXT								
ROTXR.W Rd		R:W NEXT								
ROTXR.W #2,Rd		R:W NEXT								
ROTXR.L ERd		R:W NEXT								
ROTXR.L #2,ERd		R:W NEXT								
RTE		R:W NEXT	R:W stack (EXR)	R:W stack (H)	R:W stack (L)	Internal operation, 1 state	R:W *5			
RTS	Normal	R:W NEXT	R:W stack	Internal operation, 1 state	R:W *5					
	Advanced	R:W NEXT	R:W stack (H)	R:W stack (L)	Internal operation, 1 state	R:W *5				
SHAL.B Rd		R:W NEXT								
SHAL.B #2,Rd		R:W NEXT								
SHAL.W Rd		R:W NEXT								
SHAL.W #2,Rd		R:W NEXT								
SHAL.L ERd		R:W NEXT								
SHAL.L #2,ERd		R:W NEXT								
SHAR.B Rd		R:W NEXT								
SHAR.B #2,Rd		R:W NEXT								
SHAR.W Rd		R:W NEXT								
SHAR.W #2,Rd		R:W NEXT								
SHAR.L ERd		R:W NEXT								
SHAR.L #2,ERd		R:W NEXT								
SHLL.B Rd		R:W NEXT								
SHLL.B #2,Rd		R:W NEXT								
SHLL.W Rd		R:W NEXT								
SHLL.W #2,Rd		R:W NEXT								
SHLL.L ERd		R:W NEXT								
SHLL.L #2,ERd		R:W NEXT								
SHLR.B Rd		R:W NEXT								
SHLR.B #2,Rd		R:W NEXT								
SHLR.W Rd		R:W NEXT								
SHLR.W #2,Rd		R:W NEXT								
SHLR.L ERd		R:W NEXT								
SHLR.L #2,ERd		R:W NEXT								

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction		1	2	3	4	5	6	7	8	9
SLEEP		R:W NEXT	Internal operation, 1 state							
STC CCR,Rd		R:W NEXT								
STC EXR,Rd		R:W NEXT								
STC CCR,@ERd		R:W 2nd	R:W NEXT	W:W EA						
STC EXR,@ERd		R:W 2nd	R:W NEXT	W:W EA						
STC CCR,@(d:16,ERd)		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC EXR,@(d:16,ERd)		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC CCR,@(d:32,ERd)		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA			
STC EXR,@(d:32,ERd)		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA			
STC CCR,@-ERd		R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W EA					
STC EXR,@-ERd		R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W EA					
STC CCR,@aa:16		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC EXR,@aa:16		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA					
STC CCR,@aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA				
STC EXR,@aa:32		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA				
STM.L(ERn-ERn+1),@-SP		R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W stack (H) *3	W:W stack (L) *3				
STM.L(ERn-ERn+2),@-SP		R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W stack (H) *3	W:W stack (L) *3				
STM.L(ERn-ERn+3),@-SP		R:W 2nd	R:W NEXT	Internal operation, 1 state	W:W stack (H) *3	W:W stack (L) *3				
STMAC MACH,ERd*		R:W NEXT			← Repeated n times*3 →					
STMAC MACL,ERd*		R:W NEXT								
SUB.B Rs,Rd		R:W NEXT								
SUB.W #xx:16,Rd		R:W 2nd	R:W NEXT							
SUB.W Rs,Rd		R:W NEXT								
SUB.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT						
SUB.L ERs,ERd		R:W NEXT								
SUBS #1/2/4,ERd		R:W NEXT								
SUBX #xx:8,Rd		R:W NEXT								
SUBX Rs,Rd		R:W NEXT								
TAS @ERd		R:W 2nd	R:W NEXT	R:B EA	W:B EA					
TRAPA #x:2	Normal	R:W NEXT	Internal operation, 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR)	R:W VEC	Internal operation, 1 state	R:W *8	

**Table 2-8 Instruction Execution Cycles (cont)**

Instruction		1	2	3	4	5	6	7	8	9
TRAPA #x:2 (cont)	Advanced	R:W NEXT	Internal operation, 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR)	R:W VEC	R:W VEC+2	Internal operation, 1 state	R:W *8
XOR.B #xx8,Rd		R:W NEXT								
XOR.B Rs,Rd		R:W NEXT								
XOR.W #xx:16,Rd		R:W 2nd	R:W NEXT							
XOR.W Rs,Rd		R:W NEXT								
XOR.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT						
XOR.L ERs,ERd		R:W 2nd	R:W NEXT							
XORC #xx:8,CCR		R:W NEXT								
XORC #xx:8,EXR		R:W 2nd	R:W NEXT							
Reset exception handling	Normal	R:W VEC	Internal operation, 1 state	R:W *6						
	Advanced	R:W VEC	R:W VEC+2	Internal operation, 1 state	R:W *6					
Interrupt exception handling	Normal	R:W *7	Internal operation, 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR)	R:W VEC	Internal operation, 1 state	R:W *8	
	Advanced	R:W *7	Internal operation, 1 state	W:W stack (L)	W:W stack (H)	W:W stack (EXR)	R:W VEC	R:W VEC+2	Internal operation, 1 state	R:W *8

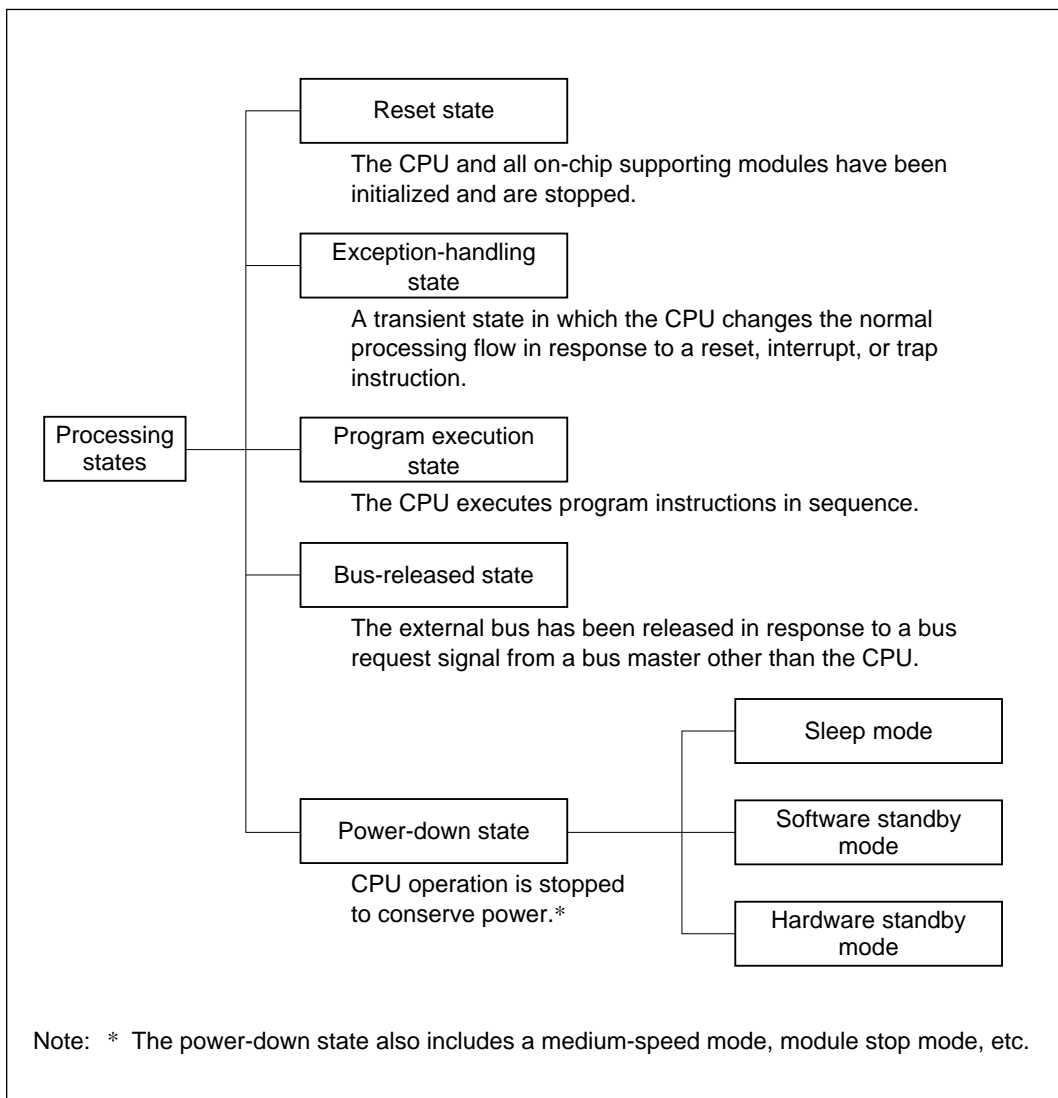
Notes: \*These instructions are supported by the H8S/2600 CPU only.

1. EAs is the contents of ER5. EAd is the contents of ER6.
2. EAs is the contents of ER5. EAd is the contents of ER6. Both registers are incremented by 1 after execution of the instruction. n is the initial value of R4L or R4. If n = 0, these bus cycles are not executed.
3. Repeated two times to save or restore two registers, three times for three registers, or four times for four registers.
4. For the number of states required for byte-size read or write, refer to the relevant microcontroller hardware manual.
5. Start address after return.
6. Start address of the program.
7. Prefetch address, equal to two plus the PC value pushed onto the stack. In recovery from sleep mode or software standby mode the read operation is replaced by an internal operation.
8. Start address of the interrupt-handling routine.

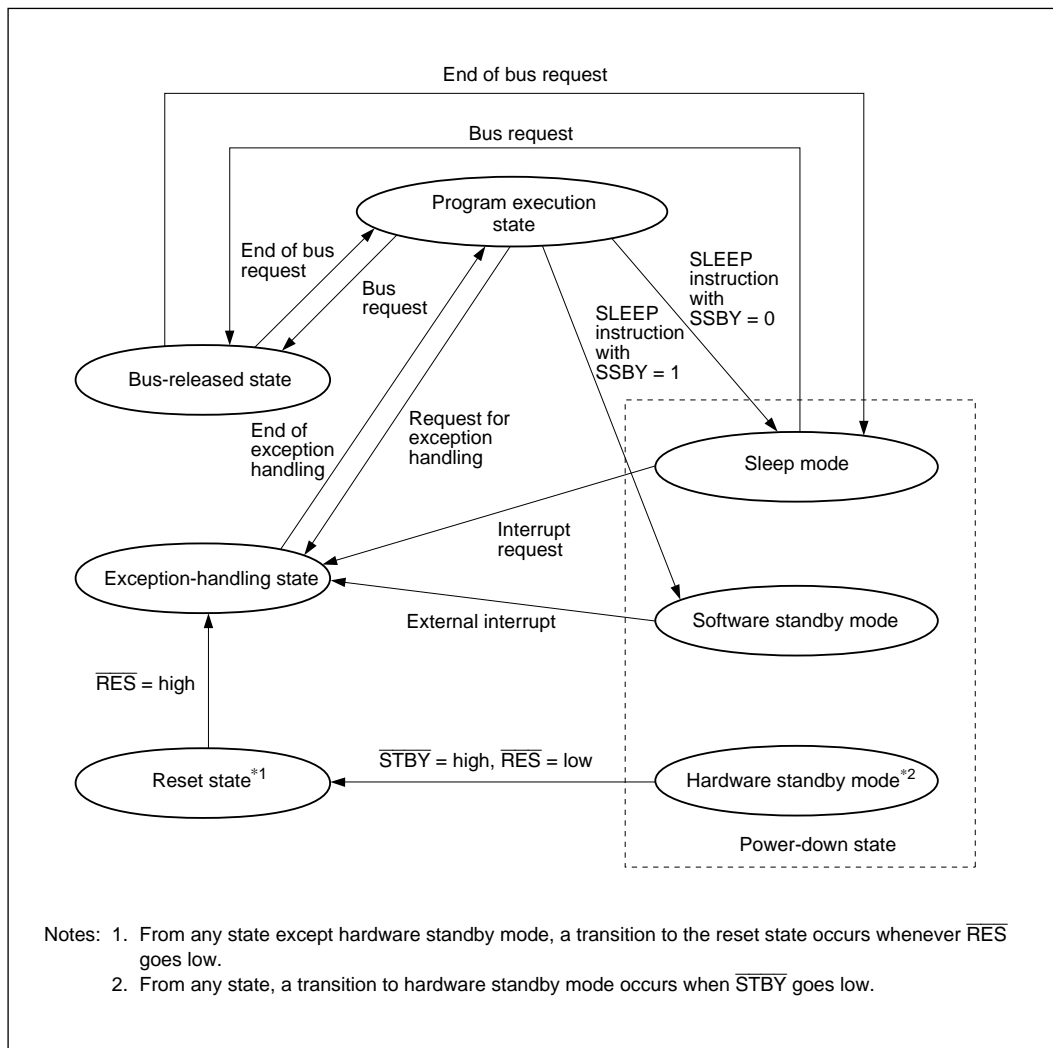
## Section 3 Processing States

### 3.1 Overview

The CPU has five main processing states: the reset state, exception handling state, program execution state, bus-released state, and power-down state. Figure 3-1 shows a diagram of the processing states. Figure 3-2 indicates the state transitions.



**Figure 3-1 Processing States**



**Figure 3-2 State Transitions**

## 3.2 Reset State

When the  $\overline{RES}$  input goes low all current processing stops and the CPU enters the reset state. Reset exception handling starts when the  $\overline{RES}$  signal changes from low to high.

The reset state can also be entered by a watchdog timer overflow. For details, refer to the relevant microcontroller hardware manual.



### 3.3 Exception-Handling State

The exception-handling state is a transient state that occurs when the CPU alters the normal processing flow due to a reset, interrupt, or trap instruction. The CPU fetches a start address (vector) from the exception vector table and branches to that address.

#### 3.3.1 Types of Exception Handling and Their Priority

Exception handling is performed for traces, resets, interrupts, and trap instructions. Table 3-1 indicates the types of exception handling and their priority. Trap instruction exception handling is always accepted, in the program execution state.

Exception handling and the stack structure differ according to the interrupt control mode set in SYSCR.

**Table 3-1 Exception Handling Types and Priority**

Priority	Type of Exception	Detection Timing	Start of Exception Handling
High ↑	Reset	Synchronized with clock	Exception handling starts immediately when RES changes from low to high
	Trace	End of instruction execution or end of exception-handling sequence* <sup>1</sup>	When the trace (T) bit is set to 1, the trace starts at the end of the current instruction or current exception-handling sequence
	Interrupt	End of instruction execution or end of exception-handling sequence* <sup>2</sup>	When an interrupt is requested, exception handling starts at the end of the current instruction or current exception-handling sequence
	Trap instruction	When TRAPA instruction is executed	Exception handling starts when a trap (TRAPA) instruction is executed* <sup>3</sup>
Low			

Notes: 1. Traces are enabled only in interrupt control modes 2 and 3. Trace exception-handling is not executed at the end of the RTE instruction.  
2. Interrupts are not detected at the end of the ANDC, ORC, XORC, and LDC instructions, or immediately after reset exception handling.  
3. Trap instruction exception handling is always accepted, in the program execution state.

For details on interrupt control modes, exception sources, and exception handling, refer to the relevant microcontroller hardware manual.

### 3.3.2 Reset Exception Handling

After the  $\overline{\text{RES}}$  pin has gone low and the reset state has been entered, reset exception handling starts when  $\overline{\text{RES}}$  goes high again. When reset exception handling starts the CPU fetches a start address (vector) from the exception vector table and starts program execution from that address. All interrupts, including NMI, are disabled during reset exception handling and after it ends.

### 3.3.3 Trace

Traces are enabled only in interrupt control modes 2 and 3. Trace mode is entered when the T bit of EXR is set to 1. When trace mode is established, trace exception handling starts at the end of each instruction.

At the end of a trace exception-handling sequence, the T bit of EXR is cleared to 0 and trace mode is cleared. Interrupt masks are not affected.

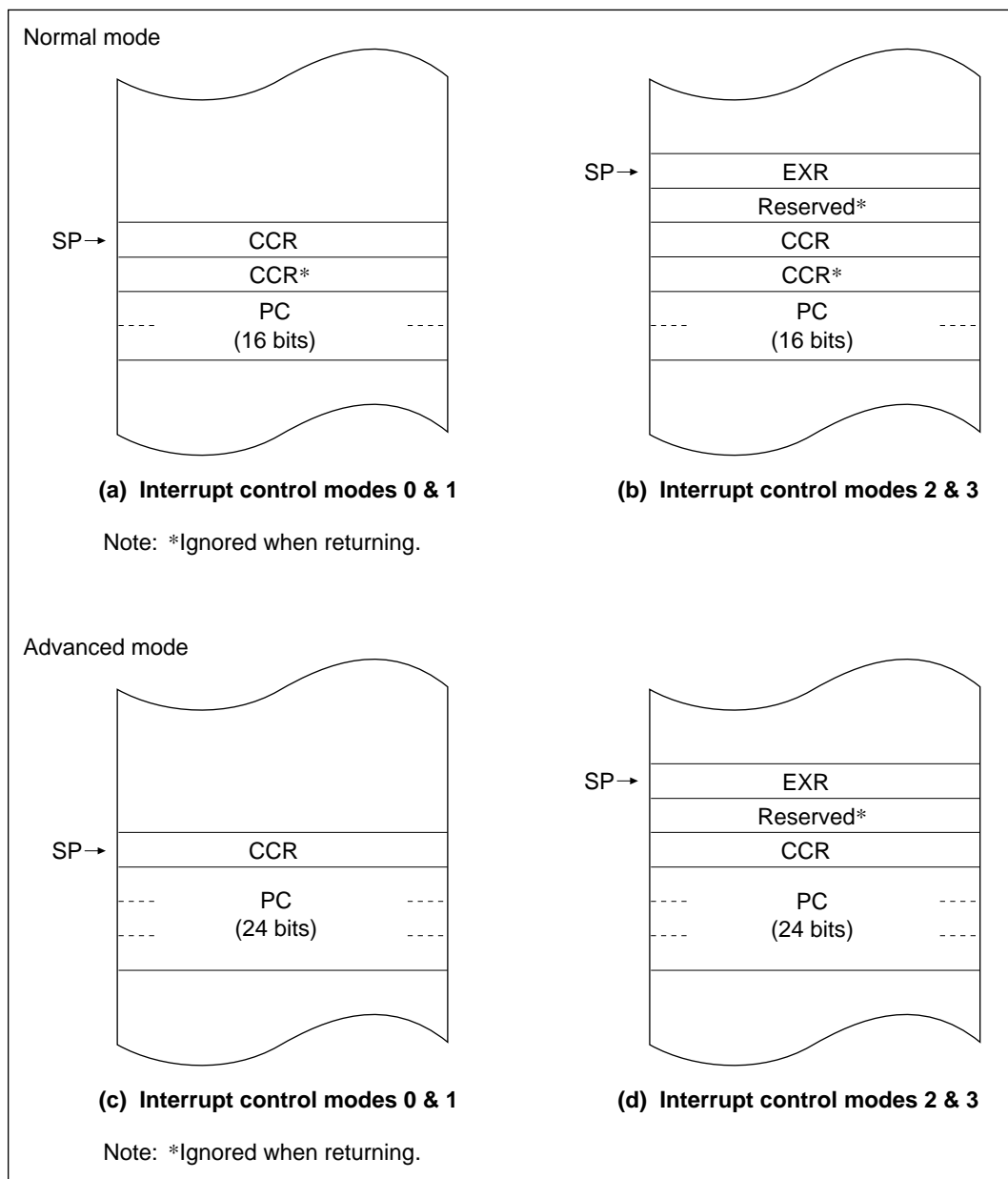
The T bit saved on the stack retains its value of 1, and when the RTE instruction is executed to return from the trace exception-handling routine, trace mode is entered again. Trace exception-handling is not executed at the end of the RTE instruction.

Trace mode is not entered in interrupt control modes 0 and 1, regardless of the state of the T bit.

### 3.3.4 Interrupt Exception Handling and Trap Instruction Exception Handling

When interrupt or trap-instruction exception handling begins, the CPU references the stack pointer (ER7) and pushes the program counter and other control registers onto the stack. Next, the CPU alters the settings of the interrupt mask bits in the control registers. Then the CPU fetches a start address (vector) from the exception vector table and execution branches to that address.

Figure 3-3 shows the stack after exception handling ends, for the case of interrupt mode 1 in advanced mode.



**Figure 3-3 Stack Structure after Exception Handling (Example)**

### 3.4 Program Execution State

In this state the CPU executes program instructions in sequence.

### 3.5 Bus-Released State

This is a state in which the bus has been released in response to a bus request from a bus master other than the CPU. While the bus is released, the CPU halts except for internal operations.

Bus masters other than the CPU may include the direct memory access controller (DMAC) and data transfer controller (DTC).

For further details, refer to the relevant microcontroller hardware manual.

### 3.6 Power-Down State

The power-down state includes both modes in which the CPU stops operating and modes in which the CPU does not stop. There are three modes in which the CPU stops operating: sleep mode, software standby mode, and hardware standby mode. There are also two other power-down modes: medium-speed mode and module stop mode. In medium-speed mode the CPU and other bus masters operate on a medium-speed clock. Module stop mode permits halting of the operation of individual modules, other than the CPU. For details, refer to the relevant microcontroller hardware manual.

#### 3.6.1 Sleep Mode

A transition to sleep mode is made if the SLEEP instruction is executed while the software standby bit (SSBY) in the system control register (SYSCR) is cleared to 0. In sleep mode, CPU operations stop immediately after execution of the SLEEP instruction. The contents of CPU registers are retained.

#### 3.6.2 Software Standby Mode

A transition to software standby mode is made if the SLEEP instruction is executed while the SSBY bit in SYSCR is set to 1. In software standby mode, the CPU and clock halt and all on-chip operations stop. The on-chip supporting modules are reset, but as long as a specified voltage is supplied, the contents of CPU registers and on-chip RAM are retained. The I/O ports also remain in their existing states.

#### 3.6.3 Hardware Standby Mode

A transition to hardware standby mode is made when the  $\overline{\text{STBY}}$  pin goes low. In hardware standby mode, the CPU and clock halt and all on-chip operations stop. The on-chip supporting modules are reset, but as long as a specified voltage is supplied, on-chip RAM contents are retained.

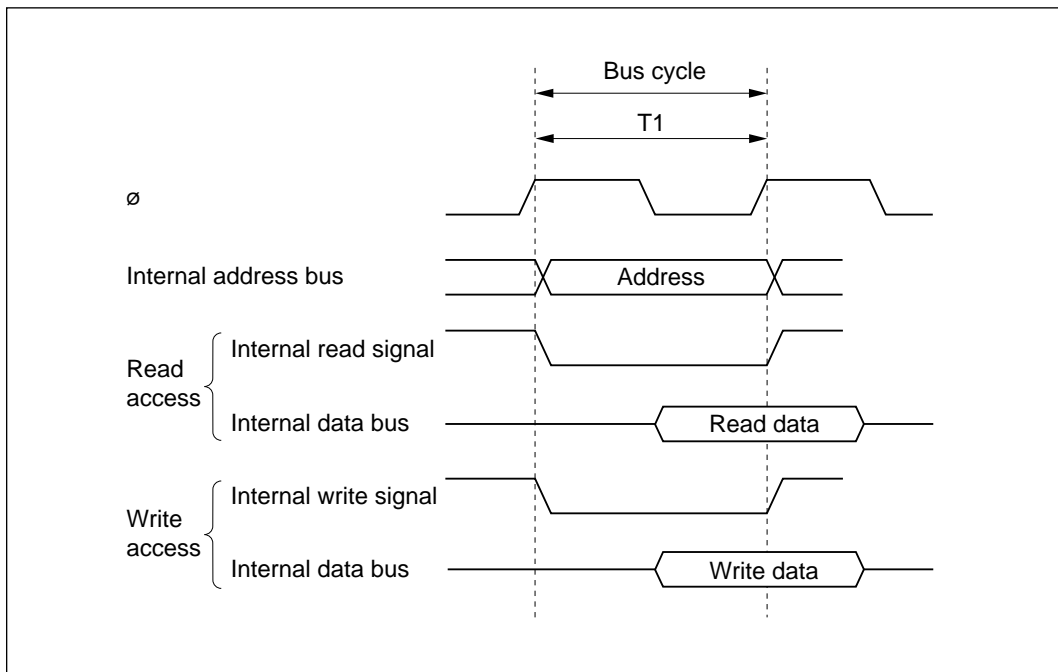
## Section 4 Basic Timing

### 4.1 Overview

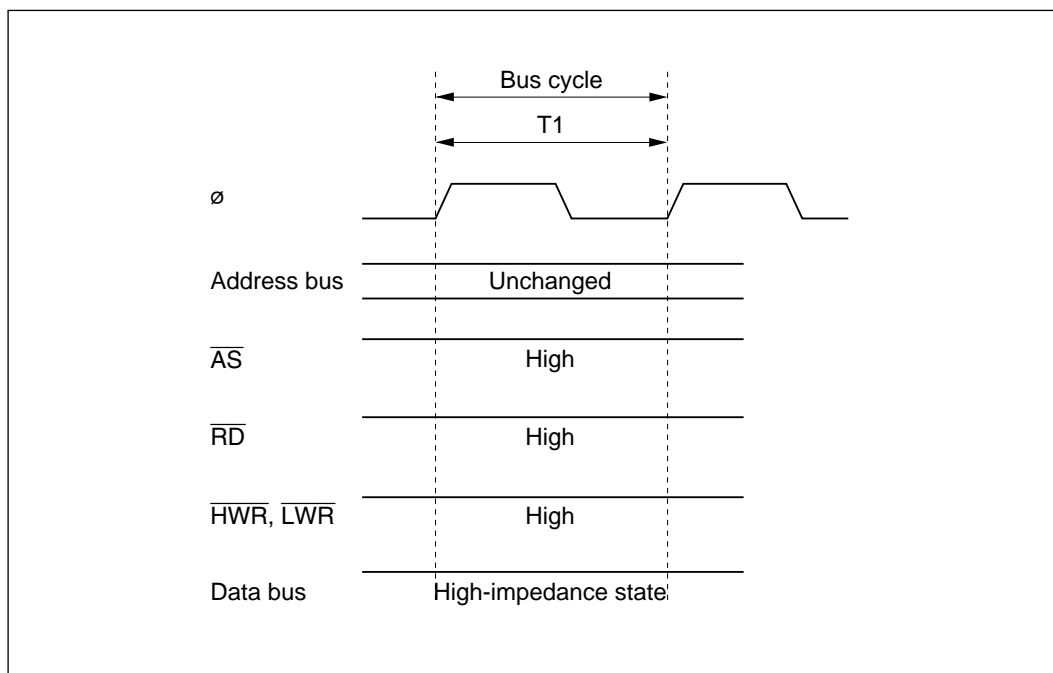
The CPU is driven by a system clock, denoted by the symbol  $\phi$ . The period from one rising edge of  $\phi$  to the next is referred to as a “state.” The memory cycle or bus cycle consists of one, two, or three states. Different methods are used to access on-chip memory, on-chip supporting modules, and the external address space. Refer to the relevant microcontroller hardware manual for details.

### 4.2 On-Chip Memory (ROM, RAM)

On-chip memory is accessed in one state. The data bus is 16 bits wide, permitting both byte and word access. Figure 4-1 shows the on-chip memory access cycle. Figure 4-2 shows the pin states.



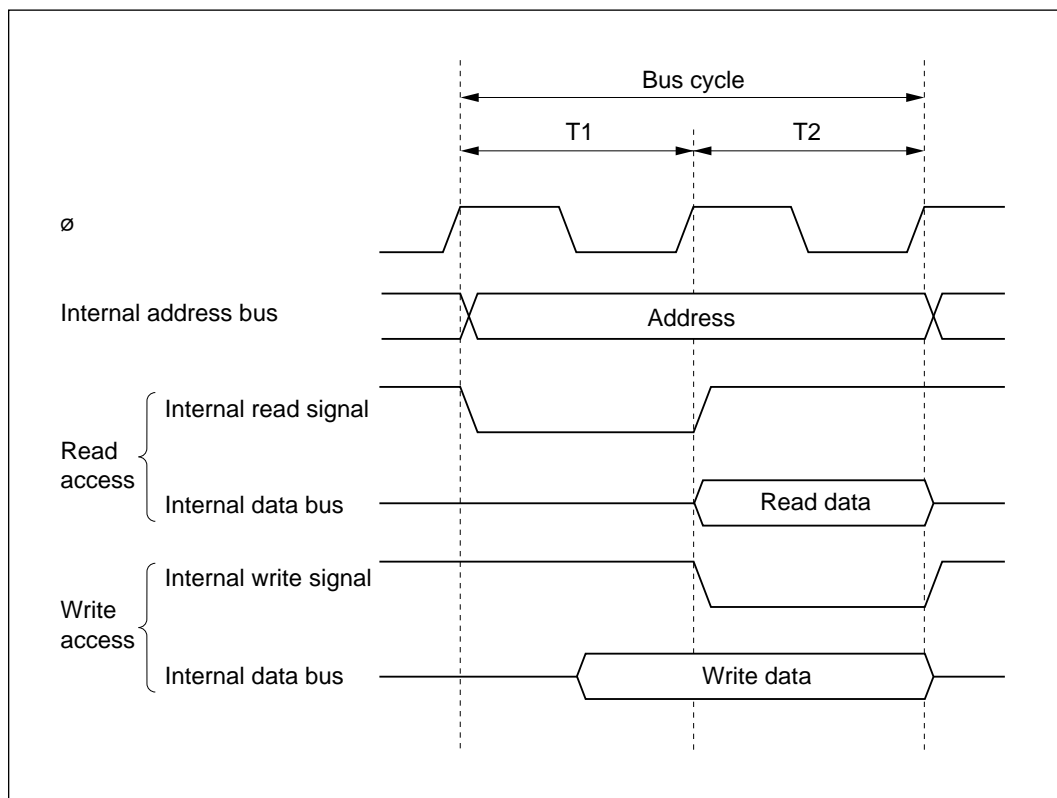
**Figure 4-1 On-Chip Memory Access Cycle**



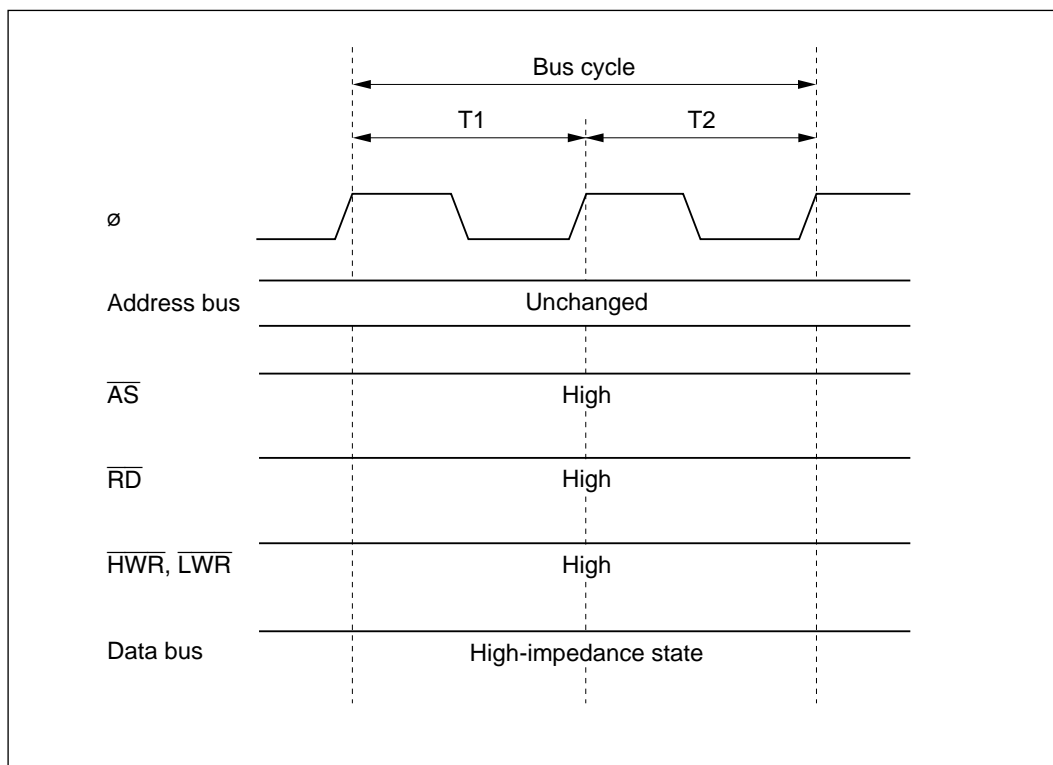
**Figure 4-2 Pin States during On-Chip Memory Access**

### 4.3 On-Chip Supporting Module Access Timing

The on-chip supporting modules are accessed in two states. The data bus is either 8 bits or 16 bits wide, depending on the particular on-chip register being accessed. Figure 4-3 shows the access timing for the on-chip supporting modules. Figure 4-4 shows the pin states.



**Figure 4-3 On-Chip Supporting Module Access Cycle**

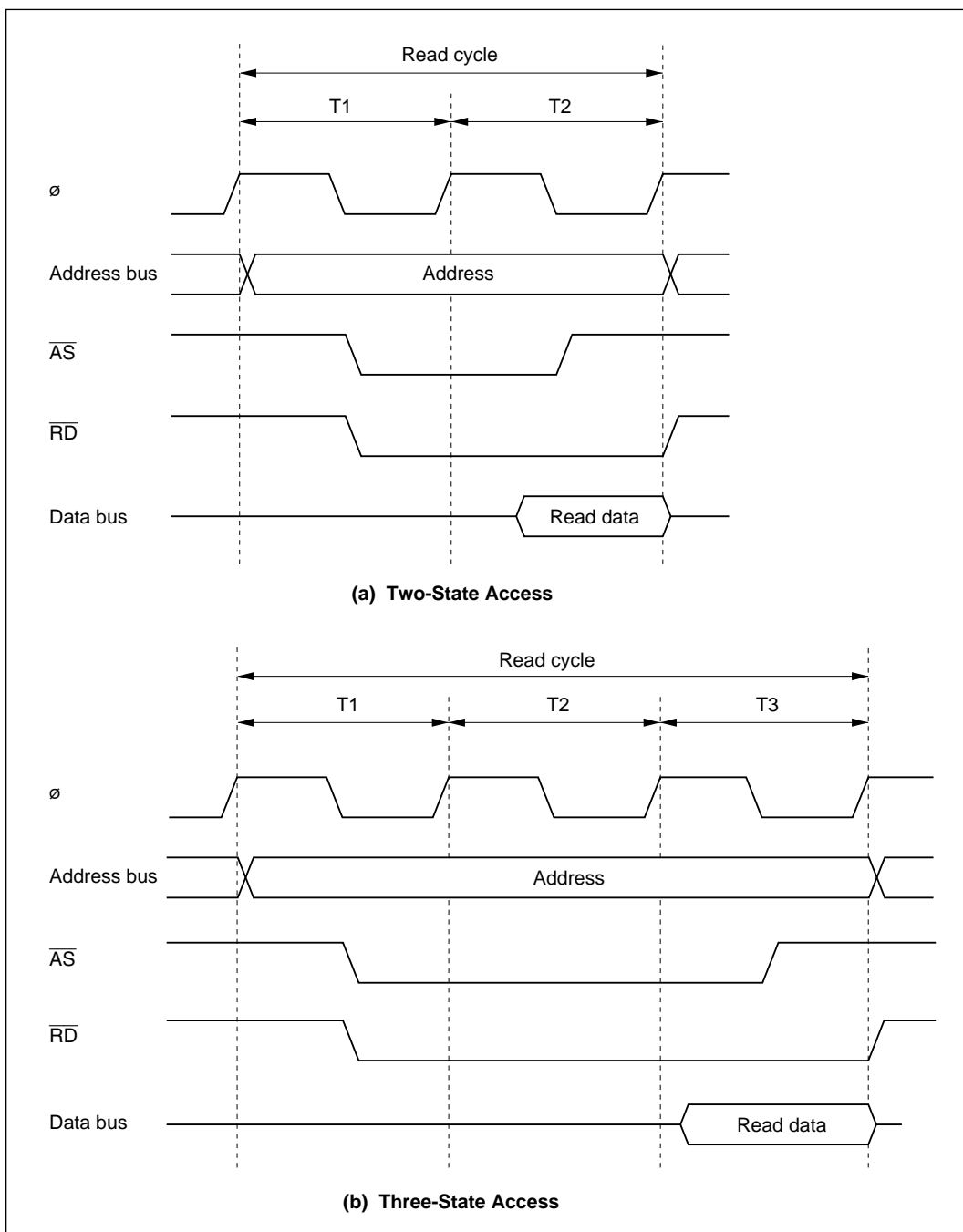


**Figure 4-4 Pin States during On-Chip Supporting Module Access**

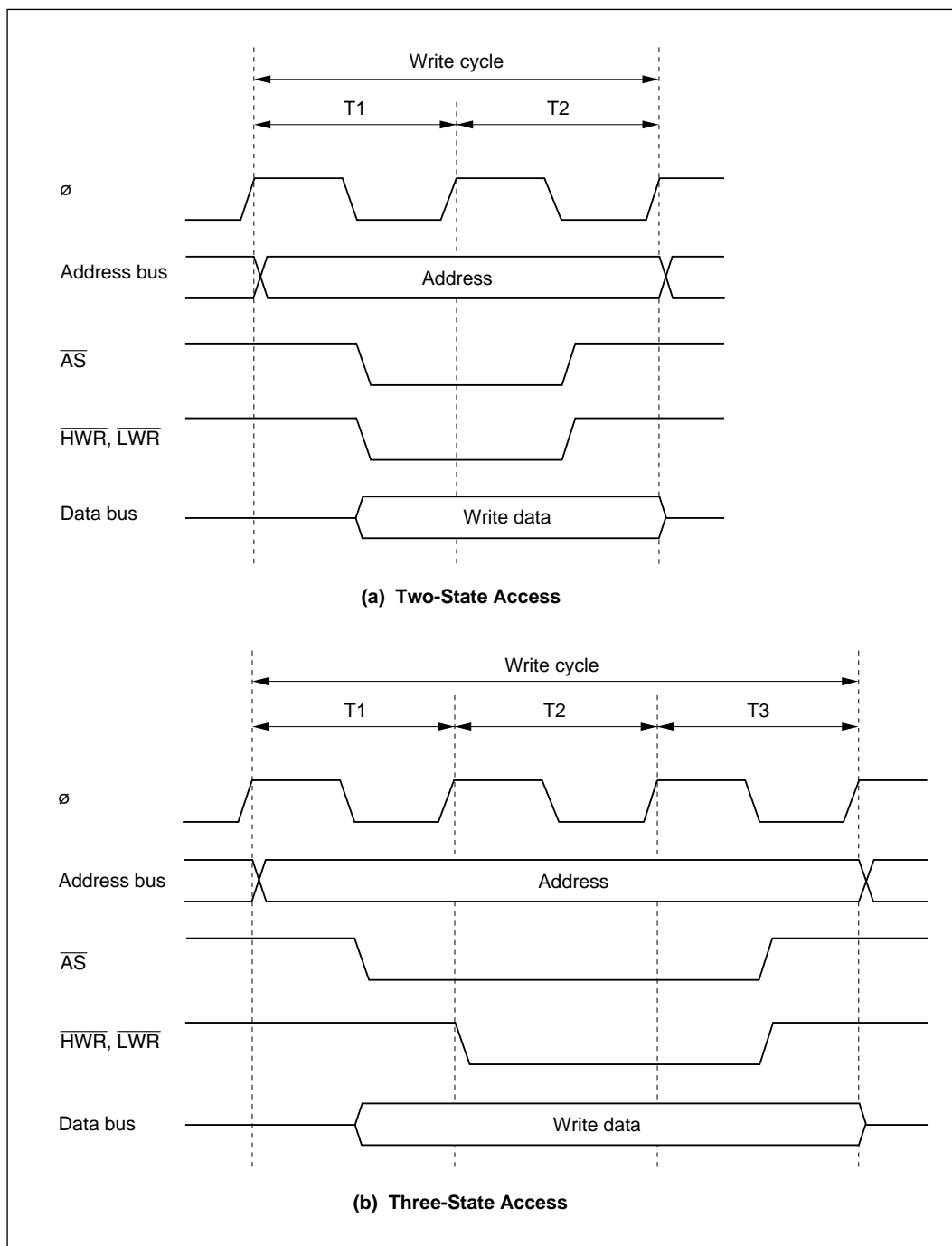
#### 4.4 External Address Space Access Timing

The external address space is accessed with an 8-bit or 16-bit data bus width in a two-state or three-state bus cycle. Figure 4-5 shows the read timing for two-state and three-state access. Figure 4-6 shows the write timing for two-state and three-state access. In three-state access, wait states can be inserted. For further details, refer to the relevant microcontroller hardware manual.





**Figure 4-5 External Device Access Timing (Read Timing)**



**Figure 4-6 External Device Access Timing (Write Timing)**