

The revision list can be viewed directly by clicking the title page.

The revision list summarizes the locations of revisions and additions. Details should always be checked by referring to the relevant text.

16

H8/300L Series Software Manual

Renesas 16-Bit Single-Chip Microcomputer
H8 Family/H8/300L Series

Software Manual

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
The information described here may contain technical inaccuracies or typographical errors.
Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (<http://www.renesas.com>).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

General Precautions on Handling of Product

1. Treatment of NC Pins

Note: Do not connect anything to the NC pins.

The NC (not connected) pins are either not connected to any of the internal circuitry or are used as test pins or to reduce noise. If something is connected to the NC pins, the operation of the LSI is not guaranteed.

2. Treatment of Unused Input Pins

Note: Fix all unused input pins to high or low level.

Generally, the input pins of CMOS products are high-impedance input pins. If unused pins are in their open states, intermediate levels are induced by noise in the vicinity, a pass-through current flows internally, and a malfunction may occur.

3. Processing before Initialization

Note: When power is first supplied, the product's state is undefined.

The states of internal circuits are undefined until full power is supplied throughout the chip and a low level is input on the reset pin. During the period where the states are undefined, the register settings and the output state of each pin are also undefined. Design your system so that it does not malfunction because of processing while it is in this undefined state. For those products which have a reset function, reset the LSI immediately after the power supply has been turned on.

4. Prohibition of Access to Undefined or Reserved Addresses

Note: Access to undefined or reserved addresses is prohibited.

The undefined or reserved addresses may be used to expand functions, or test registers may have been allocated to these addresses. Do not access these registers; the system's operation is not guaranteed if they are accessed.

Preface

The H8/300L Series of single-chip microcomputers is built around the high-speed H8/300L CPU, with an architecture featuring eight 16-bit (or sixteen 8-bit) general registers and a concise, optimized instruction set.

This manual gives detailed descriptions of the H8/300L instructions. The descriptions apply to all chips in the H8/300L Series. Assembly-language programmers should also read the separate H8/300 Series Cross Assembler User's Manual.

For hardware details, refer to the hardware manual of the specific chip.

Main Revisions for this Edition

| Item | Page | Revision (See Manual for Details) |
|------|------|--|
| All | — | All references to Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names changed to Renesas Technology Corp. Designation for categories changed from “series” to “group” |

Contents

| | | |
|-----------|---|----|
| Section 1 | CPU | 1 |
| 1.1 | Overview..... | 1 |
| 1.1.1 | Features | 1 |
| 1.1.2 | Data Structure | 2 |
| 1.1.3 | Address Space..... | 5 |
| 1.1.4 | Register Configuration..... | 5 |
| 1.2 | Registers..... | 6 |
| 1.2.1 | General Registers..... | 6 |
| 1.2.2 | Control Registers | 6 |
| 1.2.3 | Initial Register Values..... | 7 |
| 1.3 | Instructions..... | 8 |
| 1.3.1 | Types of Instructions..... | 8 |
| 1.3.2 | Instruction Functions..... | 9 |
| 1.3.3 | Basic Instruction Formats | 20 |
| 1.3.4 | Addressing Modes and Effective Address Calculation..... | 26 |
| Section 2 | Instruction Set | 31 |
| 2.1 | Explanation Format..... | 31 |
| 2.2 | Instructions..... | 36 |
| 2.2.1 (1) | ADD (add binary) (byte)..... | 36 |
| 2.2.1 (2) | ADD (add binary) (word)..... | 37 |
| 2.2.2 | ADDS (add with sign extension)..... | 38 |
| 2.2.3 | ADDX (add with extend carry)..... | 39 |
| 2.2.4 | AND (AND logical) | 40 |
| 2.2.5 | ANDC (AND control register)..... | 41 |
| 2.2.6 | BAND (bit AND)..... | 42 |
| 2.2.7 | Bcc (branch conditionally) | 43 |
| 2.2.8 | BCLR (bit clear)..... | 46 |
| 2.2.9 | BIAND (bit invert AND) | 47 |
| 2.2.10 | BILD (bit invert load) | 48 |
| 2.2.11 | BIOR (bit invert inclusive OR) | 49 |
| 2.2.12 | BIST (bit invert store) | 50 |
| 2.2.13 | BIXOR (bit invert exclusive OR)..... | 51 |
| 2.2.14 | BLD (bit load)..... | 52 |
| 2.2.15 | BNOT (bit NOT)..... | 53 |
| 2.2.16 | BOR (bit inclusive OR)..... | 54 |
| 2.2.17 | BSET (bit set)..... | 55 |

| | | |
|------------|--|----|
| 2.2.18 | BSR (branch to subroutine) | 56 |
| 2.2.19 | BST (bit store) | 57 |
| 2.2.20 | BTST (bit test) | 58 |
| 2.2.21 | BXOR (bit exclusive OR) | 59 |
| 2.2.22 (1) | CMP (compare) (byte)CMP | 60 |
| 2.2.22 (2) | CMP (compare) (word)CMP | 61 |
| 2.2.23 | DAA (decimal adjust add) | 62 |
| 2.2.24 | DAS (decimal adjust subtract) | 64 |
| 2.2.25 | DEC (decrement) | 65 |
| 2.2.26 | DIVXU (divide extend as unsigned) | 66 |
| 2.2.27 | EEPMOV (move data to EEPROM) | 68 |
| 2.2.28 | INC (increment) | 69 |
| 2.2.29 | JMP (jump) | 70 |
| 2.2.30 | JSR (Jump to subroutine) | 71 |
| 2.2.31 | LDC (load to control register) LDC | 72 |
| 2.2.32 (1) | MOV (move data) (byte)MOV | 73 |
| 2.2.32 (2) | MOV (move data) (word)MOV | 74 |
| 2.2.32 (3) | MOV (move data) (byte)MOV | 75 |
| 2.2.32 (4) | MOV (move data) (word)MOV | 76 |
| 2.2.32 (5) | MOV (move data) (byte)MOV | 77 |
| 2.2.32 (6) | MOV (move data) (word)MOV | 78 |
| 2.2.33 | MULXU (multiply extend as unsigned) | 79 |
| 2.2.34 | NEG (negate) | 80 |
| 2.2.35 | NOP (no operation) | 81 |
| 2.2.36 | NOT (NOT = logical complement) | 82 |
| 2.2.37 | OR (inclusive OR logical) | 83 |
| 2.2.38 | ORC (inclusive OR control register) | 84 |
| 2.2.39 | POP (pop data) | 85 |
| 2.2.40 | PUSH (push data) | 86 |
| 2.2.41 | ROTL (rotate left) | 87 |
| 2.2.42 | ROTR (rotate right) | 88 |
| 2.2.43 | ROTXL (rotate with extend carry left) | 89 |
| 2.2.44 | ROTXR (rotate with extend carry right) | 90 |
| 2.2.45 | RTE (return from exception) | 91 |
| 2.2.46 | RTS (return from subroutine) | 92 |
| 2.2.47 | SHAL (shift arithmetic left) | 93 |
| 2.2.48 | SHAR (shift arithmetic right) | 94 |
| 2.2.49 | SHLL (shift logical left) | 95 |
| 2.2.50 | SHLR (shift logical right) | 96 |
| 2.2.51 | SLEEP (sleep) | 97 |
| 2.2.52 | STC (store from control register) | 98 |

| | | |
|--|---|-----|
| 2.2.53 (1) | SUB (subtract binary) (byte)SUB | 99 |
| 2.2.53 (2) | SUB (subtract binary) (word)SUB | 100 |
| 2.2.54 | SUBS (subtract with sign extension)..... | 101 |
| 2.2.55 | SUBX (subtract with extend carry) | 102 |
| 2.2.56 | XOR (exclusive OR logical) | 103 |
| 2.2.57 | XORC (exclusive OR control register) | 104 |
| 2.3 | Operation Code Map | 105 |
| 2.4 | List of Instructions | 107 |
| 2.5 | Number of Execution States..... | 114 |
| Section 3 CPU Operation States | | 121 |
| 3.1 | Program Execution State..... | 122 |
| 3.2 | Exception Handling States | 122 |
| 3.2.1 | Types and Priorities of Exception Handling | 122 |
| 3.2.2 | Exception Sources and Vector Table | 123 |
| 3.2.3 | Outline of Exception Handling Operation..... | 124 |
| 3.3 | Reset State..... | 125 |
| 3.4 | Power-Down State | 125 |
| Section 4 Basic Operation Timing | | 127 |
| 4.1 | On-chip Memory (RAM, ROM)..... | 127 |
| 4.2 | On-chip Peripheral Modules and External Devices | 128 |

Section 1 CPU

1.1 Overview

The H8/300L CPU at the heart of the H8/300L Series features 16 general registers of 8 bits each (or 8 registers of 16-bits each), and a concise, optimized instruction set geared to high-speed operation.

1.1.1 Features

The H8/300L CPU has the following features.

General register configuration

16 8-bit registers (can be used as 8 16-bit registers)

55 basic instructions

- Multiply and divide instructions
- Powerful bit manipulation instructions

8 addressing modes

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

64-kbyte address space

High-speed operation

- All frequently used instructions are executed in 2 to 4 states
- High-speed operating frequency: 5 MHz
Add/subtract between 8/16-bit registers: 0.4 μ s
8 \times 8-bit multiply: 2.8 μ s
16 \div 8-bit divide: 2.8 μ s

Low-power operation

- Transition to power-down state using SLEEP instruction

1.1.2 Data Structure

The H8/300L CPU can process 1-bit data, 4-bit (packed BCD) data, 8-bit (byte) data, and 16-bit (word) data.

- Bit manipulation instructions operate on 1-bit data specified as bit n ($n = 0, 1, 2, \dots, 7$) in a byte operand.
- All operational instructions except ADDS and SUBS can operate on byte data.
- The MOV.W, ADD.W, SUB.W, CMP.W, ADDS, SUBS, MULXU (8 bits \times 8 bits), and DIVXU (16 bits \div 8 bits) instructions operate on word data.
- The DAA and DAS instruction perform decimal arithmetic adjustments on byte data in packed BCD form. Each 4-bit of the byte is treated as a decimal digit.

Data Structure in General Registers: Data of all the sizes above can be stored in general registers as shown in figure 1-1.

| Data type | Register No. | Data format |
|---|--------------|-------------|
| 1-Bit data | RnH | |
| 1-Bit data | RnL | |
| Byte data | RnH | |
| Byte data | RnL | |
| Word data | Rn | |
| 4-Bit BCD data | RnH | |
| 4-Bit BCD data | RnL | |
| RnH: Upper 8 bits of General Register RnL: Lower 8 bits of General Register MSB: Most Significant Bit LSB: Least Significant Bit | | |

Figure 1-1 Register Data Structure

Data Structure in Memory: Figure 1-2 shows the structure of data in memory. The H8/300L CPU is able to access word data in memory (MOV.W instruction), but only if the word data starts from an even-numbered address. If an odd address is designated, no address error occurs, but the access is performed starting from the previous even address, with the least significant bit of the address regarded as 0.* The same applies to instruction codes.

* Note that the LSIs in the H8/300L Series also contain on-chip peripheral modules for which access in word size is not possible. Details are given in the applicable hardware manual.

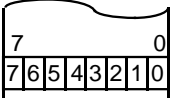

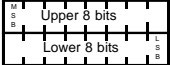
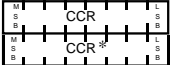
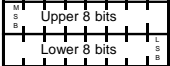
| Data type | Address | Data format |
|--|-----------------------------|---|
| 1-Bit data | Address n |  |
| Byte data | Address n |  |
| Word data | Even address Odd address |  |
| Byte data (CCR) on stack | Even address Odd address |  |
| Word data on stack | Even address Odd address |  |
| CCR: Condition code register. Note: Word data must begin at an even address. *: Ignored when returned. | | |

Figure 1-2 Memory Data Formats

The stack is always accessed a word at a time. When the CCR is pushed on the stack, two identical copies of the CCR are pushed to make a complete word. When they are returned, the lower byte is ignored.

1.1.3 Address Space

The H8/300L CPU supports a 64-kbyte address space (program code + data). The memory map differs depending on the particular chip in the H8/300L Series and its operating mode. See the applicable hardware manual for details.

1.1.4 Register Configuration

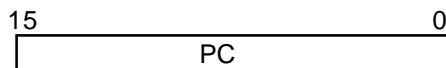
Figure 1-3 shows the register configuration of the H8/300L CPU. There are 16 8-bit general registers (R0H, R0L, ..., R7H, R7L), which can also be accessed as eight 16-bit registers (R0 to R7). There are two control registers: the 16-bit program counter (PC) and the 8-bit condition code register (CCR).

General Registers (Rn)

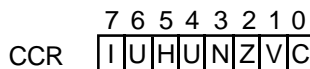
| | | |
|-----|------|-----|
| 7 | 0 7 | 0 |
| R0H | | R0L |
| R1H | | R1L |
| R2H | | R2L |
| R3H | | R3L |
| R4H | | R4L |
| R5H | | R5L |
| R6H | | R6L |
| R7H | (SP) | R7L |

SP: Stack Pointer

Control Registers (CR)



Program Counter



Condition Code Register

Carry flag

Overflow flag

Zero flag

Negative flag

Half-carry flag

Interrupt mask bit

User bit

Figure 1-3 CPU Registers

1.2 Registers

1.2.1 General Registers

All the general registers can be used as both data registers and address registers. When used as address registers, the general registers are accessed as 16-bit registers (R0 to R7). When used as data registers, they can be accessed as 16-bit registers (R0 to R7), or the high (R0H to R7H) and low (R0L to R7L) bytes can be accessed separately as 8-bit registers. The register length is determined by the instruction.

R7 also functions as the stack pointer, used implicitly by hardware in processing interrupts and subroutine calls. In assembly language, the letters SP can be coded as a synonym for R7. As indicated in figure 1-4, R7 (SP) points to the top of the stack.

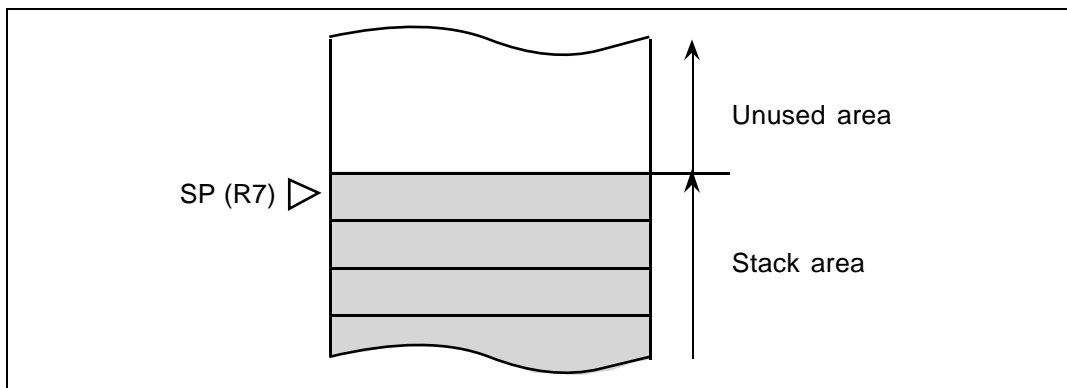


Figure 1-4 Stack Pointer

1.2.2 Control Registers

The CPU has a 16-bit program counter (PC) and an 8-bit condition code register (CCR).

(1) Program Counter (PC): This 16-bit register indicates the address of the next instruction the CPU will execute. Instructions are fetched by 16-bit (word) access, so the least significant bit of the PC is ignored (always regarded as 0).

(2) Condition Code Register (CCR): This 8-bit register indicates the internal status of the CPU with an interrupt mask (I) bit and five flag bits: half-carry (H), negative (N), zero (Z), overflow (V), and carry (C) flags. The two unused bits are available to the user. The bit configuration of the condition code register is shown below.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------------|-----|-----|-----|-----|-----|-----|-----|-----|
| | I | U | H | U | N | Z | V | C |
| Initial value | * | * | * | * | * | * | * | * |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

* Not fixed

Bit 7—Interrupt Mask Bit (I): When this bit is set to 1, all interrupts except NMI are masked. This bit is set to 1 automatically at the start of interrupt handling.

Bits 6 and 4—User Bits (U): These bits can be written and read by software for its own purposes using LDC, STC, ANDC, ORC, and XORC instructions.

Bit 5—Half-Carry (H): This bit is used by add, subtract, and compare instructions to indicate a borrow or carry out of bit 3 or bit 11. It is referenced by the decimal adjust instructions.

Bit 3—Negative (N): This bit indicates the value of the most significant bit (sign bit) of the result of an instruction.

Bit 2—Zero (Z): This bit is set to 1 to indicate a zero result and cleared to 0 to indicate a nonzero result.

Bit 1—Overflow (V): This bit is set to 1 when an arithmetic overflow occurs, and cleared to 0 at other times.

Bit 0—Carry (C): This bit is used by:

- Add, subtract, and compare instructions, to indicate a carry or borrow at the most significant bit
- Shift and rotate instructions, to store the value shifted out of the most or least significant bit
- Bit manipulation instructions, as a bit accumulator

Note that some instructions involve no flag changes. The flag operations with each instruction are indicated in the individual instruction descriptions that follow in section 2, Instruction Set. CCR is used by LDC, STC, ANDC, ORC, and XORC instructions. The N, Z, V, and C flags are used by the conditional branch instruction (Bcc).

1.2.3 Initial Register Values

When the CPU is reset, the program counter (PC) is loaded from the vector table and the interrupt mask bit (I) in CCR is set to 1. The other CCR bits and the general registers are not initialized.

The initial value of the stack pointer (R7) is not fixed. To prevent program crashes the stack pointer should be initialized by software, by the first instruction executed after a reset.

1.3 Instructions

Features:

- The H8/300L CPU has a concise set of 55 instructions.
- A general-register architecture is adopted.
- All instructions are 2 or 4 bytes long.
- Fast multiply/divide instructions and extensive bit manipulation instructions are supported.
- Eight addressing modes are supported.

1.3.1 Types of Instructions

Table 1-1 classifies the H8/300L instructions by type. Section 2, Instruction Set, gives detailed descriptions.

Table 1-1 Instruction Classification

| Function | Instructions | Types |
|-----------------------|--|-------|
| Data transfer | MOV, POP*, PUSH* | 1 |
| Arithmetic operations | ADD, SUB, ADDX, SUBX, INC, DEC, ADDS, SUBS, DAA, DAS, MULXU, DIVXU, CMP, NEG | 14 |
| Logic operations | AND, OR, XOR, NOT | 4 |
| Shift | SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR | 8 |
| Bit manipulation | BSET, BCLR, BNOT, BTST, BAND, BIAN, BOR, BIOR, BXOR, BIXOR, BLD, BILD, BST, BIST | 14 |
| Branch | Bcc**, JMP, BSR, JSR, RTS | 5 |
| System control | RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP | 8 |
| Block data transfer | EEPMOV | 1 |

Total 55

* POP Rn is equivalent to MOV.W @SP+, Rn.
 PUSH Rn is equivalent to MOV.W Rn, @-SP.

** Bcc is a conditional branch instruction in which cc represents a condition.

1.3.2 Instruction Functions

Tables 1-2 to 1-9 give brief descriptions of the instructions in each functional group.

The following notation is used.

Notation

| | |
|-------------|--------------------------------|
| Rd | General register (destination) |
| Rs | General register (source) |
| Rn | General register |
| (EAd) | Destination operand |
| (EAs) | Source operand |
| CCR | Condition code register |
| N | N (negative) bit of CCR |
| Z | Z (zero) bit of CCR |
| V | V (overflow) bit of CCR |
| C | C (carry) bit of CCR |
| PC | Program counter |
| SP | Stack pointer (R7) |
| #Imm | Immediate data |
| op | Operation field |
| disp | Displacement |
| + | Addition |
| – | Subtraction |
| × | Multiplication |
| ÷ | Division |
| ^ | AND logical |
| ∨ | OR logical |
| ⊕ | Exclusive OR logical |
| → | Move |
| ¬ | Not |
| :3, :8, :16 | 3-bit, 8-bit, or 16-bit length |

Table 1-2 Data Transfer Instructions

| Instruction | Size* | Function |
|--------------------|--------------|--|
| MOV | B/W | <p>(EAs) → Rd, Rs → (EAd)</p> <p>Moves data between two general registers or between a general register and memory, or moves immediate data to a general register.</p> <p>The Rn, @Rn, @(d:16, Rn), @aa:16, #xx:8 or #xx:16, @-Rn, and @Rn+ addressing modes are available for byte or word data. The @aa:8 addressing mode is available for byte data only.</p> <p>The @-R7 and @R7+ modes require word operands. Do not specify byte size for these two modes.</p> |
| POP | W | <p>@SP+ → Rn</p> <p>Pops a 16-bit general register from the stack.</p> <p>Equivalent to MOV.W @SP+, Rn.</p> |
| PUSH | W | <p>Rn → @-SP</p> <p>Pushes a 16-bit general register onto the stack.</p> <p>Equivalent to MOV.W Rn, @-SP.</p> |

* Size: Operand size

B: Byte

W: Word

Table 1-3 Arithmetic Instructions

| Instruction | Size* | Function |
|--------------------|--------------|---|
| ADD | B/W | $Rd \pm Rs \rightarrow Rd, Rd + \#Imm \rightarrow Rd$ |
| SUB | | Performs addition or subtraction on data in two general registers, or addition on immediate data and data in a general register. Immediate data cannot be subtracted from data in a general register. Word data can be added or subtracted only when both words are in general registers. |
| ADDX | B | $Rd \pm Rs \pm C \rightarrow Rd, Rd \pm \#Imm \pm C \rightarrow Rd$ |
| SUBX | | Performs addition or subtraction with carry or borrow on byte data in two general registers, or addition or subtraction on immediate data and data in a general register. |
| INC | B | $Rd \pm 1 \rightarrow Rd$ |
| DEC | | Increments or decrements a general register. |
| ADDS | W | $Rd \pm 1 \rightarrow Rd, Rd \pm 2 \rightarrow Rd$ |
| SUBS | | Adds or subtracts immediate data to or from data in a general register. The immediate data must be 1 or 2. |
| DAA | B | $Rd \text{ decimal adjust} \rightarrow Rd$ |
| DAS | | Decimal-adjusts (adjusts to packed BCD) an addition or subtraction result in a general register by referring to the condition code register. |
| MULXU | B | $Rd \times Rs \rightarrow Rd$ |
| | | Performs 8-bit \times 8-bit unsigned multiplication on data in two general registers, providing a 16-bit result. |
| DIVXU | B | $Rd \div Rs \rightarrow Rd$ |
| | | Performs 16-bit \div 8-bit unsigned division on data in two general registers, providing an 8-bit quotient and 8-bit remainder. |
| CMP | B/W | $Rd - Rs, Rd - \#Imm$ |
| | | Compares data in a general register with data in another general register or with immediate data. Word data can be compared only between two general registers. |
| NEG | B | $0 - Rd \rightarrow Rd$ |
| | | Obtains the two's complement (arithmetic complement) of data in a general register. |

* Size: Operand size

B: Byte

W: Word

Table 1-4 Logic Operation Instructions

| Instruction | Size* | Function |
|--------------------|--------------|---|
| AND | B | $Rd \wedge Rs \rightarrow Rd$, $Rd \wedge \#Imm \rightarrow Rd$ Performs a logical AND operation on a general register and another general register or immediate data. |
| OR | B | $Rd \vee Rs \rightarrow Rd$, $Rd \vee \#Imm \rightarrow Rd$ Performs a logical OR operation on a general register and another general register or immediate data. |
| XOR | B | $Rd \oplus Rs \rightarrow Rd$, $Rd \oplus \#Imm \rightarrow Rd$ Performs a logical exclusive OR operation on a general register and another general register or immediate data. |
| NOT | B | $\neg Rd \rightarrow Rd$ Obtains the one's complement (logical complement) of general register contents. |

* Size: Operand size

B: Byte

Table 1-5 Shift Instructions

| Instruction | Size* | Function |
|--------------------|--------------|--|
| SHAL | B | $Rd \text{ shift} \rightarrow Rd$ |
| SHAR | | Performs an arithmetic shift operation on general register contents. |
| SHLL | B | $Rd \text{ shift} \rightarrow Rd$ |
| SHLR | | Performs a logical shift operation on general register contents. |
| ROTL | B | $Rd \text{ rotate} \rightarrow Rd$ |
| ROTR | | Rotates general register contents. |
| ROTXL | B | $Rd \text{ rotate through carry} \rightarrow Rd$ |
| ROTXR | | Rotates general register contents through the C (carry) bit. |

* Size: Operand size

B: Byte

Table 1-6 Bit Manipulation Instructions

| Instruction | Size* | Function |
|--------------------|--------------|---|
| BSET | B | $1 \rightarrow (\text{<bit-No.> of <EAd>})$ Sets a specified bit in a general register or memory to 1. The bit is specified by a bit number, given in 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 to 0. The bit is specified by a bit number, given in 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. The bit is specified by a bit number, given in 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 and sets or clears the Z flag accordingly. The bit is specified by a bit number, given in 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 C flag with a specified bit in a general register or memory. |
| BIAND | B | $C \wedge [\neg (\text{<bit-No.> of <EAd>})] \rightarrow C$ ANDs the C flag with the inverse of a specified bit in a general register or memory. The bit number is specified by 3-bit immediate data. |
| BOR | B | $C \vee (\text{<bit-No.> of <EAd>}) \rightarrow C$ ORs the C flag with a specified bit in a general register or memory. |
| BIOR | B | $C \vee [\neg (\text{<bit-No.> of <EAd>})] \rightarrow C$ ORs the C flag with the inverse of a specified bit in a general register or memory. The bit number is specified by 3-bit immediate data. |

Table 1-6 Bit Manipulation Instructions (Cont.)

| Instruction | Size* | Function |
|--------------------|--------------|--|
| BXOR | B | $C \oplus (\text{<bit-No.> of <EAd>}) \rightarrow C$ Exclusive-ORs the C flag with a specified bit in a general register or memory. |
| BIXOR | B | $C \oplus [\neg (\text{<bit-No.> of <EAd>})] \rightarrow$ Exclusive-ORs the C flag with the inverse of a specified bit in a general register or memory. The bit number is specified by 3-bit immediate data. |
| BLD | B | $(\text{<bit-No.> of <EAd>}) \rightarrow C$ Copies a specified bit in a general register or memory to the C flag. |
| BILD | B | $\neg (\text{<bit-No.> of <EAd>}) \rightarrow C$ Copies the inverse of a specified bit in a general register or memory to the C flag. The bit number is specified by 3-bit immediate data. |
| BST | B | $C \rightarrow (\text{<bit-No.> of <EAd>})$ Copies the C flag to a specified bit in a general register or memory. |
| BIST | B | $\neg C \rightarrow (\text{<bit-No.> of <EAd>})$ Copies the inverse of the C flag to a specified bit in a general register or memory. The bit number is specified by 3-bit immediate data. |

* Size: Operand size

B: Byte

Table 1-7 Branching Instructions

| Instruction | Size* | Function |
|--------------------|-------------------------------|--|
| Bcc | — | Branches if condition cc is true. The branching conditions are as follows. |
| Mnemonic | Description | Condition |
| 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 displacement from the current address. |
| JSR | — | Branches to a subroutine at a specified address. |
| RTS | — | Returns from a subroutine. |

Table 1-8 System Control Instructions

| Instruction | Size* | Function |
|--------------------|--------------|---|
| RTE | — | Returns from an exception handling routine. |
| SLEEP | — | Causes a transition to power-down state. |
| LDC | B | $R_s \rightarrow CCR, \#Imm \rightarrow CCR$ Moves immediate data or general register contents to the condition code register. |
| STC | B | $CCR \rightarrow R_d$ Copies the condition code register to a specified general register. |
| ANDC | B | $CCR \wedge \#Imm \rightarrow CCR$ Logically ANDs the condition code register with immediate data. |
| ORC | B | $CCR \vee \#Imm \rightarrow CCR$ Logically ORs the condition code register with immediate data. |
| XORC | B | $CCR \oplus \#Imm \rightarrow CCR$ Logically exclusive-ORs the condition code register with immediate data. |
| NOP | — | $PC + 2 \rightarrow PC$ Only increments the program counter. |

* Size: Operand size

B: Byte

Table 1-9 Block Data Transfer Instruction

| Instruction | Size* | Function |
|-------------|-------|--|
| EEPMOV | — | <p>if $R4L \neq 0$ then</p> <p>repeat @R5+ \rightarrow @R6+</p> <p>$R4L - 1 \rightarrow R4L$</p> <p>until $R4L = 0$</p> <p>else next;</p> <p>Moves a data block according to parameters set in general registers</p> <p>R4L, R5, and R6.</p> <p>R4L: size of block (bytes)</p> <p>R5: starting source address</p> <p>R6: starting destination address</p> <p>Execution of the next instruction starts as soon as the block transfer is completed.</p> <p>This instruction is for writing to the large-capacity EEPROM provided on chip with some models in the H8/300L Series. For details see the applicable hardware manual.</p> |

Notes on Bit Manipulation Instructions: BSET, BCLR, BNOT, BST, and BIST are read-modify-write instructions. They read a byte of data, modify one bit in the byte, then write the byte back. Care is required when these instructions are applied to registers with write-only bits and to the I/O port registers.

| Sequence | Operation |
|----------|--|
| 1 Read | Read one data byte at the specified address |
| 2 Modify | Modify one bit in the data byte |
| 3 Write | Write the modified data byte back to the specified address |

Example 1: BCLR is executed to clear bit 0 in port control register 4 (PCR4) under the following conditions.

P4⁷ : Input pin, Low

P4⁶ : Input pin, High

P4⁵ – P4⁰ : Output pins, Low

The intended purpose of this BCLR instruction is to switch P4⁰ from output to input.

Before Execution of BCLR Instruction

| | P47 | P46 | P45 | P44 | P43 | P42 | P41 | P40 |
|--------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Input/output | Input | Input | Output | Output | Output | Output | Output | Output |
| Pin state | Low | High | Low | Low | Low | Low | Low | Low |
| PCR4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| PDR4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Execution of BCLR Instruction

BCLR #0 @PCR4 ;clear bit 0 in PCR4

After Execution of BCLR Instruction

| | P47 | P46 | P45 | P44 | P43 | P42 | P41 | P40 |
|--------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Input/output | Output | Output | Output | Output | Output | Output | Output | Input |
| Pin state | Low | High | Low | Low | Low | Low | Low | High |
| PCR4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| PDR4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Explanation: To execute the BCLR instruction, the CPU begins by reading PCR4. Since PCR4 is a write-only register, it is read as H'FF, even though its true value is H'3F.

Next the CPU clears bit 0 of the read data, changing the value to H'FE.

Finally, the CPU writes this value (H'FE) back to PCR4 to complete the BCLR instruction.

As a result, bit 0 in PCR4 is cleared to 0, making P4⁰ an input pin. In addition, bits 7 and 6 in PCR4 are set to 1, making P4⁷ and P4⁶ output pins.

Example 2: BSET is executed to set bit 0 in the port 4 port data register (PDR4) under the following conditions.

P4⁷ : Input pin, Low

P4⁶ : Input pin, High

P4⁵ – P4⁰ : Output pins, Low

The intended purpose of this BSET instruction is to switch the output level at P4⁰ from Low to High.

Before Execution of BSET Instruction

| | P47 | P46 | P45 | P44 | P43 | P42 | P41 | P40 |
|--------------|-------|-------|--------|--------|--------|--------|--------|--------|
| Input/output | Input | Input | Output | Output | Output | Output | Output | Output |
| Pin state | Low | High | Low | Low | Low | Low | Low | Low |
| PCR4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| PDR4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Execution of BSET Instruction

BSET #0 @PDR4 ;set bit 0 in port 4 port data register

After Execution of BSET Instruction

| | P47 | P46 | P45 | P44 | P43 | P42 | P41 | P40 |
|--------------|-------|-------|--------|--------|--------|--------|--------|--------|
| Input/output | Input | Input | Output | Output | Output | Output | Output | Output |
| Pin state | Low | High | Low | Low | Low | Low | Low | High |
| PCR4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| PDR4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Explanation: To execute the BSET instruction, the CPU begins by reading port 4. Since P4⁷ and P4⁶ are input pins, the CPU reads the level of these pins directly, not the value in the port data register. It reads P4⁷ as Low (0) and P4⁶ as High (1).

Since P4⁵ to P4⁰ are output pins, for these pins the CPU reads the value in PDR4. The CPU therefore reads the value of port 4 as H'40, although the actual value in PDR4 is H'80.

Next the CPU sets bit 0 of the read data to 1, changing the value to H'41.

Finally, the CPU writes this value (H'41) back to PDR4 to complete the BSET instruction.

As a result, bit 0 in PDR4 is set to 0, switching pin P4⁰ to High output. However, bits 7 and 6 in PDR4 change their values.

1.3.3 Basic Instruction Formats

(1) Format of Data Transfer Instructions

Figure 1-5 shows the format used for data transfer instructions.

| | | | | |
|-------|----------------|----------------|----------------|---|
| 15 | 8 | 7 | 0 | MOV |
| op | | r _m | r _n | R _m → R _n |
| 15 | 8 | 7 | 0 | |
| op | | r _m | r _n | R _n → @R _m , or @R _m → R _n |
| 15 | 8 | 7 | 0 | |
| op | | r _m | r _n | @(d:16, R _m) → R _n , or |
| disp. | | | | R _n → @(d:16, R _m) |
| 15 | 8 | 7 | 0 | |
| op | | r _m | r _n | @R _m + → R _n , or R _n → @-R _m |
| 15 | 8 | 7 | 0 | |
| op | r _n | abs. | | @aa:8 → R _n , or R _n → @aa:8 |
| 15 | 8 | 7 | 0 | |
| op | | | r _n | @aa:16 → R _n , or |
| abs. | | | | R _n → @aa:16 |
| 15 | 8 | 7 | 0 | |
| op | r _n | IMM | | #xx:8 → R _n |
| 15 | 8 | 7 | 0 | |
| op | | | r _n | #xx:16 → R _n |
| IMM | | | | |
| 15 | 8 | 7 | 0 | |
| op | | | r _n | POP, PUSH |

Notation

op: Operation field

r_m, r_n: Register field

disp: Displacement

abs.: Absolute address

IMM: Immediate data

Figure 1-5 Instruction Format of Data Transfer Instructions

(2) Format of Arithmetic, Logic Operation, and Shift Instructions

Figure 1-6 shows the format used for arithmetic, logic operation, and shift instructions.

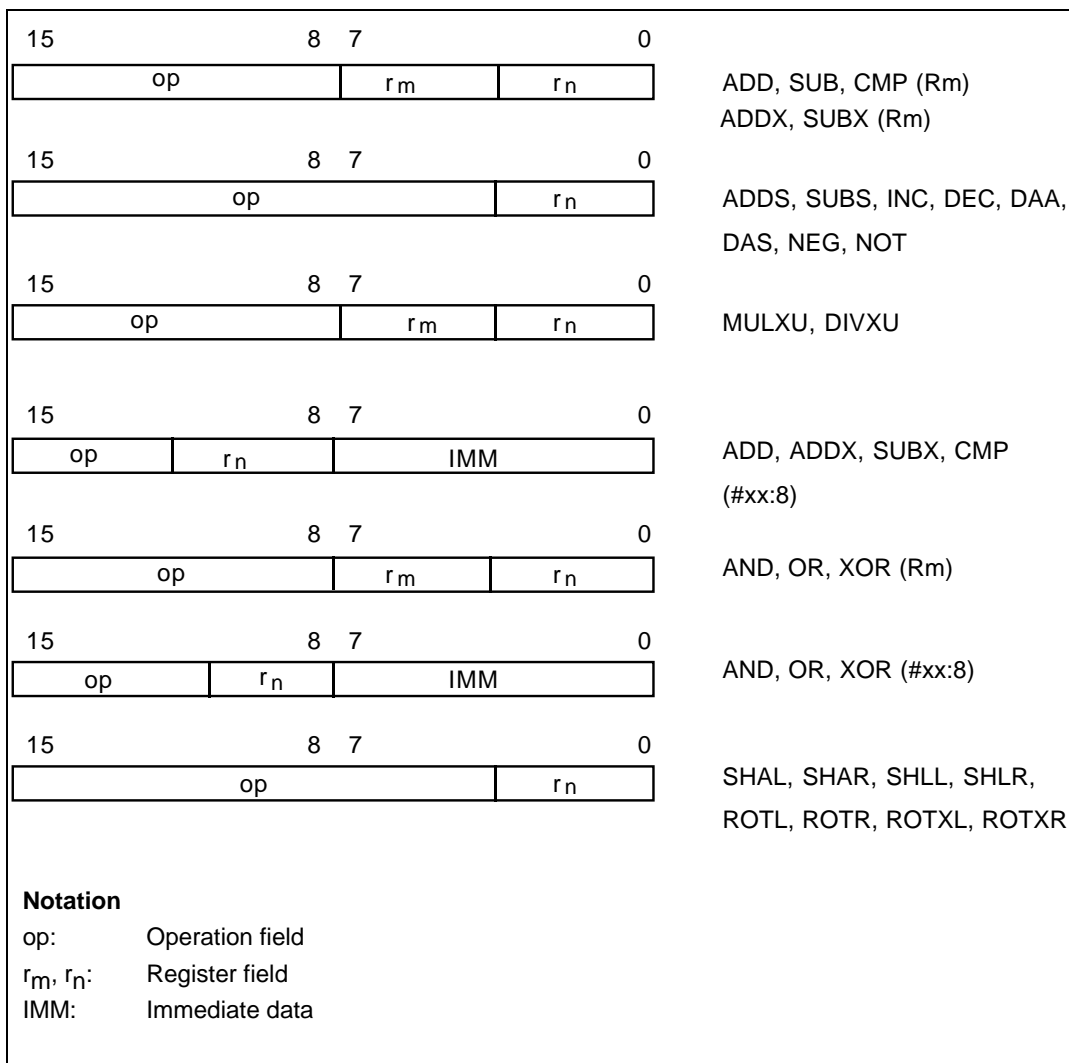


Figure 1-6 Instruction Format of Arithmetic, Logic, and Shift Instructions

(3) Format of Bit Manipulation Instructions

Figure 1-7 shows the format used for bit manipulation instructions.

| | | | | |
|----|---|----------------|----------------|----------------------------------|
| 15 | 8 | 7 | 0 | BSET, BCLR, BNOT, BTST |
| op | | IMM | r _n | Operand: register direct (Rn) |
| | | | | Bit No.: immediate (#xx:3) |
| 15 | 8 | 7 | 0 | |
| op | | r _m | r _n | Operand: register direct (Rn) |
| | | | | Bit No.: register direct (Rm) |
| 15 | 8 | 7 | 0 | |
| op | | r _n | 0 0 0 0 | Operand: register indirect (@Rn) |
| op | | IMM | 0 0 0 0 | Bit No.: immediate (#xx:3) |
| 15 | 8 | 7 | 0 | |
| op | | r _n | 0 0 0 0 | Operand: register indirect (@Rn) |
| op | | r _m | 0 0 0 0 | Bit No.: register direct (Rm) |
| 15 | 8 | 7 | 0 | |
| op | | abs. | | Operand: absolute (@aa:8) |
| op | | IMM | 0 0 0 0 | Bit No.: immediate (#xx:3) |
| 15 | 8 | 7 | 0 | |
| op | | abs. | | Operand: absolute (@aa:8) |
| op | | r _m | 0 0 0 0 | Bit No.: register direct (Rm) |
| 15 | 8 | 7 | 0 | BAND, BOR, BXOR, BLD, BST |
| op | | IMM | r _n | Operand: register direct (Rn) |
| | | | | Bit No.: immediate (#xx:3) |
| 15 | 8 | 7 | 0 | |
| op | | r _n | 0 0 0 0 | Operand: register indirect (@Rn) |
| op | | IMM | 0 0 0 0 | Bit No.: immediate (#xx:3) |
| 15 | 8 | 7 | 0 | |
| op | | abs. | | Operand: absolute (@aa:8) |
| op | | IMM | 0 0 0 0 | Bit No.: immediate (#xx:3) |

Notation

op: Operation field

r_m, r_n: Register field

abs.: Absolute address

IMM: Immediate data

Figure 1-7 Instruction Format of Bit Manipulation Instructions

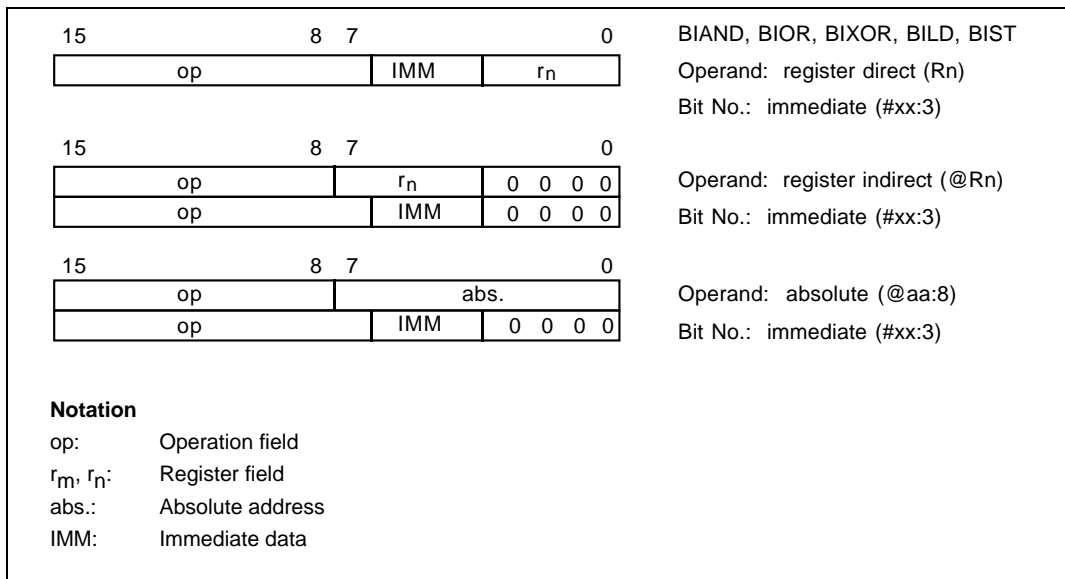


Figure 1-7 Instruction Format of Bit Manipulation Instructions (Cont.)

(4) Format of Branching Instructions

Figure 1-8 shows the format used for branching instructions.

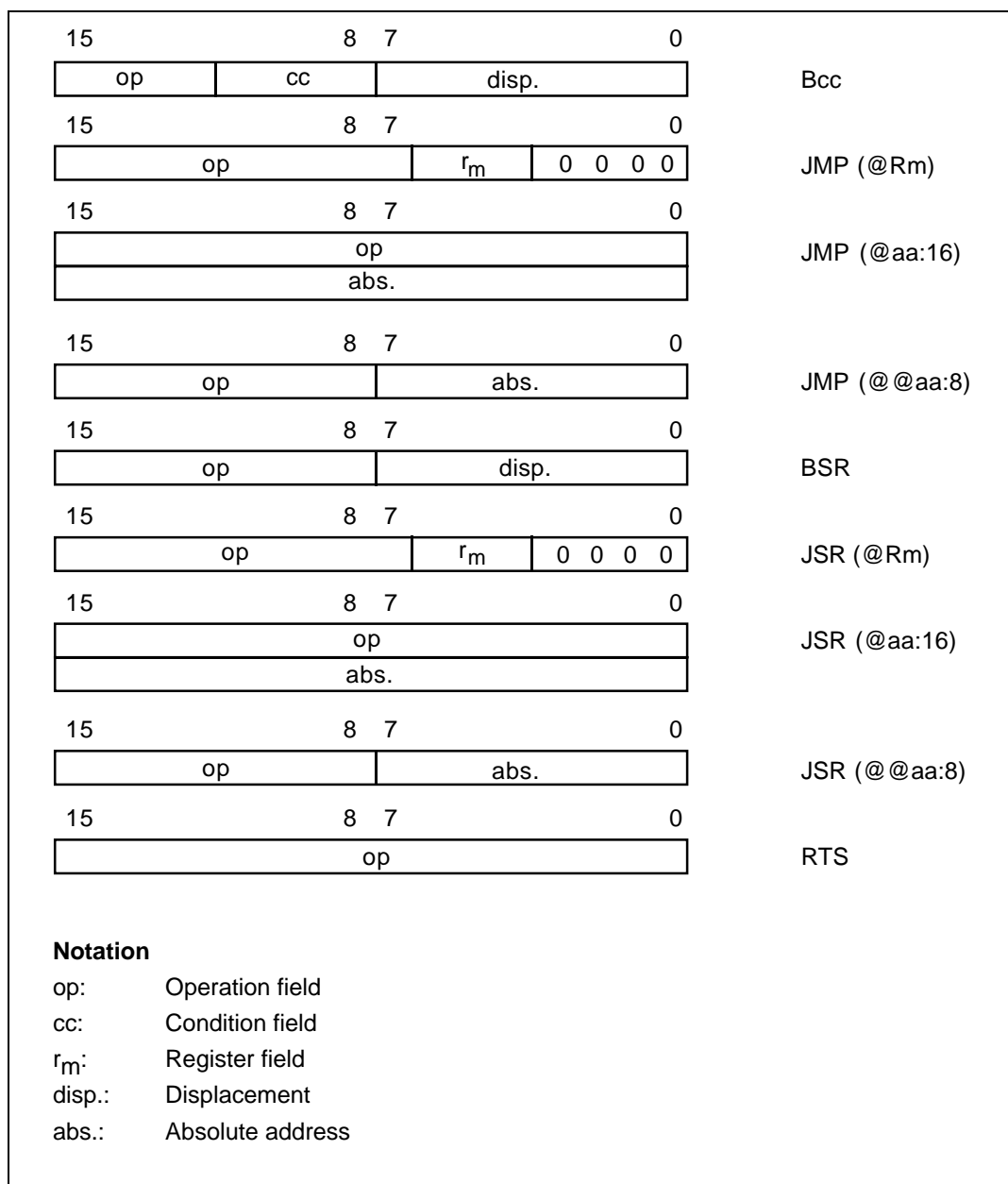


Figure 1-8 Instruction Format of Branching Instructions

(5) Format of System Control Instructions

Figure 1-9 shows the format used for system control instructions.

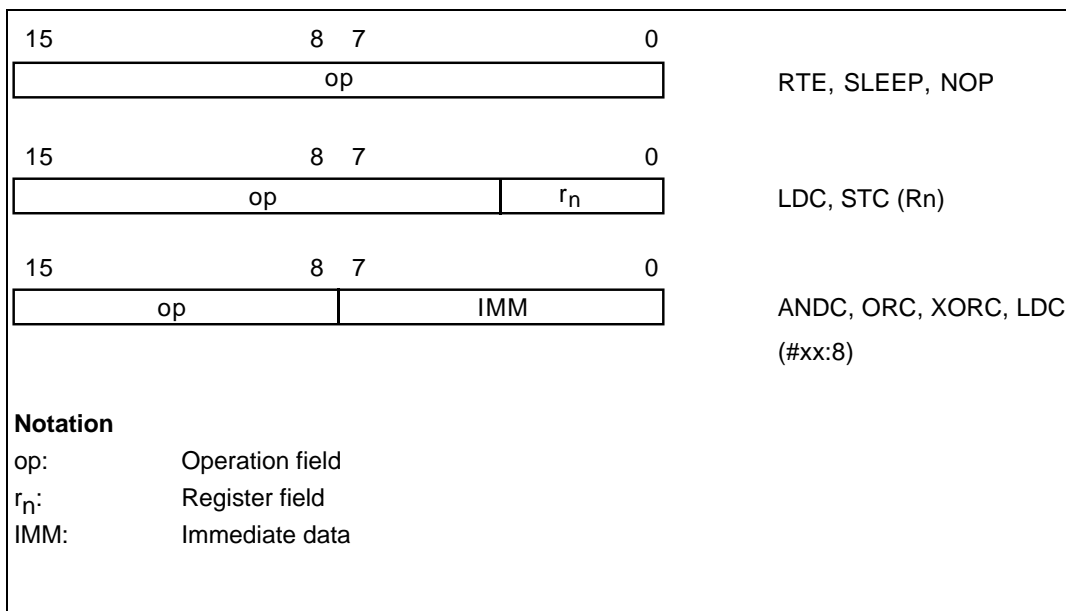


Figure 1-9 Instruction Format of System Control Instructions

(6) Format of Block Data Transfer Instruction

Figure 1-10 shows the format used for the block data transfer instruction.

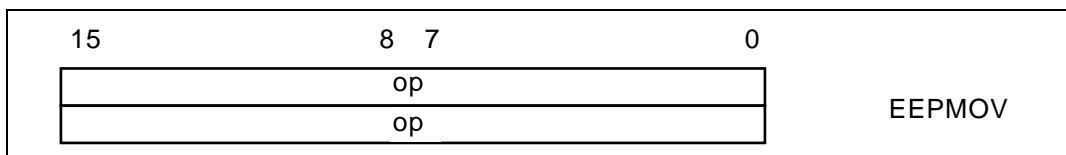


Figure 1-10 Instruction Format of Block Data Transfer Instruction

1.3.4 Addressing Modes and Effective Address Calculation

Table 1-10 lists the eight addressing modes and their assembly-language notation. Each instruction can use a specific subset of these addressing modes.

Arithmetic, logic, and shift instructions use register direct addressing (1). The ADD.B, ADDX, SUBX, CMP.B, AND, OR, and XOR instructions can also use immediate addressing (6).

The MOV instruction uses all the addressing modes except program-counter relative (7) and memory indirect (8).

Bit manipulation instructions use register direct (1), register indirect (2), or absolute (5) addressing to identify a byte operand and 3-bit immediate addressing to identify a bit within the byte. The BSET, BCLR, BNOT, and BTST instructions can also use register direct addressing (1) to identify the bit.

Table 1-10 Addressing Modes

| No. | Mode | Notation |
|-----|--|----------------------|
| (1) | Register direct | Rn |
| (2) | Register indirect | @Rn |
| (3) | Register indirect with 16-bit displacement | @(d:16, Rn) |
| (4) | Register indirect with post-increment | @Rn+ |
| | Register indirect with pre-decrement | @-Rn |
| (5) | Absolute address (8 or 16 bits) | @aa:8, @aa:16 |
| (6) | Immediate (3-, 8-, or 16-bit data) | #xx:3, #xx:8, #xx:16 |
| (7) | PC-relative (8-bit displacement) | @(d:8, PC) |
| (8) | Memory indirect | @@aa:8 |

(1) Register Direct—Rn: The register field of the instruction specifies an 8- or 16-bit general register containing the operand. In most cases the general register is accessed as an 8-bit register. Only the MOV.W, ADD.W, SUB.W, CMP.W, ADDS, SUBS, MULXU (8 bits × 8 bits), and DIVXU (16 bits ÷ 8 bits) instructions have 16-bit operands.

(2) Register Indirect—@Rn: The register field of the instruction specifies a 16-bit general register containing the address of the operand.

(3) Register Indirect with Displacement—@(d**:16, **Rn**):** This mode, which is used only in MOV instructions, is similar to register indirect but the instruction has a second word (bytes 3 and 4) which is added to the contents of the specified general register to obtain the operand address. For the MOV.W instruction, the resulting address must be even.

(4) Register Indirect with Post-Increment or Pre-Decrement—@Rn**+ or @-**Rn**:**

- Register indirect with post-increment—@**Rn**+

The @**Rn**+ mode is used with MOV instructions that load registers from memory.

It is similar to the register indirect mode, but the 16-bit general register specified in the register field of the instruction is incremented after the operand is accessed. The size of the increment is 1 or 2 depending on the size of the operand: 1 for a byte operand; 2 for a word operand. For a word operand, the original contents of the 16-bit general register must be even.

- Register indirect with pre-decrement—@-**Rn**

The @-**Rn** mode is used with MOV instructions that store register contents to memory.

It is similar to the register indirect mode, but the 16-bit general register specified in the register field of the instruction is decremented before the operand is accessed. The size of the decrement is 1 or 2 depending on the size of the operand: 1 for a byte operand; 2 for a word operand. For a word operand, the original contents of the 16-bit general register must be even.

(5) Absolute Address—@aa**:8 or @**aa**:16:** The instruction specifies the absolute address of the operand in memory. The @**aa**:8 mode uses an 8-bit absolute address of the form H'FFxx. The upper 8 bits are assumed to be 1, so the possible address range is H'FF00 to H'FFFF (65280 to 65535). The MOV.B, MOV.W, JMP, and JSR instructions can use 16-bit absolute addresses.

(6) Immediate—#xx**:8 or #**xx**:16:** The instruction contains an 8-bit operand in its second byte, or a 16-bit operand in its third and fourth bytes. Only MOV.W instructions can contain 16-bit immediate values. The ADDS and SUBS instructions implicitly contain the value 1 or 2 as immediate data. Some bit manipulation instructions contain 3-bit immediate data (#**xx**:3) in the second or fourth byte of the instruction, specifying a bit number.

(7) PC-Relative—@(d**:8, **PC**):** This mode is used to generate branch addresses in the Bcc and BSR instructions. An 8-bit value in byte 2 of the instruction code is added as a sign-extended value to the program counter contents. The result must be an even number. The possible branching range is -126 to +128 bytes (-63 to +64 words) from the current address.

(8) Memory Indirect—@@aa**:8:** This mode can be used by the JMP and JSR instructions. The second byte of the instruction code specifies an 8-bit absolute address from H'0000 to H'00FF (0 to 255). Note that the initial part of the area from H'0000 to H'00FF contains the exception vector table. See the applicable hardware manual for details. The word located at this address contains the branch address.

If an odd address is specified as a branch destination or as the operand address of a MOV.W instruction, the least significant bit is regarded as 0, causing word access to be performed at the address preceding the specified address. See the memory data structure description in section 1.1.2, Data Structure.

Effective Address Calculation

Table 1-11 explains how the effective address is calculated in each addressing mode.

Table 1-11 Effective Address Calculation (1)

| No. | Addressing mode, instruction format | Effective address calculation | Effective address |
|-----|--|---|--|
| 1 | Register direct Rn | None | <div><div><div>3030</div><div>reg m</div></div><div><div>3030</div><div>reg n</div></div></div> <p>Operands are contained in registers m and n</p> |
| 2 | Register indirect @Rn | <div><div><div>1576430</div><div>OPreg</div></div><div><div>150</div><div>16-bit register contents</div></div><div><div>150</div><div>Operand is at address indicated by register</div></div></div> | |

Table 1-11 Effective Address Calculation (2)

| No. | Addressing mode, instruction format | Effective address calculation | Effective address |
|---|---|--|--|
| 3 | Register indirect with displacement @ (d:16, Rn) | <p>16-bit register contents</p> <p>16-bit displacement</p> <p>Operand address is sum of register contents and displacement</p> | |
| 4 | Register indirect with pre-decrement @-Rn | <p>16-bit register contents</p> <p>1 or 2*</p> <p>Register is decremented before operand access</p> | |
| | Register indirect with post-increment @Rn+ | <p>16-bit register contents</p> <p>1 or 2*</p> <p>Register is incremented after operand access</p> | |
| * 1 for a byte operand, 2 for a word operand | | | |
| 5 | Absolute address @aa: 8 | None | <p>Operand address is in range from H'FF00 to H'FFFF</p> |
| | Absolute address @aa: 16 | | <p>Any address</p> |

Table 1-11 Effective Address Calculation (3)

| No. | Addressing mode instruction format | Effective address calculation | Effective address |
|-----|--|---|--|
| 6 | Immediate #xx:8. | None | |
| | <div><div>15870</div><div><div>OP</div><div>IMM</div></div></div> | | Operand is 1-byte immediatedata |
| | Immediate #xx:16 | None | |
| | <div><div>150</div><div><div>OP</div><div>IMM</div></div></div> | | Operand is 2-byte immediatedata |
| 7 | PC-relative @(d:8, PC) | | |
| | <div><div>15870</div><div><div>OP</div><div>disp</div></div></div> | <div><div>150</div><div>PC contents</div></div> <div><div>150</div><div>Sign extensiondisp</div></div> <div><div>150</div><div><div>+</div></div></div> | <div><div>150</div><div>Destinationaddress</div></div> |
| 8 | Memory indirect @@aa:8 | | |
| | <div><div>15870</div><div><div>OP</div><div>abs</div></div></div> | <div><div>15870</div><div>H'00</div></div> <div><div>150</div><div>16-bit memory contents</div></div> | <div><div>150</div><div>Destinationaddress</div></div> |

reg, regm, regn:

General register

op:

Operation field

disp

Displacement

abs:

Absolute address

IMM:

Immediate data

Section 2 Instruction Set

2.1 Explanation Format

Section 2 gives full descriptions of all the H8/300L Series instructions, presenting them in alphabetic order. Each instruction is explained in a table like the following:

| ADD (add binary) (byte) | | ADD | | | | | |
|--|-------|---|------------------|----------|----------|----------|---------------|
| Operation | | Condition Code | | | | | |
| Rd + (EAs) → Rd | | <div>I H N Z V C</div> <div><div>—</div><div>—</div><div>↑</div><div>—</div><div>↑</div><div>↑</div><div>↑</div><div>↑</div></div> | | | | | |
| Assembly-Language Format | | I: Previous value remains unchanged. | | | | | |
| ADD.B <EAs>, Rd | | H: Set to 1 when there is a carry from bit 3; otherwise cleared to 0. | | | | | |
| Operand Size | | N: Set to 1 when the result is negative; otherwise cleared to 0. | | | | | |
| Byte | | Z: Set to 1 when the result is zero; otherwise cleared to 0. | | | | | |
| | | V: Set to 1 when an overflow occurs; otherwise cleared to 0. | | | | | |
| | | C: Set to 1 when there is a carry from bit 7; otherwise cleared to 0. | | | | | |
| Description | | | | | | | |
| This instruction adds the source operand to the contents of an 8-bit general register and places the result in the general register. | | | | | | | |
| Instruction Formats and Number of Execution States | | | | | | | |
| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | ADD.B | #xx:8, Rd | 8 | rd | IMM | | 2 |
| Register direct | ADD.B | Rs, Rd | 0 | 8 | rs | rd | 2 |

The parts of the table are explained below.

Name: The full and mnemonic names of the instruction are given at the top of the page.

Operation: The instruction is described in symbolic notation. The following symbols are used.

| Symbol | Meaning |
|---------|--|
| Rd | General register (destination)* |
| Rs | General register (source)* |
| Rn | General register* |
| <EAd> | Destination operand |
| <EAs> | Source operand |
| PC | Program counter |
| SP | Stack pointer |
| CCR | Condition code register |
| N | N (negative) flag of CCR |
| Z | Z (zero) flag of CCR |
| V | V (overflow) flag of CCR |
| C | C (carry) flag of CCR |
| disp | Displacement |
| → | Transfer from left operand to right operand; or state transition from left state to right state. |
| + | Addition |
| − | Subtraction |
| × | Multiplication |
| ÷ | Division |
| ^ | AND logical |
| ∨ | OR logical |
| ⊕ | Exclusive OR logical |
| ¬ | Inverse logic (logical complement) |
| () < > | Contents of operand effective address |

* General registers are either 8 bits (R0H/R0L - R7H/R7L) or 16 bits (R0 - R7).

Assembly-Language Format:

The assembly-language coding of the instruction is given. An example is:

| | | | |
|-------------|----------|----------------------|-------------|
| <u>ADD.</u> | <u>B</u> | <u><EAs></u> , | <u>Rd</u> |
| └─┘ | └┘ | └┘ | └┘ |
| Mnemonic | Size | Source | Destination |

The operand size is indicated by the letter B (byte) or W (word). Some instructions have restrictions on the size of operands they handle.

The abbreviation EAs or EAd (effective address of source or destination) is used for operands that permit more than one addressing mode. The H8/300L CPU supports the following eight addressing modes. The method of calculating effective addresses is explained in section 1.3.4, Addressing Modes and Effective Address Calculation, above.

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

Operand size: Word or byte. Byte size is indicated for bit-manipulation instructions because these instructions access a full byte in order to read or write one bit.

Condition code: The effect of instruction execution on the flag bits in CCR is indicated. The following notation is used:

| Symbol | Meaning |
|--------|---|
| ↕ | The flag is altered according to the result of the instruction. |
| 0 | The flag is cleared to "0." |
| — | The flag is not changed. |
| * | Not fixed; the flag is left in an unpredictable state. |

Description: The action of the instruction is described in detail.

Instruction Formats: Each possible format of the instruction is shown explicitly, indicating the addressing mode, the object code, and the number of states required for execution when the instruction and its operands are located in on-chip memory. The following symbols are used:

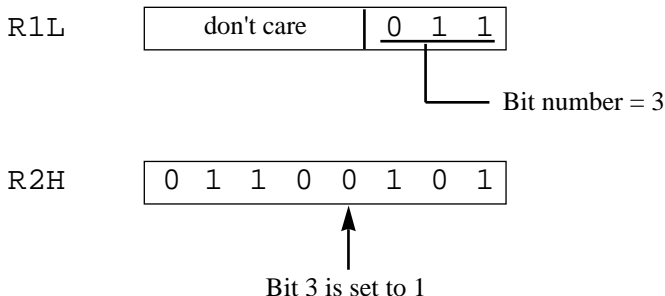
| Symbol | Meaning |
|-----------------|---|
| Imm. | Immediate data (3, 8, or 16 bits) |
| abs. | An absolute address (8 bits or 16 bits) |
| disp. | Displacement (8 bits or 16 bits) |
| r^s, r^d, r^n | General register number (3 bits or 4 bits) The s, d, and n correspond to the letters in the operand notation. |

Register Designation: 16-bit general registers are indicated by a 3-bit r^s, r^d , or r^n value. 8-bit registers are indicated by a 4-bit r^s, r^d , or r^n value. Address registers used in the @Rn, @(disp:16, Rn), @Rn+, and @-Rn addressing modes are always 16-bit registers. Data registers are 8-bit or 16-bit registers depending on the size of the operand. For 8-bit registers, the lower three bits of r^s, r^d , or r^n give the register number. The most significant bit is 1 if the lower byte of the register is used, or 0 if the upper byte is used. Registers are thus indicated as follows:

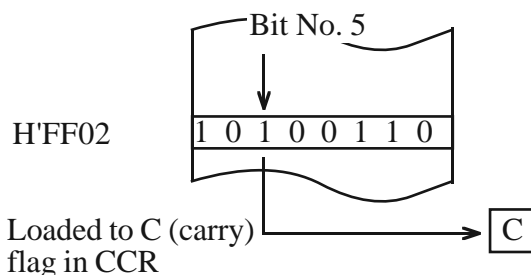
| 16-Bit register | | 16-Bit registers | |
|-----------------------------|----------|-----------------------------|----------|
| $r^s, r^d, \text{ or } r^n$ | Register | $r^s, r^d, \text{ or } r^n$ | Register |
| 0 0 0 | R0 | 0 0 0 0 | R0H |
| 0 0 1 | R1 | 0 0 0 1 | R1H |
| : | : | : | : |
| 1 1 1 | R7 | 0 1 1 1 | R7H |
| | | 1 0 0 0 | R0L |
| | | 1 0 0 1 | R1L |
| | | : | : |
| | | 1 1 1 1 | R7L |

Bit Data Access: Bit data are accessed as the n-th bit of a byte operand in a general register or memory. The bit number is given by 3-bit immediate data, or by a value in a general register. When a bit number is specified in a general register, only the lower three bits of the register are significant. Two examples are shown below.

BSET R1L, R2H



BLD #5, @H'FF02:8



The addressing mode and operand size apply to the register or memory byte containing the bit.

Number of States Required for Execution: The number of states indicated is the number required when the instruction and any memory operands are located in on-chip ROM or RAM. If the instruction or an operand is located in external memory or the on-chip register field, additional states are required for each access. See section 2.5, Number of Execution States.

2.2 Instructions

2.2.1 (1) ADD (add binary) (byte) ADD

ADD**Operation**

Rd + (EAs) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↑ | — | ↑ | ↑ |

Assembly-Language Format

ADD.B <EAs>, Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Set to 1 when there is a carry from bit 3;
otherwise cleared to 0.N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.V: Set to 1 when an overflow occurs;
otherwise cleared to 0.C: Set to 1 when there is a carry from bit 7;
otherwise cleared to 0.**Description**

This instruction adds the source operand to the contents of an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|-----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | ADD.B | #xx:8, Rd | 8 | rd | IMM | | 2 |
| Register direct | ADD.B | Rs, Rd | 0 | 8 | rs | rd | 2 |

2.2.1 (2) ADD (add binary) (word)**ADD****Operation** $Rd + Rs \rightarrow Rd$ **Condition Code**

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↑ | — | ↑ | ↑ |

Assembly-Language Format

ADD.W Rs, Rd

Operand Size

Word

I: Previous value remains unchanged.

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

This instruction adds word data in two general registers and places the result in the second general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|-----------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ADD.W | Rs, Rd | 0 9 | 0 rs 0 rd | | | 2 |

2.2.2 ADDS (add with sign extension)**ADDS****Operation**

Condition Code

Rd + 1 → Rd

Rd + 2 → Rd

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

ADDS #1, Rd

ADDS #2, Rd

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Operand Size

Word

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction adds the immediate value 1 or 2 to word data in a general register. Unlike the ADD instruction, it does not affect the condition code flags.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ADDS | #1, Rd | 0 B | 0 0 rd | | | 2 |
| Register direct | ADDS | #2, Rd | 0 B | 8 0 rd | | | 2 |

Note: This instruction cannot access byte-size data.

2.2.3 ADDX (add with extend carry)**ADDX****Operation**
 $Rd + (EAs) + C \rightarrow Rd$
Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↕ | — | ↕ | ↕ |

Assembly-Language Format

ADDX <EAs>, Rd

I: Previous value remains unchanged.

Operand Size

Byte

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

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

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

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

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

Description

This instruction adds the source operand and carry flag to the contents of an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|-----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | ADDX | #xx:8, Rd | 9 rd | IMM | | | 2 |
| Register direct | ADDX | Rs, Rd | 0 E | rs rd | | | 2 |

2.2.4 AND (AND logical)

AND

| | | | | | | | | | | | | | | |
|------------------------------------|---|---|----------------|----------------|---|---|---|---|---|---|----------------|----------------|---|---|
| Operation | Condition Code | | | | | | | | | | | | | |
| Rd \wedge (EAs) \rightarrow Rd | | | | | | | | | | | | | | |
| Assembly-Language Format | <table><tr><td>I</td><td>H</td><td>N</td><td>Z</td><td>V</td><td>C</td></tr><tr><td>—</td><td>—</td><td>—</td><td>\updownarrow</td><td>\updownarrow</td><td>0</td><td>—</td></tr></table> | I | H | N | Z | V | C | — | — | — | \updownarrow | \updownarrow | 0 | — |
| I | H | N | Z | V | C | | | | | | | | | |
| — | — | — | \updownarrow | \updownarrow | 0 | — | | | | | | | | |
| AND <EAs>, Rd | I: Previous value remains unchanged. | | | | | | | | | | | | | |
| Operand Size | H: Previous value remains unchanged. | | | | | | | | | | | | | |
| Byte | N: Set to 1 when the result is negative; otherwise cleared to 0. | | | | | | | | | | | | | |
| | Z: Set to 1 when the result is zero; otherwise cleared to 0. | | | | | | | | | | | | | |
| | V: Cleared to 0. | | | | | | | | | | | | | |
| | C: Previous value remains unchanged. | | | | | | | | | | | | | |

Description

This instruction ANDs the source operand with the contents of an 8-bit general register and places the result in the general register.

| Instruction Formats and Number of Execution States | | | | | | | |
|--|-------|-----------|------------------|----------|----------|----------|---------------|
| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | AND | #xx:8, Rd | E | rd | IMM | | 2 |
| Register direct | AND | Rs, Rd | 1 | 6 | rs | rd | 2 |

2.2.5 ANDC (AND control register)**ANDC****Operation**

Condition Code

CCR \wedge #IMM \rightarrow CCR**Assembly-Language Format**

ANDC #xx:8, CCR

Operand Size

Byte

| I | | H | | N | | Z | | V | | C |
|---|--|---|--|---|--|---|--|---|--|---|
| ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ |

I: ANDed with bit 7 of the immediate data.

H: ANDed with bit 5 of the immediate data.

N: ANDed with bit 3 of the immediate data.

Z: ANDed with bit 2 of the immediate data.

V: ANDed with bit 1 of the immediate data.

C: ANDed with bit 0 of the immediate data.

Description

This instruction ANDs the condition code register (CCR) with immediate data and places the result in the condition code register. Bits 6 and 4 are ANDed as well as the flag bits.

No interrupt requests are accepted immediately after this instruction. All interrupts, including the nonmaskable interrupt (NMI), are deferred until after the next instruction.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | ANDC | #xx:8, CCR | 0 6 | IMM | | | 2 |

2.2.6 BAND (bit AND)

BAND

| | |
|--|--|
| Operation | Condition Code |
| $C \wedge (<\text{Bit No.}> \text{ of } <\text{EAd}>) \rightarrow C$ | |
| Assembly-Language Format | |
| BAND #xx:3, <EAd> | |
| Operand Size | |
| Byte | |
| | I: Previous value remains unchanged. H: Previous value remains unchanged. N: Previous value remains unchanged. Z: Previous value remains unchanged. V: Previous value remains unchanged. C: ANDed with the specified bit. |

Description

This instruction ANDs a specified bit with the carry flag and places the result in the carry flag.

The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.

Bit No.

<EAd>*→ Byte data in register or memory

7 #xx:3 0

C

The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|-----|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BAND | #xx:3, Rd | 7 | 6 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BAND | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 6 | 0 | IMM | 0 | 6 |
| Absolute address | BAND | #xx:3, @aa:8 | 7 | E | | abs | 7 | 6 | 0 | IMM | 0 | 6 | |

* Register direct, register indirect, or absolute addressing.

Rev. 2.00 Dec 27, 2004 page 42 of 128
REJ09B0214-0200

RENESAS

2.2.7 Bcc (branch conditionally)**Bcc****Operation**

Condition Code

If cc then

PC + d:8 → PC

else next;

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

Assembly-Language Format

Bcc d:8


 Condition code field

(For mnemonics, see the table on the next page.)

I: Previous value remains unchanged.

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

—

Bcc (branch conditionally)**Bcc****Description**

If the specified condition is false, this instruction does nothing; the next instruction is executed.

If the specified condition is true, a signed displacement is added to the address of the next instruction and execution branches to the resulting address.

The displacement is a signed 8-bit value which must be even. The branch destination address can be located in the range -126 to +128 bytes from the address of the Bcc instruction.

The applicable conditions and their mnemonics are given below.

| Mnemonic | cc Field | Description | Condition | Meaning |
|-----------------|-----------------|-------------------------------|---------------------------|---------------------------------|
| BRA (BT) | 0 0 0 0 | Always (True) | Always true | |
| BRN (BF) | 0 0 0 1 | Never (False) | Never | |
| BHI | 0 0 1 0 | High | $C \vee Z = 0$ | $X > Y$ (Unsigned) |
| BLS | 0 0 1 1 | Low or Same | $C \vee Z = 1$ | $X \leq Y$ (Unsigned) |
| BCC (BHS) | 0 1 0 0 | Carry Clear (High or Same) | $C = 0$ | $X \geq Y$ (Unsigned) |
| BCS (BLO) | 0 1 0 1 | Carry Set (Low) | $C = 1$ | $X < Y$ (Unsigned) |
| BNE | 0 1 1 0 | Not Equal | $Z = 0$ | $X \neq Y$ (Signed or unsigned) |
| BEQ | 0 1 1 1 | Equal | $Z = 1$ | $X = Y$ (Signed or unsigned) |
| BVC | 1 0 0 0 | Overflow Clear | $V = 0$ | |
| BVS | 1 0 0 1 | Overflow Set | $V = 1$ | |
| BPL | 1 0 1 0 | Plus | $N = 0$ | |
| BMI | 1 0 1 1 | Minus | $N = 1$ | |
| BGE | 1 1 0 0 | Greater or Equal | $N \oplus V = 0$ | $X \geq Y$ (Signed) |
| BLT | 1 1 0 1 | Less Than | $N \oplus V = 1$ | $X < Y$ (Signed) |
| BGT | 1 1 1 0 | Greater Than | $Z \vee (N \oplus V) = 0$ | $X > Y$ (Signed) |
| BLE | 1 1 1 1 | Less or Equal | $Z \vee (N \oplus V) = 1$ | $X \leq Y$ (Signed) |

BT, BF, BHS, and BLO are synonyms for BRA, BRN, BCC, and BCS, respectively.

Bcc (branch conditionally)**Bcc****Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-----------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| PC relative | BRA (BT) | d:8 | 4 | 0 | disp. | | 4 |
| PC relative | BRN (BF) | d:8 | 4 | 1 | disp. | | 4 |
| PC relative | BHI | d:8 | 4 | 2 | disp. | | 4 |
| PC relative | BLS | d:8 | 4 | 3 | disp. | | 4 |
| PC relative | BCC (BHS) | d:8 | 4 | 4 | disp. | | 4 |
| PC relative | BCS (BLO) | d:8 | 4 | 5 | disp. | | 4 |
| PC relative | BNE | d:8 | 4 | 6 | disp. | | 4 |
| PC relative | BEQ | d:8 | 4 | 7 | disp. | | 4 |
| PC relative | BVC | d:8 | 4 | 8 | disp. | | 4 |
| PC relative | BVS | d:8 | 4 | 9 | disp. | | 4 |
| PC relative | BPL | d:8 | 4 | A | disp. | | 4 |
| PC relative | BMI | d:8 | 4 | B | disp. | | 4 |
| PC relative | BGE | d:8 | 4 | C | disp. | | 4 |
| PC relative | BLT | d:8 | 4 | D | disp. | | 4 |
| PC relative | BGT | d:8 | 4 | E | disp. | | 4 |
| PC relative | BLE | d:8 | 4 | F | disp. | | 4 |

* The branch address must be even.

2.2.8 BCLR (bit clear)

BCLR

Operation

0 → (<Bit No.> of <EAd>)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BCLR #xx:3, <EAd>

BCLR Rn, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

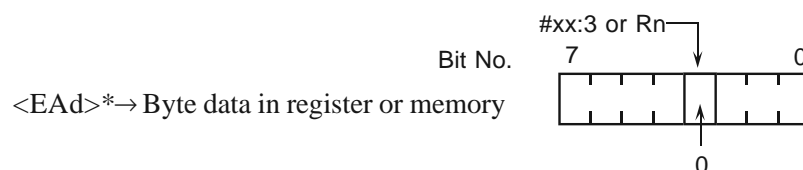
V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction 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 general register. The destination operand can be located in a general register or memory.

The specified bit is not tested before being cleared. The condition code flags are not altered.



Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|----|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BCLR | #xx:3, Rd | 7 | 2 | 0 | IMM | rd | | | | | 2 | |
| Register indirect | BCLR | #xx:3, @Rd | 7 | D | 0 | rd | 0 | 7 | 2 | 0 | IMM | 0 | 8 |
| Absolute address | BCLR | #xx:3, @aa:8 | 7 | F | abs | | | 7 | 2 | 0 | IMM | 0 | 8 |
| Register direct | BCLR | Rn, Rd | 6 | 2 | rn | rd | | | | | | | 2 |
| Register indirect | BCLR | Rn, @Rd | 7 | D | 0 | rd | 0 | 6 | 2 | rn | 0 | | 8 |
| Absolute address | BCLR | Rn, @aa:8 | 7 | F | abs | | | 6 | 2 | rn | 0 | | 8 |

* Register direct, register indirect, or absolute addressing.

2.2.9 BIAN (bit invert AND)

BIAND

Operation

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

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↑ |

Assembly-Language Format

BIAND #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

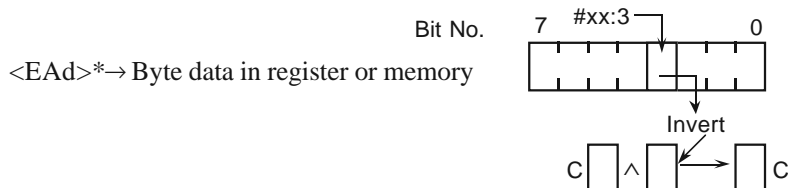
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: ANDed with the inverse of the specified bit.

Description

This instruction ANDs the inverse of a specified bit with the carry flag and places the result in the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-------------------|-------|--------------|------------------|------------|----------|-----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | BIAND | #xx:3, Rd | 7 6 | 1:IMM rd | | | 2 |
| Register indirect | BIAND | #xx:3, @Rd | 7 C | 0:rd 0 | 7 6 | 1:IMM 0 | 6 |
| Absolute address | BIAND | #xx:3, @aa:8 | 7 E | abs | 7 6 | 1:IMM 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.10 BILD (bit invert load)

BILD

Operation

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

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↑ |

Assembly-Language Format

BILD #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

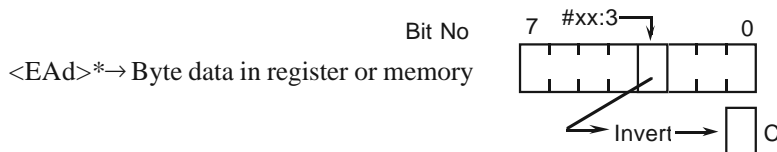
V: Previous value remains unchanged.

C: Loaded with the inverse of the specified bit.

Description

This instruction loads the inverse of a specified bit into the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data.

The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BILD | #xx:3, Rd | 7 | 7 | 1 | IMM | rd | | | | | 2 | |
| Register indirect | BILD | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 7 | 1 | IMM | 0 | 6 |
| Absolute address | BILD | #xx:3, @aa:8 | 7 | E | | abs | | 7 | 7 | 1 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.11 BIOR (bit invert inclusive OR)**BIOR****Operation** $C \vee [\neg (<\text{Bit No.}> \text{ of } <\text{EAd}>)] \rightarrow C$ **Condition Code**

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↑ |

Assembly-Language Format

BIOR #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

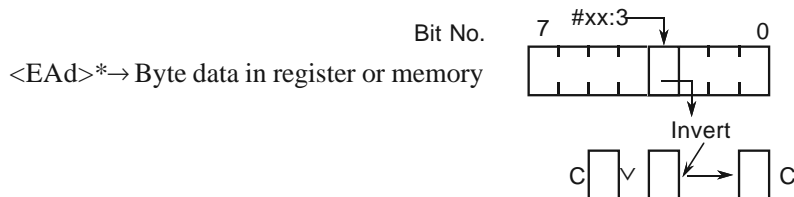
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: ORed with the inverse of the specified bit.

Description

This instruction ORs the inverse of a specified bit with the carry flag and places the result in the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BIOR | #xx:3, Rd | 7 | 4 | 1 | IMM | rd | | | | | 2 | |
| Register indirect | BIOR | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 4 | 1 | IMM | 0 | 6 |
| Absolute address | BIOR | #xx:3, @aa:8 | 7 | E | | abs | | 7 | 4 | 1 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.12 BIST (bit invert store)

BIST

Operation

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

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BIST #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

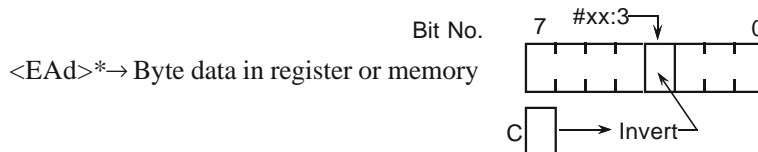
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction stores the inverse of the carry flag to a specified bit location in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The values of the unspecified bits are not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-------------------|-------|--------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | BIST | #xx:3, Rd | 6 7 | 1 IMM rd | | | 2 |
| Register indirect | BIST | #xx:3, @Rd | 7 D | 0 rd 0 | 6 7 | 1 IMM 0 | 8 |
| Absolute address | BIST | #xx:3, @aa:8 | 7 F | abs | 6 7 | 1 IMM 0 | 8 |

* Register direct, register indirect, or absolute addressing.

2.2.13 BIXOR (bit invert exclusive OR)**BIXOR****Operation**
 $C \oplus [\neg (\text{<Bit No.> of <EAd>})] \rightarrow C$
Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↑ |

Assembly-Language Format

BIXOR #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

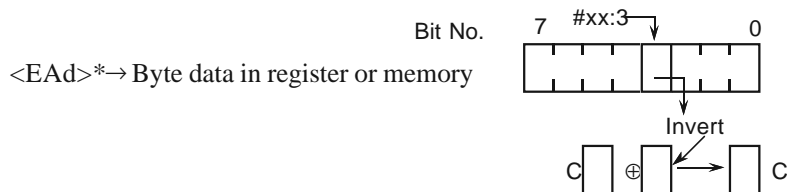
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Exclusive-ORed with the inverse of the specified bit.

Description

This instruction exclusive-ORs the inverse of a specified bit with the carry flag and places the result in the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BIXOR | #xx:3, Rd | 7 | 5 | 1 | IMM | rd | | | | | 2 | |
| Register indirect | BIXOR | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 5 | 1 | IMM | 0 | 6 |
| Absolute address | BIXOR | #xx:3, @aa:8 | 7 | E | | abs | | 7 | 5 | 1 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.14 BLD (bit load)

BLD

Operation
(<Bit No.> of <EAd>) → C

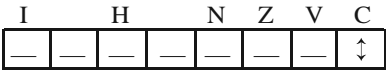
Assembly-Language Format

BLD #xx:3, <EAd>

Operand Size

Byte

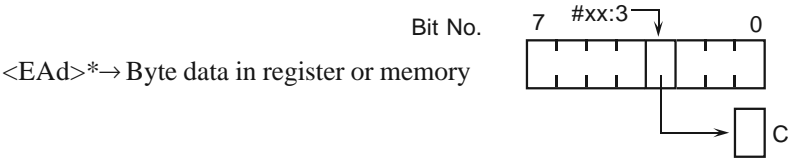
Condition Code



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

Description

This instruction loads a specified bit into the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BLD | #xx:3, Rd | 7 | 7 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BLD | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 7 | 0 | IMM | 0 | 6 |
| Absolute address | BLD | #xx:3, @aa:8 | 7 | E | abs | | | 7 | 7 | 0 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.15 BNOT (bit NOT)

BNOT

Operation

¬ (<Bit No.> of <EAd>)

→ (<Bit No.> of <EAd>)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BNOT #xx:3, <EAd>

BNOT Rn, <EAd>

I: Previous value remains unchanged.

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.

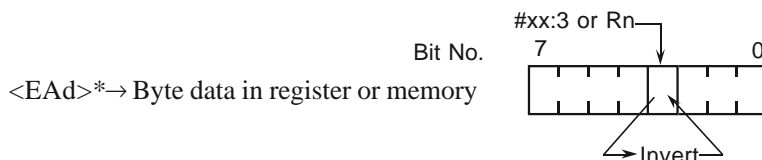
Operand Size

Byte

Description

This instruction inverts a specified bit in a general register or memory location. The bit number is specified by 3-bit immediate data, or by the lower three-bits of a general register.

The operation is shown schematically below.



The bit is not tested before being inverted. The condition code flags are not altered.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|----|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BNOT | #xx:3, Rd | 7 | 1 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BNOT | #xx:3, @Rd | 7 | D | 0 | rd | 0 | 7 | 1 | 0 | IMM | 0 | 8 |
| Absolute address | BNOT | #xx:3, @aa:8 | 7 | F | abs | | | 7 | 1 | 0 | IMM | 0 | 8 |
| Register direct | BNOT | Rn, Rd | 6 | 1 | rn | rd | | | | | | | 2 |
| Register indirect | BNOT | Rn, @Rd | 7 | D | 0 | rd | 0 | 6 | 1 | rn | | 0 | 8 |
| Absolute address | BNOT | Rn, @aa:8 | 7 | F | abs | | | 6 | 1 | rn | | 0 | 8 |

* Register direct, register indirect, or absolute addressing.

2.2.16 BOR (bit inclusive OR)

BOR

Operation

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

Assembly-Language Format

BOR #xx:3, <EAd>

Operand Size

Byte

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↑ |

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

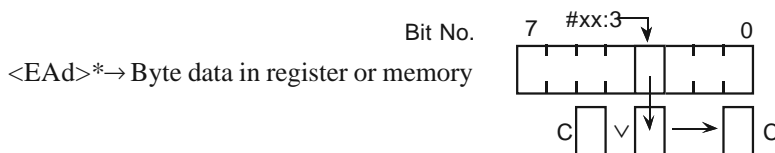
V: Previous value remains unchanged.

C: ORed with the specified bit.

Description

This instruction ORs a specified bit with the carry flag and places the result in the carry flag.

The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BOR | #xx:3, Rd | 7 | 4 | 0 | IMM | rd | | | | | 2 | |
| Register indirect | BOR | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 4 | 0 | IMM | 0 | 6 |
| Absolute address | BOR | #xx:3, @aa:8 | 7 | E | | abs | | 7 | 4 | 0 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.17 BSET (bit set)**BSET****Operation**

1 → (<Bit No.> of <EAd>)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BSET #xx:3,<EAd>

BSET Rn,<EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

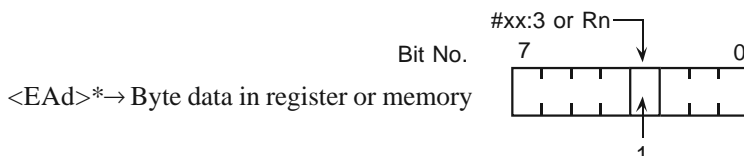
V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction 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 general register. The destination operand can be located in a general register or memory.

The specified bit is not tested before being cleared. The condition code flags are not altered.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|-----|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BSET | #xx:3, Rd | 7 | 0 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BSET | #xx:3, @Rd | 7 | D | 0 | rd | 0 | 7 | 0 | 0 | IMM | 0 | 8 |
| Absolute address | BSET | #xx:3, @aa:8 | 7 | F | abs | | 7 | 0 | 0 | IMM | 0 | 8 | |
| Register direct | BSET | Rn, Rd | 6 | 0 | rn | rd | | | | | 2 | | |
| Register indirect | BSET | Rn, @Rd | 7 | D | 0 | rd | 0 | 6 | 0 | rn | 0 | 8 | |
| Absolute address | BSET | Rn, @aa:8 | 7 | F | abs | | 6 | 0 | rn | 0 | 8 | | |

* Register direct, register indirect, or absolute addressing.

2.2.18 BSR (branch to subroutine)**BSR****Operation**

Condition Code

PC → @-SP

PC + d:8 → PC

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BSR d:8

I: Previous value remains unchanged.

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 (PC) value onto the stack, then adds a specified displacement to the program counter value and branches to the resulting address. The program counter value used is the address of the instruction following the BSR instruction.

The displacement is a signed 8-bit value which must be even. The possible branching range is –126 to +128 bytes from the address of the BSR instruction.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| PC-relative | BSR | d:8 | 5 | 5 | disp | | 6 |

2.2.19 BST (bit store)**BST****Operation**

C → (<Bit No.> of <EAd>)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

BST #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

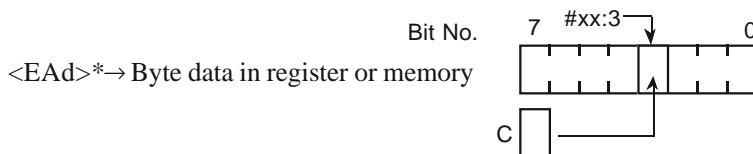
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction stores the carry flag to a specified flag location in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | No. of states | | | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|---------------|----------|-----|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | | 4th byte | | | |
| Register direct | BST | #xx:3, Rd | 6 | 7 | 0 | IMM | rd | | | 2 | | | |
| Register indirect | BST | #xx:3, @Rd | 7 | D | 0 | rd | 0 | 6 | 7 | 0 | IMM | 0 | 8 |
| Absolute address | BST | #xx:3, @aa:8 | 7 | F | abs | | | 6 | 7 | 0 | IMM | 0 | 8 |

* Register direct, register indirect, or absolute addressing.

2.2.20 BTST (bit test)**BTST****Operation** $\neg (<\text{Bit No.}> \text{ of } <\text{EAd}>) \rightarrow Z$ **Condition Code**

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↑ | — | — |

Assembly-Language Format

BTST #xx:3, <EAd>

BTST Rn, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

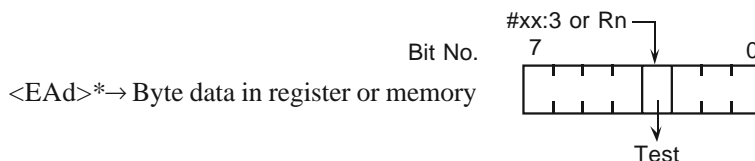
Z: Set to 1 when the specified bit is zero;
otherwise cleared to 0.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction tests a specified bit in a general register or memory location and sets or clears the Zero flag accordingly. The bit number can be specified by 3-bit immediate data, or by the lower three bits of an 8-bit general register. The operation is shown schematically below.



The value of the specified bit is not altered.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BTST | #xx:3, Rd | 7 | 3 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BTST | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 3 | 0 | IMM | 0 | 6 |
| Absolute address | BTST | #xx:3, @aa:8 | 7 | E | | abs | | 7 | 3 | 0 | IMM | 0 | 6 |
| Register direct | BTST | Rn, Rd | 6 | 3 | | rn | rd | | | | | | 2 |
| Register indirect | BTST | Rn, @Rd | 7 | C | 0 | rd | 0 | 6 | 3 | | rn | 0 | 6 |
| Absolute address | BTST | Rn, @aa:8 | 7 | E | | abs | | 6 | 3 | | rn | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.21 BXOR (bit exclusive OR)**BXOR****Operation**
 $C \oplus (\text{<Bit No.> of <EAd>}) \rightarrow C$
Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | ↕ |

Assembly-Language Format

BXOR #xx:3, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

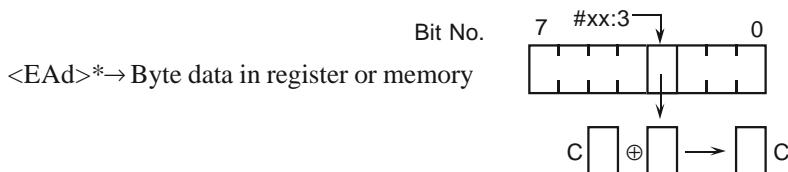
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Exclusive-ORed with the specified bit.

Description

This instruction exclusive-ORs a specified bit with the carry flag and places the result in the carry flag. The specified bit can be located in a general register or memory. The bit number is specified by 3-bit immediate data. The operation is shown schematically below.



The value of the specified bit is not changed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | | | No. of states | | |
|-------------------|-------|--------------|------------------|---|----------|-----|----------|---|----------|---|---------------|---|---|
| | | | 1st byte | | 2nd byte | | 3rd byte | | 4th byte | | | | |
| Register direct | BXOR | #xx:3, Rd | 7 | 5 | 0 | IMM | rd | | | | 2 | | |
| Register indirect | BXOR | #xx:3, @Rd | 7 | C | 0 | rd | 0 | 7 | 5 | 0 | IMM | 0 | 6 |
| Absolute address | BXOR | #xx:3, @aa:8 | 7 | E | abs | | | 7 | 5 | 0 | IMM | 0 | 6 |

* Register direct, register indirect, or absolute addressing.

2.2.22 (1) CMP (compare) (byte)**CMP****Operation**

Rd – (EAs); set condition code

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↑ | — | ↑ | ↑ |

Assembly-Language Format

CMP.B <EAs>, Rd

Operand Size

Byte

I: Previous value remains unchanged.

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

This instruction subtracts an 8-bit source register or immediate data from an 8-bit destination register and sets the condition code flags according to the result. The destination register is not altered.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | CMP.B | #xx:8,Rd | A rd | IMM | | | 2 |
| Register direct | CMP.B | Rs, Rd | 1 C | rs rd | | | 2 |

2.2.22 (2) CMP (compare) (word)**CMP****Operation**

Rd – Rs; set condition code

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↕ | — | ↕ | ↕ |

Assembly-Language Format

CMP.W Rs, Rd

I: Previous value remains unchanged.

Operand Size

Word

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

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

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

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

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

Description

This instruction subtracts a source register from a destination register and sets the condition code flags according to the result. The destination register is not altered.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|-----------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | CMP.W | Rs, Rd | 1 D | 0 rs 0 rd | | | 2 |

2.2.23 DAA (decimal adjust add)**DAA****Operation**

Condition Code

Rd (decimal adjust) → Rd

Assembly-Language Format

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | * | — | ↑ | ↑ |

DAA Rd

I: Previous value remains unchanged.

Operand Size

H: Unpredictable.

Byte

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

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

V: Unpredictable.

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

Description

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

Valid results are not assured if this instruction is executed under conditions other than those stated above.

| Status before adjustment | | | | Value added | Resulting C flag |
|--------------------------|--------------|--------|--------------|-------------|------------------|
| C flag | Upper nibble | H flag | Lower nibble | | |
| 0 | 0 – 9 | 0 | 0 – 9 | H'00 | 0 |
| 0 | 0 – 8 | 0 | A – F | H'06 | 0 |
| 0 | 0 – 9 | 1 | 0 – 3 | H'06 | 0 |
| 0 | A – F | 0 | 0 – 9 | H'60 | 1 |
| 0 | 9 – F | 0 | A – F | H'66 | 1 |
| 0 | A – F | 1 | 0 – 3 | H'66 | 1 |
| 1 | 0 – 2 | 0 | 0 – 9 | H'60 | 1 |
| 1 | 0 – 2 | 0 | A – F | H'66 | 1 |
| 1 | 0 – 3 | 1 | 0 – 3 | H'66 | 1 |

DAA (decimal adjust add)**DAA****Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | DAA | Rd | 0 | F | 0 | rd | 2 |

2.2.24 DAS (decimal adjust subtract)**DAS****Operation**

Condition Code

Rd (decimal adjust) → Rd

Assembly-Language Format

DAS Rd

Operand Size

Byte

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | * | — | ↑ | — |

I: Previous value remains unchanged.

H: Unpredictable.

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

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

V: Unpredictable.

C: Previous value remains unchanged.

Description

When the result of a subtraction operation performed by the SUB.B, SUBX, or NEG instruction on 4-bit BCD data is contained in an 8-bit general register and the carry and half-carry flags, the DAA instruction adjusts the result by adding H'00, H'FA, H'A0, or H'9A to the general register according to the table below.

Valid results are not assured if this instruction is executed under conditions other than those stated above.

| Status before adjustment | | | | Value added | Resulting C flag |
|--------------------------|--------------|--------|--------------|-------------|------------------|
| C flag | Upper nibble | H flag | Lower nibble | | |
| 0 | 0 – 9 | 0 | 0 – 9 | H'00 | 0 |
| 0 | 0 – 8 | 1 | 6 – F | H'FA | 0 |
| 1 | 7 – F | 0 | 0 – 9 | H'A0 | 1 |
| 1 | 6 – F | 1 | 6 – F | H'9A | 1 |

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | DAS | Rd | 1 F | 0 rd | | | 2 |

2.2.25 DEC (decrement)**DEC****Operation**

Rd – 1 → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↓ | ↓ | ↓ |

Assembly-Language Format

DEC Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.V: Set to 1 when an overflow occurs (the previous value in
Rd was H'80); otherwise cleared to 0.

C: Previous value remains unchanged.

Description

This instruction decrements an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | DEC | Rd | 1 A | 0 rd | | | 2 |

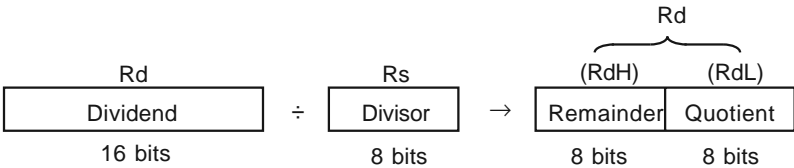
2.2.26 DIVXU (divide extend as unsigned)

DIVXU

| | | | | | | | | | | | | | |
|---------------------------------|---|---|-----|-----|---|---|---|---|---|---|-----|-----|---|
| Operation | Condition Code | | | | | | | | | | | | |
| $Rd \div Rs \rightarrow Rd$ | | | | | | | | | | | | | |
| Assembly-Language Format | <table><tr><td>I</td><td>H</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></tr></table> | I | H | N | Z | V | C | — | — | — | ↑ ↓ | ↑ ↓ | — |
| I | H | N | Z | V | C | | | | | | | | |
| — | — | — | ↑ ↓ | ↑ ↓ | — | | | | | | | | |
| DIVXU Rs, Rd | I: Previous value remains unchanged. | | | | | | | | | | | | |
| Operand Size | H: Previous value remains unchanged. | | | | | | | | | | | | |
| Byte | N: Set to 1 when the divisor is negative; otherwise cleared to 0. | | | | | | | | | | | | |
| | Z: Cleared to 0 when divisor $\neq 0$; otherwise not guaranteed. | | | | | | | | | | | | |
| | V: Previous value remains unchanged. | | | | | | | | | | | | |
| | C: Previous value remains unchanged. | | | | | | | | | | | | |

Description

This instruction divides a 16-bit general register by an 8-bit general register and places the result in the 16-bit general register. The quotient is placed in the lower byte. The remainder is placed in the upper byte. The operation is shown schematically below.



Valid results (Rd, N, Z) are not assured if division by zero is attempted or an overflow occurs. Division by zero is indicated in the Zero flag. Overflow can be avoided by the coding shown on the next page.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | DIVXU | Rs, Rd | 5 | 1 | rs | 0 rd | 14 |

DIVXU (divide extend as unsigned)**DIVXU****Note: DIVXU Overflow**

Since the DIVXU instruction performs 16-bit ÷ 8-bit → 8-bit division, an overflow will occur if the divisor byte is equal to or less than the upper byte of the dividend. For example, H'FFFF ÷ H'01 → H'FFFF causes an overflow. (The quotient has more than 8 bits.)

Overflows can be avoided by using a subprogram like the following. A work register is required.

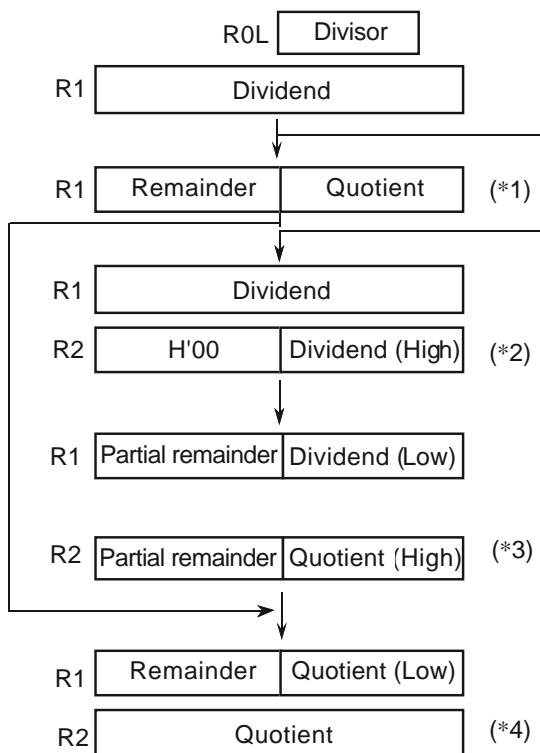
To perform

DIVXU R0L, R1:

```

MOV.B #H'00, R2H
CMP.B R0L, R1H
BCC L1
DIVXU R0L, R1      (*1)
MOV.B R1L, R2L
BRA L2
L1 MOV.B R1H, R2L  (*2)
   DIVXU R0L, R2
   MOV.B R2H, R1H  (*3)
   DIVXU R0L, R1
   MOV.B R2L, R2H
   MOV.B R1L, R2L
L2  RTS           (*4)

```



2.2.27 EEPMOV (move data to EEPROM)**EEPMOV****Operation**

Condition Code

if R4L \neq 0 thenrepeat @R5+ \rightarrow @R6+R4L - 1 \rightarrow R4L

until R4L = 0

else next;

Assembly-Language Format

EEPMOV

Operand Size

—

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

I: Previous value remains unchanged.

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 a block of data from the memory location specified in general register R5 to the memory location specified in general register R6. General register R4L gives the byte length of the block.

Data are transferred a byte at a time. After each byte transfer, R5 and R6 are incremented and R4L is decremented. When R4L reaches 0, the transfer ends and the next instruction is executed. No interrupt requests are accepted during the data transfer.

At the end of this instruction, R4L contains H'00. R5 and R6 contain the last transfer address +1.

The memory locations specified by general registers R5 and R6 are read before the block transfer is performed.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|--------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | EEPMOV | | 7 B | 5 C | 5 9 | 8 F | 9+4n* |

* n is the initial value in R4L ($0 \leq n \leq 255$). Although n bytes of data are transferred, memory is accessed 2 (n + 1) times, requiring 4 (n + 1) states.

2.2.28 INC (increment)**INC****Operation**

Rd + 1 → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | ↓ | — |

Assembly-Language Format

INC Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.V: Set to 1 when an overflow occurs (the previous value in
Rd was H'7F); otherwise cleared to 0.

C: Previous value remains unchanged.

Description

This instruction increments an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | INC | Rd | 0 A | 0 rd | | | 2 |

2.2.29 JMP (jump)**JMP****Operation**

(EAd) → PC

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

JMP <EA>

Operand Size

—

I: Previous value remains unchanged.

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 branches unconditionally to a specified destination address.

The destination address must be even.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-------------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register indirect | JMP | @Rn | 5 9 | 0 rn 0 | | | 4 |
| Absolute address | JMP | @aa:16 | 5 A | 0 0 | abs. | | 6 |
| Memory indirect | JMP | @@aa:8 | 5 B | abs. | | | 8 |

2.2.30 JSR (Jump to subroutine)**JSR****Operation**

PC → @-SP

(EAd) → PC

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

JSR <EA>

Operand Size

—

I: Previous value remains unchanged.

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 pushes the program counter onto the stack, then branches to a specified destination address. The program counter value pushed on the stack is the address of the instruction following the JSR instruction. The destination address must be even.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-------------------|-------|----------|------------------|--------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register indirect | JSR | @Rn | 5 | D 0 rn 0 | | | 6 |
| Absolute address | JSR | @aa:16 | 5 | E 0 0 | abs. | | 8 |
| Memory indirect | JSR | @ @aa:8 | 5 | F | abs. | | 8 |

2.2.31 LDC (load to control register) LDC

Operation

(EAs) → CCR

Condition Code

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |

Assembly-Language Format

LDC <EAs>, CCR

I: Loaded from the source operand.

Operand Size

Byte

H: Loaded from the source operand.

N: Loaded from the source operand.

Z: Loaded from the source operand.

V: Loaded from the source operand.

C: Loaded from the source operand.

Description

This instruction loads the source operand contents into the condition code register (CCR). Bits 4 and 6 are loaded as well as the flag bits.

No interrupt requests are accepted immediately after this instruction. All interrupts are deferred until after the next instruction.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | LDC | #xx:8, CCR | 0 | 7 | IMM | | 2 |
| Register direct | LDC | Rs, CCR | 0 | 3 | 0 | rs | 2 |

2.2.32 (1) MOV (move data) (byte)**MOV****Operation**

Rs → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.B Rs, Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction moves one byte of data from a source register to a destination register and sets condition code flags according to the data value.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | MOV.B | Rs, Rd | 0 C | rs rd | | | 2 |

2.2.32 (2) MOV (move data) (word)**MOV****Operation**

Rs → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.W Rs, Rd

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Operand Size

Word

Description

This instruction moves one word of data from a source register to a destination register and sets condition code flags according to the data value.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|-----------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | MOV.W | Rs, Rd | 0 D | 0 rs 0 rd | | | 2 |

2.2.32 (3) MOV (move data) (byte)**MOV****Operation**

(EAs) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.B <EAs>, Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction moves one byte of data from a source operand to a destination register and sets condition code flags according to the data value.

The MOV.B @R7+, Rd instruction should never be used, because it leaves an odd value in the stack pointer. See section 3.2.3 for details.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|---------------------------------------|-------|---------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | MOV.B | #xx:8, Rd | F | rd | IMM | | 2 |
| Register indirect | MOV.B | @RS, Rd | 6 | 8 | 0 rs | rd | 4 |
| Register indirect with displacement | MOV.B | @(d:16,Rs),Rd | 6 | E | 0 rs | rd | 6 |
| Register indirect with post-increment | MOV.B | @Rs+, Rd | 6 | C | 0 rs | rd | 6 |
| Absolute address | MOV.B | @aa:8, Rd | 2 | rd | abs | | 4 |
| Absolute address | MOV.B | @aa:16, Rd | 6 | A | 0 | rd | 6 |

2.2.32 (4) MOV (move data) (word)**MOV****Operation**

(EAs) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.W <EAs>, Rd

Operand Size

Word

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction moves one word of data from a source operand to a destination register and sets condition code flags according to the data value.

If the source operand is in memory, it must be located at an even address.

MOV.W @R7+, Rd is identical in machine language to POP.W Rd.

Note that the LSIs in the H8/300L Series contain on-chip peripheral modules for which access in word size is not possible. Details are given in the applicable hardware manual.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|---------------------------------------|-------|---------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | MOV.W | #xx:16, Rd | 7 | 9 | 0 | 0:rd | 4 |
| Register indirect | MOV.W | @RS, Rd | 6 | 9 | 0 | rs:0:rd | 4 |
| Register indirect with displacement | MOV.W | @(d:16,Rs),Rd | 6 | F | 0 | rs:0:rd | 6 |
| Register indirect with post-increment | MOV.W | @Rs+, Rd | 6 | D | 0 | rs:0:rd | 6 |
| Absolute address | MOV.W | @aa:16, Rd | 6 | B | 0 | 0:rd | 6 |

2.2.32 (5) MOV (move data) (byte)**MOV****Operation**

Rs → (EAd)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.B Rs, <EAd>

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction moves one byte of data from a source register to memory and sets condition code flags according to the data value.

The MOV.B Rs, @-R7 instruction should never be used, because it leaves an odd value in the stack pointer. See section 3.2.3, Outline of Exception Handling Operation, for details.

The instruction MOV.B RnH, @-Rn or MOV.B RnL, @-Rn decrements register Rn, then moves the upper or lower byte of the decremented result to memory.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | No. of states | |
|--------------------------------------|-------|----------------|------------------|----|----------|----|----------|-------|---------------|----------|
| | | | 1st byte | | 2nd byte | | 3rd byte | | | 4th byte |
| Register indirect | MOV.B | Rs, @Rd | 6 | 8 | 1 | rd | rs | | | 4 |
| Register indirect with displacement | MOV.B | Rs, @(d:16,Rd) | 6 | E | 1 | rd | rs | disp. | | 6 |
| Register indirect with pre-decrement | MOV.B | Rs, @-Rd | 6 | C | 1 | rd | rs | | | 6 |
| Absolute address | MOV.B | Rs, @aa:8 | 3 | rs | abs | | | | | 4 |
| Absolute address | MOV.B | Rs, @aa:16 | 6 | A | 8 | rs | abs. | | | 6 |

2.2.32 (6) MOV (move data) (word)**MOV****Operation**

Rs → (EAd)

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

MOV.W Rs, <EAd>

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Operand Size

Word

Description

This instruction moves one word of data from a general register to memory and sets condition code flags according to the data value.

The destination address in memory must be even.

MOV.W Rs, @-R7 is identical in machine language to PUSH.W Rs.

The instruction MOV.W Rn, @-Rn decrements register Rn by 2, then moves the decremented result to memory.

Note that the LSIs in the H8/300L Series contain on-chip peripheral modules for which access in word size is not possible. Details are given in the applicable hardware manual.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|--------------------------------------|-------|-----------------|------------------|----------|-----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register indirect | MOV.W | Rs, @Rd | 6 | 9 | 1 rd 0 rs | | 4 |
| Register indirect with displacement | MOV.W | Rs, @(d:16, Rd) | 6 | F | 1 rd 0 rs | disp. | 6 |
| Register indirect with pre-decrement | MOV.W | Rs, @-Rd | 6 | D | 1 rd 0 rs | | 6 |
| Absolute address | MOV.W | Rs, @aa:16 | 6 | B | 8 0 rs | abs. | 6 |

2.2.33 MULXU (multiply extend as unsigned)**MULXU****Operation** $Rd \times Rs \rightarrow Rd$ **Condition Code**

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

MULXU Rs, Rd

I: Previous value remains unchanged.

Operand Size

Byte

H: Previous value remains unchanged.

N: Previous value remains unchanged.

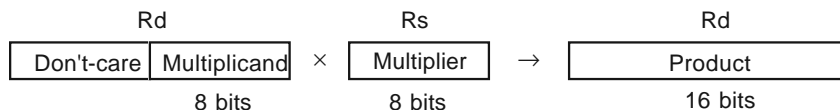
Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction performs 8-bit \times 8-bit \rightarrow 16-bit multiplication. It multiplies a destination register by a source register and places the result in the destination register. The source register is an 8-bit register. The destination register is a 16-bit register containing the data to be multiplied in the lower byte. (The upper byte is ignored.) The result is placed in both bytes of the destination register. The operation is shown schematically below.



The multiplier can occupy either the upper or lower byte of the source register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|-------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | MULXU | Rs, Rd | 5 0 | rs 0 rd | | | 14 |

2.2.34 NEG (negate)

NEG

| | | | | | | | | | | | | | |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operation | Condition Code | | | | | | | | | | | | |
| 0 – Rd → Rd | | | | | | | | | | | | | |
| Assembly-Language Format | <table><tr><td>I</td><td>H</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></tr></table> | I | H | N | Z | V | C | — | — | ↑ | — | ↑ | ↑ |
| I | H | N | Z | V | C | | | | | | | | |
| — | — | ↑ | — | ↑ | ↑ | | | | | | | | |
| NEG Rd | I: Previous value remains unchanged. | | | | | | | | | | | | |
| Operand Size | H: Set to 1 when there is a borrow from bit 3; otherwise cleared to 0. | | | | | | | | | | | | |
| Byte | N: Set to 1 when the result is negative; otherwise cleared to 0. | | | | | | | | | | | | |
| | Z: Set to 1 when the result is zero; otherwise cleared to 0. | | | | | | | | | | | | |
| | V: Set to 1 when an overflow occurs (the previous contents of the destination register was H'80); otherwise cleared to 0. | | | | | | | | | | | | |
| | C: Set to 1 when there is a borrow from bit 7 (the previous contents of the destination register was not H'00); otherwise cleared to 0. | | | | | | | | | | | | |

Description

This instruction replaces the contents of an 8-bit general register with its two's complement (subtracts the register contents from H'00).

If the original contents of the destination register was H'80, the register value remains H'80 and the overflow flag is set.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | NEG | Rd | 1 7 | 8 rd | | | 2 |

2.2.35 NOP (no operation)**NOP****Operation**

PC + 2 → PC

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

Assembly-Language Format

NOP

I: Previous value remains unchanged.

Operand Size

—

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

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | NOP | | 0 0 | 0 0 | | | 2 |

2.2.36 NOT (NOT = logical complement)**NOT****Operation** $\neg \text{Rd} \rightarrow \text{Rd}$ **Condition Code**

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

NOT Rd

I: Previous value remains unchanged.

Operand Size

Byte

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction replaces the contents of an 8-bit general register with its one's complement (subtracts the register contents from H'FF).

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | NOT | Rd | 1 | 7 | 0 | rd | 2 |

2.2.37 OR (inclusive OR logical)**OR****Operation**

Condition Code

Rd \vee (EAs) \rightarrow Rd**Assembly-Language Format**

OR <EAs>, Rd

Operand Size

Byte

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | 0 | — |

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction ORs the source operand with the contents of an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|-----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | OR | #xx:8, Rd | C rd | IMM | | | 2 |
| Register direct | OR | Rs, Rd | 1 4 | rs rd | | | 2 |

2.2.38 ORC (inclusive OR control register)**ORC****Operation**

Condition Code

CCR \vee #IMM \rightarrow CCR**Assembly-Language Format**

ORC #xx:8, CCR

Operand Size

Byte

| I | | H | | N | | Z | | V | | C |
|---|--|---|--|---|--|---|--|---|--|---|
| ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ |

I: ORed with bit 7 of the immediate data.

H: ORed with bit 5 of the immediate data.

N: ORed with bit 3 of the immediate data.

Z: ORed with bit 2 of the immediate data.

V: ORed with bit 1 of the immediate data.

C: ORed with bit 0 of the immediate data.

Description

This instruction ORs the condition code register (CCR) with immediate data and places the result in the condition code register. Bits 6 and 4 are ORed as well as the flag bits.

No interrupt requests are accepted immediately after this instruction. All interrupts are deferred until after the next instruction.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | ORC | #xx:8, CCR | 0 | 4 | IMM | | 2 |

2.2.39 POP (pop data)**POP****Operation**

@SP+ → Rn

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | — |

Assembly-Language Format

POP Rn

Operand Size

Word

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the data value is negative;
otherwise cleared to 0.Z: Set to 1 when the data value is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction pops data from the stack to a 16-bit general register and sets condition code flags according to the data value.

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

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | POP | Rd | 6 D | 7 0; rn | | | 6 |

2.2.40 **PUSH (push data)**

PUSH

| | | | | | | | | | | | | | | | | | | | |
|---------------------------------|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operation | Condition Code | | | | | | | | | | | | | | | | | | |
| Rn → @-SP | | | | | | | | | | | | | | | | | | | |
| Assembly-Language Format | <table><tr><td>I</td><td>H</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>0</td></tr><tr><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr></table> | I | H | N | Z | V | C | — | — | — | ↑ | ↑ | 0 | — | — | — | — | — | — |
| I | H | N | Z | V | C | | | | | | | | | | | | | | |
| — | — | — | ↑ | ↑ | 0 | | | | | | | | | | | | | | |
| — | — | — | — | — | — | | | | | | | | | | | | | | |
| PUSH Rn | I: Previous value remains unchanged. | | | | | | | | | | | | | | | | | | |
| Operand Size | H: Previous value remains unchanged. | | | | | | | | | | | | | | | | | | |
| Word | N: Set to 1 when the data value is negative; otherwise cleared to 0. | | | | | | | | | | | | | | | | | | |
| | Z: Set to 1 when the data value is zero; otherwise cleared to 0. | | | | | | | | | | | | | | | | | | |
| | V: Cleared to 0. | | | | | | | | | | | | | | | | | | |
| | C: Previous value remains unchanged. | | | | | | | | | | | | | | | | | | |

Description

This instruction pushes data from a 16-bit general register onto the stack and sets condition code flags according to the data value.

PUSH.W Rn is identical in machine language to MOV.W Rn, @-SP.

| Instruction Formats and Number of Execution States | | | | | | | |
|--|-------|----------|------------------|----------|----------|----------|---------------|
| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | PUSH | Rs | 6 | D | F | 0; rn | 6 |

2.2.41 ROTL (rotate left)**ROTL****Operation**

Rd (rotated left) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | ↓ |

Assembly-Language Format

ROTL Rd

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Receives the previous value in bit 7.

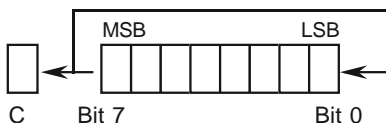
Operand Size

Byte

Description

This instruction rotates an 8-bit general register one bit to the left. The most significant bit is rotated to the least significant bit, and also copied to the carry flag.

The operation is shown schematically below.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ROTL | Rd | 1 | 2 | 8 | rd | 2 |

2.2.42 ROTR (rotate right)

ROTR

Operation

Rd (rotated right) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↑ | 0 | ↑ |

Assembly-Language Format

ROTR Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

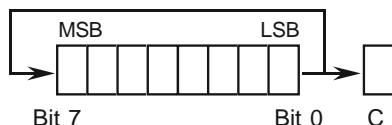
V: Cleared to 0.

C: Receives the previous value in bit 0.

Description

This instruction rotates an 8-bit general register one bit to the right. The least significant bit is rotated to the most significant bit, and also copied to the carry flag.

The operation is shown schematically below.



Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ROTR | Rd | 1 | 3 | 8 | rd | 2 |

2.2.43 ROTXL (rotate with extend carry left)**ROTXL****Operation**

Rd (rotated with carry left) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ |
| — | — | — | — | 0 | ↓ |

Assembly-Language Format

ROTXL Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

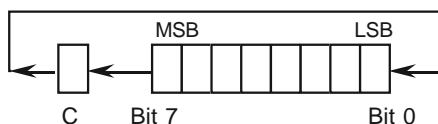
V: Cleared to 0.

C: Receives the previous value in bit 7.

Description

This instruction rotates an 8-bit general register one bit to the left through the carry flag. The carry flag is rotated into the least significant bit of the register. The most significant bit rotates into the carry flag.

The operation is shown schematically below.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ROTXL | Rd | 1 | 2 | 0 | rd | 2 |

2.2.44 ROTXR (rotate with extend carry right)**ROTXR****Operation**

Rd (rotated with carry right) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↑ | 0 | ↑ |

Assembly-Language Format

ROTXR Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

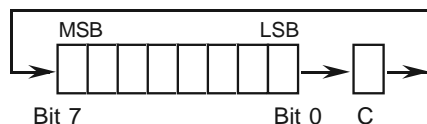
V: Cleared to 0.

C: Receives the previous value in bit 0.

Description

This instruction rotates an 8-bit general register one bit to the right through the carry flag. The least significant bit is rotated into the carry flag. The carry flag rotates into the most significant bit.

The operation is shown schematically below.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | ROTXR | Rd | 1 3 | 0 rd | | | 2 |

2.2.45 RTE (return from exception)**RTE****Operation**

@SP+ → CCR

@SP+ → PC

Condition Code

| I | | H | | N | | Z | | V | | C |
|---|--|---|--|---|--|---|--|---|--|---|
| ↓ | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ |

Assembly-Language Format

RTE

Operand Size

—

I: Restored from stack.

H: Restored from stack.

N: Restored from stack.

Z: Restored from stack.

V: Restored from stack.

C: Restored from stack.

Description

This instruction returns from an exception-handling routine. It pops the 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.

The CCR is one byte in size, but it is popped from the stack as a word (in which the lower 8 bits are ignored). This instruction therefore adds 4 to the value of the stack pointer (R7).

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | RTE | | 5 6 | 7 0 | | | 10 |

2.2.46 RTS (return from subroutine)**RTS****Operation**

Condition Code

@SP+ → PC

Assembly-Language Format

RTS

Operand Size

—

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

I: Previous value remains unchanged.

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 returns from a subroutine. It pops 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.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | RTS | | 5 | 4 | 7 | 0 | 8 |

2.2.47 SHAL (shift arithmetic left)

SHAL

Operation

Rd (shifted arithmetic left) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↑ | ↑ |

Assembly-Language Format

SHAL Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

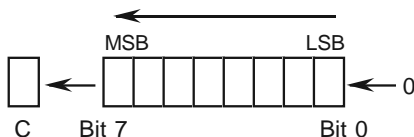
N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.V: Set to 1 when an overflow occurs;
otherwise cleared to 0.

C: Receives the previous value in bit 7.

Description

This instruction shifts an 8-bit general register one bit to the left. The most significant bit shifts into the carry flag, and the least significant bit is cleared to 0.

The operation is shown schematically below.



The SHAL instruction is identical to the SHLL instruction except for its effect on the overflow (V) flag.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SHAL | Rd | 1 0 | 8 rd | | | 2 |

2.2.48 SHAR (shift arithmetic right)

SHAR

Operation

Condition Code

Rd (shifted arithmetic right) → Rd

Assembly-Language Format

SHAR Rd

Operand Size

Byte

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↑ | 0 | ↑ |

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

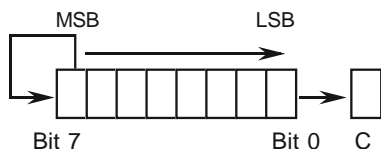
Z: Set to 1 when the result is zero; otherwise cleared to 0.

V: Cleared to 0.

C: Receives the previous value in bit 0.

Description

This instruction shifts an 8-bit general register one bit to the right. The most significant bit remains unchanged. The sign of the result does not change. The least significant bit shifts into the carry flag.



The operation is shown schematically below.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SHAR | Rd | 1 | 1 | 8 | rd | 2 |

2.2.49 SHLL (shift logical left)**SHLL****Operation**

Rd (shifted logical left) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | ↑ | ↑ |
| — | — | — | — | 0 | ↑ |

Assembly-Language Format

SHLL Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

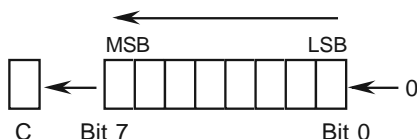
V: Cleared to 0.

C: Receives the previous value in bit 0.

Description

This instruction shifts an 8-bit general register one bit to the left. The least significant bit is cleared to 0. The most significant bit shifts into the carry flag.

The operation is shown schematically below.



The SHLL instruction is identical to the SHAL instruction except for its effect on the overflow (V) flag.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SHLL | Rd | 1 | 0 | 0 | rd | 2 |

2.2.50 SHLR (shift logical right)**SHLR****Operation**

Rd (shifted logical right) → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | ↑ | 0 | ↑ |

Assembly-Language Format

SHLR Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

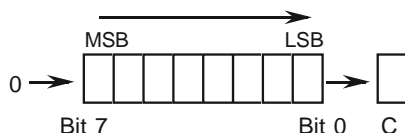
V: Cleared to 0.

C: Receives the previous value in bit 0.

Description

This instruction shifts an 8-bit general register one bit to the right. The most significant bit is cleared to 0. The least significant bit shifts into the carry flag.

The operation is shown schematically below.

**Instruction Formats and Number of Execution States**

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SHLR | Rd | 1 1 | 0 rd | | | 2 |

2.2.51 SLEEP (sleep)**SLEEP****Operation**

Program execution state
→ power-down mode

Condition Code

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

Assembly-Language Format

SLEEP

Operand Size

—

I: Previous value remains unchanged.

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

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 (interrupt or reset). When it receives an exception-handling request, the CPU exits the power-down mode and begins the exception-handling sequence.

If the interrupt mask (I) bit is set to 1, the power-down mode can be released only by a nonmaskable interrupt (NMI) or reset.

For information about the power-down modes, see the applicable hardware manual.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| — | SLEEP | | 0 1 | 8 0 | | | 2 |

2.2.52 STC (store from control register)**STC****Operation**

Condition Code

CCR → Rd

Assembly-Language Format

STC CCR, Rd

Operand Size

Byte

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

I: Previous value remains unchanged.

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 condition code register (CCR) to a specified general register. Bits 6 and 4 are copied as well as the flag bits.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | STC | CCR, Rd | 0 2 | 0 rd | | | 2 |

2.2.53 (1) SUB (subtract binary) (byte)**SUB****Operation**

Rd – Rs → Rd

Condition Code

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| — | — | ↕ | — | ↕ | ↕ |

Assembly-Language Format

SUB.B Rs, Rd

Operand Size

Byte

I: Previous value remains unchanged.

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

This instruction subtracts an 8-bit source register from an 8-bit destination register and places the result in the destination register.

Only register direct addressing is supported. To subtract immediate data it is necessary to use the SUBX.B instruction, first setting the zero flag to 1 and clearing the carry flag to 0.

The following codings can also be used to subtract nonzero immediate data.

(1) ORC #H'05, CCR

(2) ADD #(0 – Imm), Rd

SUBX #(Imm – 1), Rd

XORC #H'01, CCR

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SUB.B | Rs, Rd | 1 8 | rs rd | | | 2 |

2.2.53 (2) SUB (subtract binary) (word)

SUB

Operation

Rd - Rs → Rd

Assembly-Language Format

SUB.W Rs, Rd

Operand Size

Word

Condition Code

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| I | | H | | N | Z | V | C |
| — | — | ↕ | — | ↕ | ↕ | ↕ | ↕ |

- I: Previous value remains unchanged.
- H: Set to 1 when there is a borrow from bit 11; otherwise cleared to 0.
- N: Set to 1 when the result is negative; otherwise cleared to 0.
- Z: Set to 1 when the result is zero; otherwise cleared to 0.
- V: Set to 1 when an overflow occurs; otherwise cleared to 0.
- C: Set to 1 when there is a borrow from bit 15; otherwise cleared to 0.

Description

This instruction subtracts a 16-bit source register from a 16-bit destination register and places the result in the destination register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|----------|------------------|---------------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Register direct | SUB.W | Rs, Rd | 1 9 | 0: rs 0: rd | | | 2 |

2.2.54 SUBS (subtract with sign extension)**SUBS****Operation**

Rd – 1 → Rd

Rd – 2 → Rd

Condition Code

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

Assembly-Language Format

SUBS #1, Rd

SUBS #2, Rd

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Operand Size

Word

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction subtracts the immediate value 1 or 2 from word data in a general register.

Unlike the SUB instruction, it does not affect the condition code flags.

The SUBS instruction does not permit byte operands.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | | | No. of states |
|-----------------|-------|----------|------------------|---|----------|---|----------|----------|---------------|
| | | | 1st byte | | 2nd byte | | 3rd byte | 4th byte | |
| Register direct | SUBS | #1, Rd | 1 | B | 0 | 0 | rd | | 2 |
| Register direct | SUBS | #2, Rd | 1 | B | 8 | 0 | rd | | 2 |

2.2.55 SUBX (subtract with extend carry)

SUBX

| | | | | | | | | | | | | | |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operation | Condition Code | | | | | | | | | | | | |
| Rd – (EAs) – C → Rd | | | | | | | | | | | | | |
| Assembly-Language Format | <table><tr><td>I</td><td>H</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></tr></table> | I | H | N | Z | V | C | — | — | ↑ | — | ↑ | ↑ |
| I | H | N | Z | V | C | | | | | | | | |
| — | — | ↑ | — | ↑ | ↑ | | | | | | | | |
| SUBX <EAs>, Rd | I: Previous value remains unchanged. | | | | | | | | | | | | |
| Operand Size | H: Set to 1 if there is a borrow from bit 3; otherwise cleared to 0. | | | | | | | | | | | | |
| Byte | N: Set to 1 when the result is negative; otherwise cleared to 0. | | | | | | | | | | | | |
| | Z: Previous value remains unchanged when the result is zero; otherwise cleared to 0. | | | | | | | | | | | | |
| | V: Set to 1 when an overflow occurs; otherwise cleared to 0. | | | | | | | | | | | | |

Description

This instruction subtracts the source operand and carry flag from the contents of an 8-bit general register and places the result in the general register.

| Instruction Formats and Number of Execution States | | | | | | | |
|--|-------|-----------|--------------------|----------|----------|----------|---------------|
| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | SUBX | #xx:8, Rd | B rd | IMM | | | 2 |
| Register direct | SUBX | Rs, Rd | 1 E rs rd | | | | 2 |

2.2.56 XOR (exclusive OR logical)**XOR****Operation** $Rd \oplus (EAs) \rightarrow Rd$ **Condition Code**

| I | | H | | N | Z | V | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | ↓ | ↓ | 0 | — |

Assembly-Language Format

XOR <EAs>, Rd

Operand Size

Byte

I: Previous value remains unchanged.

H: Previous value remains unchanged.

N: Set to 1 when the result is negative;
otherwise cleared to 0.Z: Set to 1 when the result is zero;
otherwise cleared to 0.

V: Cleared to 0.

C: Previous value remains unchanged.

Description

This instruction exclusive-ORs the source operand with the contents of an 8-bit general register and places the result in the general register.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|-----------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | XOR | #xx:8, Rd | D | rd | IMM | | 2 |
| Register direct | XOR | Rs, Rd | 1 | 5 | rs | rd | 2 |

2.2.57 XORC (exclusive OR control register)**XORC****Operation**

Condition Code

 $\text{CCR} \oplus \# \text{IMM} \rightarrow \text{CCR}$ **Assembly-Language Format**

XORC #xx:8, CCR

Operand Size

Byte

| I | H | N | Z | V | C |
|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |

I: Exclusive-ORed with bit 7 of the immediate data.

H: Exclusive-ORed with bit 5 of the immediate data.

N: Exclusive-ORed with bit 3 of the immediate data.

Z: Exclusive-ORed with bit 2 of the immediate data.

V: Exclusive-ORed with bit 1 of the immediate data.

C: Exclusive-ORed with bit 0 of the immediate data.

Description

This instruction exclusive-ORs the condition code register (CCR) with immediate data and places the result in the condition code register. Bits 6 and 4 are exclusive-ORed as well as the flag bits.

No interrupt requests are accepted immediately after this instruction. All interrupts, including the nonmaskable interrupt (NMI), are deferred until after the next instruction.

Instruction Formats and Number of Execution States

| Addressing mode | Mnem. | Operands | Instruction code | | | | No. of states |
|-----------------|-------|------------|------------------|----------|----------|----------|---------------|
| | | | 1st byte | 2nd byte | 3rd byte | 4th byte | |
| Immediate | XORC | #xx:8, CCR | 0 | 5 | IMM | | 2 |

2.3 Operation Code Map

Table 2-1 shows the operation code map for instructions of the H8/300L CPU. Only the first byte (bits 15 to 8 of the first word) of the instruction code is indicated here.

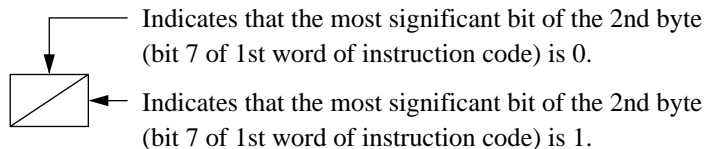


Table 2-1 Operation Code Map

| LO HI | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | | |
|----------|--------------|--------------|---------------|---------------|-----|------|------|------------|------|-----|-----|--------|-------------------------------|-----|------|-----|--|--|
| 0 | NOP | SLEEP | STC | LDC | ORC | XORC | ANDC | LDC | ADD | | INC | ADDS | MOV | | ADDX | DAA | | |
| 1 | SHLL SHAL | SHLR SHAR | ROTXL ROTL | ROTXR ROTR | OR | XOR | AND | NOT NEG | SUB | | DEC | SUBS | CMP | | SUBX | DAS | | |
| 2 | MOV | | | | | | | | | | | | | | | | | |
| 3 | MOV | | | | | | | | | | | | | | | | | |
| 4 | BRA | BRN | BHI | BLS | BCC | BCS | BNE | BEQ | BVC | BVS | BPL | BMI | BGE | BLT | BGT | BLE | | |
| 5 | MULXU | DIVXU | | | RTS | BSR | RTE | | | | JMP | | | | JSR | | | |
| 6 | BSET | BNOT | BCLR | BTST | | | | BST | MOV* | | | | | | | | | |
| 7 | | | | | BOR | BXOR | BAND | BIST | BLD | MOV | | EEPMOV | Bit manipulation instructions | | | | | |
| 8 | ADD | | | | | | | | | | | | | | | | | |
| 9 | ADDX | | | | | | | | | | | | | | | | | |
| A | CMP | | | | | | | | | | | | | | | | | |
| B | SUBX | | | | | | | | | | | | | | | | | |
| C | OR | | | | | | | | | | | | | | | | | |
| D | XOR | | | | | | | | | | | | | | | | | |
| E | AND | | | | | | | | | | | | | | | | | |
| F | MOV | | | | | | | | | | | | | | | | | |

Note: The PUSH and POP instructions are equivalent in machine language to the MOV instruction. See the descriptions of individual instructions in section 2.2, Instructions, for details.

2.4 List of Instructions

Table 2-2 List of Instructions (1)

| Mnemonic | Size | Operation | Addressing Mode and Instruction Length (Bytes) | | | | | | | | Condition Code | | | | | | No. of States * | |
|-----------------------|------|-------------------------------|--|----|-----|-------------|-----------|----------|------------|------|----------------|---|---|---|---|---|-----------------|---|
| | | | #xx:8/16 | Rn | @Rn | @(d:16, Rn) | @-Rn/@Rn+ | @aa:8/16 | @(d:8, PC) | @@aa | Implied | I | H | N | Z | V | | C |
| MOV.B #xx:8, Rd | B | #xx:8 → Rd8 | 2 | | | | | | | | | — | — | ↑ | ↓ | 0 | — | 2 |
| MOV.B Rs, Rd | B | Rs8 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↓ | 0 | — | 2 |
| MOV.B @Rs, Rd | B | @Rs16 → Rd8 | | | 2 | | | | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.B @(d:16, Rs), Rd | B | @(d:16, Rs16) → Rd8 | | | | 4 | | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.B @Rs+, Rd | B | @Rs16 → Rd8 Rs16+1 → Rs16 | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.B @aa:8, Rd | B | @aa:8 → Rd8 | | | | | | 2 | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.B @aa:16, Rd | B | @aa:16 → Rd8 | | | | | | 4 | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.B Rs, @Rd | B | Rs8 → @Rd16 | | | 2 | | | | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.B Rs, @(d:16, Rd) | B | Rs8 → @(d:16, Rd16) | | | | 4 | | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.B Rs, @-Rd | B | Rd16-1 → Rd16 Rs8 → @Rd16 | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.B Rs, @aa:8 | B | Rs8 → @aa:8 | | | | | | 2 | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.B Rs, @aa:16 | B | Rs8 → @aa:16 | | | | | | 4 | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W #xx:16, Rd | W | #xx:16 → Rd | 4 | | | | | | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.W Rs, Rd | W | Rs16 → Rd16 | | 2 | | | | | | | | — | — | ↑ | ↓ | 0 | — | 2 |
| MOV.W @Rs, Rd | W | @Rs16 → Rd16 | | | 2 | | | | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.W @(d:16, Rs), Rd | W | @(d:16, Rs16) → Rd16 | | | | 4 | | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W @Rs+, Rd | W | @Rs16 → Rd16 Rs16+2 → Rs16 | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W @aa:16, Rd | W | @aa:16 → Rd16 | | | | | | 4 | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W Rs, @Rd | W | Rs16 → @Rd16 | | | 2 | | | | | | | — | — | ↑ | ↓ | 0 | — | 4 |
| MOV.W Rs, @(d:16, Rd) | W | Rs16 → @(d:16, Rd16) | | | | 4 | | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W Rs, @-Rd | W | Rd16-2 → Rd16 Rs16 → @Rd16 | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| MOV.W Rs, @aa:16 | W | Rs16 → @aa:16 | | | | | | 4 | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| POP Rd | W | @SP → Rd16 SP+2 → SP | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |
| PUSH Rs | W | SP-2 → SP Rs16 → @SP | | | | | 2 | | | | | — | — | ↑ | ↓ | 0 | — | 6 |

Table 2-2 List of Instructions (2)

| Mnemonic | Size | Operation | Addressing Mode and Instruction Length (Bytes) | | | | | | | | | | Condition Code | | | | | | No. of States * |
|------------------|------|---|--|----|-----|--------------|------------|----------|-------------|-------|---------|----------------|----------------|-----|-----|---|-----|----|-----------------|
| | | | #xx:8/16 | Rn | @Rn | @ (d:16, Rn) | @ -Rn/@Rn+ | @aa:8/16 | @ (d:8, PC) | @ @aa | Implied | Condition Code | | | | | | | |
| | | | | | | | | | | | | I | H | N | Z | V | C | | |
| ADD.B #xx:8, Rd | B | Rd8+#xx:8 → Rd8 | 2 | | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| ADD.B Rs, Rd | B | Rd8+Rs8 → Rd8 | | 2 | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| ADD.W Rs, Rd | W | Rd16+Rs16 → Rd16 | | 2 | | | | | | | | — | (1) | ↑ | ↑ | ↑ | ↑ | 2 | |
| ADDX.B #xx:8, Rd | B | Rd8+#xx:8+C → Rd8 | 2 | | | | | | | | | — | ↑ | ↑ | (2) | ↑ | ↑ | 2 | |
| ADDX.B Rs, Rd | B | Rd8+Rs8+C → Rd8 | | 2 | | | | | | | | — | ↑ | ↑ | (2) | ↑ | ↑ | 2 | |
| ADDS.W #1, Rd | W | Rd16+1 → Rd16 | | 2 | | | | | | | | — | — | — | — | — | — | 2 | |
| ADDS.W #2, Rd | W | Rd16+2 → Rd16 | | 2 | | | | | | | | — | — | — | — | — | — | 2 | |
| INC.B Rd | B | Rd8+1 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↑ | ↑ | — | 2 | |
| DAA.B Rd | B | Rd8 decimal-adjust → Rd8 | | 2 | | | | | | | | — | * | ↑ | ↑ | * | (3) | 2 | |
| SUB.B Rs, Rd | B | Rd8–Rs8 → Rd8 | | 2 | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| SUB.W Rs, Rd | W | Rd16–Rs16 → Rd16 | | 2 | | | | | | | | — | (1) | ↑ | ↑ | ↑ | ↑ | 2 | |
| SUBX.B #xx:8, Rd | B | Rd8–#xx:8–C → Rd8 | 2 | | | | | | | | | — | ↑ | ↑ | (2) | ↑ | ↑ | 2 | |
| SUBX.B Rs, Rd | B | Rd8–Rs8–C → Rd8 | | 2 | | | | | | | | — | ↑ | ↑ | (2) | ↑ | ↑ | 2 | |
| SUBS.W #1, Rd | W | Rd16–1 → Rd16 | | 2 | | | | | | | | — | — | — | — | — | — | 2 | |
| SUBS.W #2, Rd | W | Rd16–2 → Rd16 | | 2 | | | | | | | | — | — | — | — | — | — | 2 | |
| DEC.B Rd | B | Rd8–1 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↑ | ↑ | — | 2 | |
| DAS.B Rd | B | Rd8 decimal-adjust → Rd8 | | 2 | | | | | | | | — | * | ↑ | ↑ | * | — | 2 | |
| NEG.B Rd | B | 0–Rd → Rd | | 2 | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| CMP.B #xx:8, Rd | B | Rd8–#xx:8 | | 2 | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| CMP.B Rs, Rd | B | Rd8–Rs8 | | 2 | | | | | | | | — | ↑ | ↑ | ↑ | ↑ | ↑ | 2 | |
| CMP.W Rs, Rd | W | Rd16–Rs16 | | 2 | | | | | | | | — | (1) | ↑ | ↑ | ↑ | ↑ | 2 | |
| MULXU.B Rs, Rd | B | Rd8×Rs8 → Rd16 | | 2 | | | | | | | | — | — | — | — | — | — | 14 | |
| DIVXU.B Rs, Rd | B | Rd16÷Rs8 → Rd16 (RdH: remainder, RdL: quotient) | | 2 | | | | | | | | — | — | (5) | (6) | — | — | 14 | |
| AND.B #xx:8, Rd | B | Rd8^#xx:8 → Rd8 | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| AND.B Rs, Rd | B | Rd8^Rs8 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| OR.B #xx:8, Rd | B | Rd8∨#xx:8 → Rd8 | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| OR.B Rs, Rd | B | Rd8∨Rs8 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| XOR.B #xx:8, Rd | B | Rd8⊕#xx:8 → Rd8 | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| XOR.B Rs, Rd | B | Rd8⊕Rs8 → Rd8 | | 2 | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |
| NOT.B Rd | B | $\overline{Rd} \rightarrow Rd$ | | 2 | | | | | | | | — | — | ↑ | ↑ | 0 | — | 2 | |

Table 2-2 List of Instructions (3)

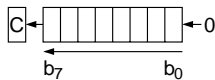
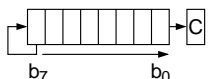
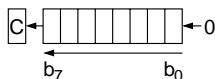
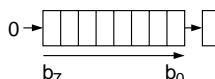
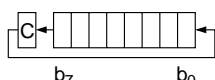
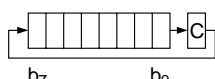
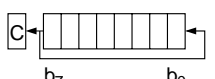
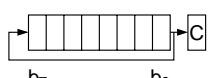
| Mnemonic | Size | Operation | Addressing Mode and Instruction Length (Bytes) | | | | | | | | Condition Code | | | | | | No. of States * | |
|-------------------|------|---|--|----|------|--------------|------------|-----------|-------------|--------|----------------|---|---|---|---|---|-----------------|---|
| | | | #xx:8/16 | Rn | @ Rn | @ (d:16, Rn) | @-Rn/@ Rn+ | @ aa:8/16 | @ (d:8, PC) | @ @ aa | Implied | I | H | N | Z | V | | C |
| SHAL.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | ↑ | ↑ | 2 |
| SHAR.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| SHLL.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| SHLR.B Rd | B |  | 2 | | | | | | | | | — | — | 0 | ↑ | 0 | ↑ | 2 |
| ROTXL.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| ROTXR.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| ROTL.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| ROTR.B Rd | B |  | 2 | | | | | | | | | — | — | ↑ | ↑ | 0 | ↑ | 2 |
| BSET #xx:3, Rd | B | (#xx:3 of Rd8) ← 1 | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BSET #xx:3, @Rd | B | (#xx:3 of @Rd16) ← 1 | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BSET #xx:3, @aa:8 | B | (#xx:3 of @aa:8) ← 1 | | | | | | 4 | | | | — | — | — | — | — | — | 8 |
| BSET Rn, Rd | B | (Rn8 of Rd8) ← 1 | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BSET Rn, @Rd | B | (Rn8 of @Rd16) ← 1 | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BSET Rn, @aa:8 | B | (Rn8 of @aa:8) ← 1 | | | | | | 4 | | | | — | — | — | — | — | — | 8 |

Table 2-2 List of Instructions (4)

| Mnemonic | Size | Operation | Addressing Mode and Instruction Length (Bytes) | | | | | | | | | | Condition Code | | | | | | No. of States * |
|-------------------|------|--|--|----|------|--------------|------------|-----------|-------------|-------|---------|---|----------------|---|---|---|---|---|-----------------|
| | | | #xx:8/16 | Rn | @ Rn | @ (d:16, Rn) | @ -Rn/@Rn+ | @ aa:8/16 | @ (d:8, PC) | @ @aa | Implied | I | H | N | Z | V | C | | |
| | | | | | | | | | | | | | | | | | | | |
| BCLR #xx:3, Rd | B | (#xx:3 of Rd8) ← 0 | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BCLR #xx:3, @Rd | B | (#xx:3 of @Rd16) ← 0 | | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BCLR #xx:3, @aa:8 | B | (#xx:3 of @aa:8) ← 0 | | | | | | 4 | | | | | — | — | — | — | — | — | 8 |
| BCLR Rn, Rd | B | (Rn8 of Rd8) ← 0 | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BCLR Rn, @Rd | B | (Rn8 of @Rd16) ← 0 | | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BCLR Rn, @aa:8 | B | (Rn8 of @aa:8) ← 0 | | | | | | 4 | | | | | — | — | — | — | — | — | 8 |
| BNOT #xx:3, Rd | B | (#xx:3 of Rd8) ← (#xx:3 of Rd8) | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BNOT #xx:3, @Rd | B | (#xx:3 of @Rd16) ← (#xx:3 of @Rd16) | | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BNOT #xx:3, @aa:8 | B | (#xx:3 of @aa:8) ← (#xx:3 of @aa:8) | | | | | | 4 | | | | | — | — | — | — | — | — | 8 |
| BNOT Rn, Rd | B | (Rn8 of Rd8) ← (Rn8 of Rd8) | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BNOT Rn, @Rd | B | (Rn8 of @Rd16) ← (Rn8 of @Rd16) | | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BNOT Rn, @aa:8 | B | (Rn8 of @aa:8) ← (Rn8 of @aa:8) | | | | | | 4 | | | | | — | — | — | — | — | — | 8 |
| BTST #xx:3, Rd | B | (#xx:3 of Rd8) → Z | | 2 | | | | | | | | | — | — | — | ↑ | — | — | 2 |
| BTST #xx:3, @Rd | B | (#xx:3 of @Rd16) → Z | | | 4 | | | | | | | | — | — | — | ↑ | — | — | 6 |
| BTST #xx:3, @aa:8 | B | (#xx:3 of @aa:8) → Z | | | | | | 4 | | | | | — | — | — | ↑ | — | — | 6 |
| BTST Rn, Rd | B | (Rn8 of Rd8) → Z | | 2 | | | | | | | | | — | — | — | ↑ | — | — | 2 |
| BTST Rn, @Rd | B | (Rn8 of @Rd16) → Z | | | 4 | | | | | | | | — | — | — | ↑ | — | — | 6 |
| BTST Rn, @aa:8 | B | (Rn8 of @aa:8) → Z | | | | | | 4 | | | | | — | — | — | ↑ | — | — | 6 |
| BLD #xx:3, Rd | B | (#xx:3 of Rd8) → C | | 2 | | | | | | | | | — | — | — | — | — | ↑ | 2 |
| BLD #xx:3, @Rd | B | (#xx:3 of @Rd16) → C | | | 4 | | | | | | | | — | — | — | — | — | ↑ | 6 |
| BLD #xx:3, @aa:8 | B | (#xx:3 of @aa:8) → C | | | | | | 4 | | | | | — | — | — | — | — | ↑ | 6 |
| BILD #xx:3, Rd | B | (#xx:3 of Rd8) → C | | 2 | | | | | | | | | — | — | — | — | — | ↑ | 2 |
| BILD #xx:3, @Rd | B | (#xx:3 of @Rd16) → C | | | 4 | | | | | | | | — | — | — | — | — | ↑ | 6 |
| BILD #xx:3, @aa:8 | B | (#xx:3 of @aa:8) → C | | | | | | 4 | | | | | — | — | — | — | — | ↑ | 6 |
| BST #xx:3, Rd | B | C → (#xx:3 of Rd8) | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| BST #xx:3, @Rd | B | C → (#xx:3 of @Rd16) | | | 4 | | | | | | | | — | — | — | — | — | — | 8 |
| BST #xx:3, @aa:8 | B | C → (#xx:3 of @aa:8) | | | | | | 4 | | | | | — | — | — | — | — | — | 8 |

Table 2-2 List of Instructions (5)

| Mnemonic | Size | Operation | | Addressing Mode and Instruction Length (Bytes) | | | | | | | | | | Condition Code | | | | | | No. of States* |
|--------------------|------|---|----------------|--|----|------|--------------|------------|-----------|-------------|-------|---------|---|----------------|---|---|---|----|---|----------------|
| | | | | #xx:8/16 | Rn | @ Rn | @ (d:16, Rn) | @-Rn/@ Rn+ | @ aa:8/16 | @ (d:8, PC) | @ @aa | Implied | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | |
| | | Branching Condition | I | H | N | Z | V | C | | | | | | | | | | | | |
| BIST #xx:3, Rd | B | $\overline{C} \rightarrow (\#xx:3 \text{ of Rd8})$ | | 2 | | | | | | | | | — | — | — | — | — | — | 2 | |
| BIST #xx:3, @Rd | B | $\overline{C} \rightarrow (\#xx:3 \text{ of @Rd16})$ | | | 4 | | | | | | | | — | — | — | — | — | — | 8 | |
| BIST #xx:3, @aa:8 | B | $\overline{C} \rightarrow (\#xx:3 \text{ of @aa:8})$ | | | | | | 4 | | | | | — | — | — | — | — | — | 8 | |
| BAND #xx:3, Rd | B | $C \wedge (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BAND #xx:3, @Rd | B | $C \wedge (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BAND #xx:3, @aa:8 | B | $C \wedge (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIAND #xx:3, Rd | B | $C \wedge (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BIAND #xx:3, @Rd | B | $C \wedge (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIAND #xx:3, @aa:8 | B | $C \wedge (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BOR #xx:3, Rd | B | $C \vee (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BOR #xx:3, @Rd | B | $C \vee (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BOR #xx:3, @aa:8 | B | $C \vee (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIOR #xx:3, Rd | B | $C \vee (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BIOR #xx:3, @Rd | B | $C \vee (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIOR #xx:3, @aa:8 | B | $C \vee (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BXOR #xx:3, Rd | B | $C \oplus (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BXOR #xx:3, @Rd | B | $C \oplus (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BXOR #xx:3, @aa:8 | B | $C \oplus (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIXOR #xx:3, Rd | B | $C \oplus (\#xx:3 \text{ of Rd8}) \rightarrow C$ | | 2 | | | | | | | | | — | — | — | — | — | ↑↓ | 2 | |
| BIXOR #xx:3, @Rd | B | $C \oplus (\#xx:3 \text{ of @Rd16}) \rightarrow C$ | | | 4 | | | | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BIXOR #xx:3, @aa:8 | B | $C \oplus (\#xx:3 \text{ of @aa:8}) \rightarrow C$ | | | | | | 4 | | | | | — | — | — | — | — | ↑↓ | 6 | |
| BRA d:8 (BT d:8) | — | $PC \leftarrow PC+d:82$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 | |
| BRN d:8 (BF d:8) | — | $PC \leftarrow PC+2$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 | |
| BHI d:8 | — | if condition is true then $PC \leftarrow$ $PC+d:8$ else next; | $C \vee Z = 0$ | | | | | | 2 | | | | — | — | — | — | — | — | 4 | |
| BLS d:8 | — | | $C \vee Z = 1$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BCC d:8 (BHS d:8) | — | | $C = 0$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BCS d:8 (BLO d:8) | — | | $C = 1$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BNE d:8 | — | | $Z = 0$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BEQ d:8 | — | | $Z = 1$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BVC d:8 | — | | $V = 0$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |
| BVS d:8 | — | | $V = 1$ | | | | | | | 2 | | | | — | — | — | — | — | — | 4 |

Table 2-2 List of Instructions (6)

| Mnemonic | Size | Operation | Branching Condition | Addressing Mode and Instruction Length (Bytes) | | | | | | | | Condition Code | | | | | | No. of States* | |
|-----------------|------|--|---------------------|--|----|-----|-------------|-----------|----------|------------|------|----------------|---|---|---|---|---|----------------|----|
| | | | | #xx:8/16 | Rn | @Rn | @(d:16, Rn) | @-Rn/@Rn+ | @aa:8/16 | @(d:8, PC) | @@aa | Implied | I | H | N | Z | V | | C |
| | | | | | | | | | | | | | | | | | | | |
| BPL d:8 | — | if condition is true then PC ← PC+d:8 else next; | N = 0 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| BMI d:8 | — | | N = 1 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| BGE d:8 | — | | N⊕V = 0 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| BLT d:8 | — | | N⊕V = 1 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| BGT d:8 | — | | Z∨(N⊕V) = 0 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| BLE d:8 | — | | Z∨(N⊕V) = 1 | | | | | | | 2 | | | — | — | — | — | — | — | 4 |
| JMP @Rn | — | PC ← Rn16 | | | 2 | | | | | | | | — | — | — | — | — | — | 4 |
| JMP @aa:16 | — | PC ← aa:16 | | | | | | | 4 | | | | — | — | — | — | — | — | 6 |
| JMP @@aa:8 | — | PC ← @aa:8 | | | | | | | | | 2 | | — | — | — | — | — | — | 8 |
| BSR d:8 | — | SP-2 → SP PC → @SP PC ← PC+d:8 | | | | | | | | 2 | | | — | — | — | — | — | — | 6 |
| JSR @Rn | — | SP-2 → SP PC → @SP PC ← Rn16 | | | 2 | | | | | | | | — | — | — | — | — | — | 6 |
| JSR @aa:16 | — | SP-2 → SP PC → @SP PC ← aa:16 | | | | | | | 4 | | | | — | — | — | — | — | — | 8 |
| JSR @@aa:8 | — | SP-2 → SP PC → @SP PC ← @aa:8 | | | | | | | | | 2 | | — | — | — | — | — | — | 8 |
| RTS | — | PC ← @SP SP+2 → SP | | | | | | | | | | 2 | — | — | — | — | — | — | 8 |
| RTE | — | CCR ← @SP SP+2 → SP PC ← @SP SP+2 → SP | | | | | | | | | | 2 | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | 10 |
| SLEEP | — | Transit to sleep mode. | | | | | | | | | | 2 | — | — | — | — | — | — | 2 |
| LDC #xx:8, CCR | B | #xx:8 → CCR | 2 | | | | | | | | | | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | 2 |
| LDC Rs, CCR | B | Rs8 → CCR | | 2 | | | | | | | | | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | 2 |
| STC CCR, Rd | B | CCR → Rd8 | | 2 | | | | | | | | | — | — | — | — | — | — | 2 |
| ANDC #xx:8, CCR | B | CCR∧#xx:8 → CCR | 2 | | | | | | | | | | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | 2 |
| ORC #xx:8, CCR | B | CCR∨#xx:8 → CCR | 2 | | | | | | | | | | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | 2 |

Table 2-2 List of Instructions (7)

| Mnemonic | Size | Operation | Addressing Mode and Instruction Length (Bytes) | | | | | | | | | | Condition Code | | | | | | No. of States * |
|-----------------|------|--|--|----|------|--------------|-------------|-----------|-------------|--------|---------|---|----------------|---|---|---|---|---|-----------------|
| | | | #xx:8/16 | Rn | @ Rn | @ (d:16, Rn) | @ -Rn/@ Rn+ | @ aa:8/16 | @ (d:8, PC) | @ @ aa | Implied | | | | | | | | |
| | | | I | H | N | Z | V | C | | | | | | | | | | | |
| XORC #xx:8, CCR | B | CCR⊕#xx:8 → CCR | 2 | | | | | | | | | | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | 2 |
| NOP | — | PC ← PC+2 | | | | | | | | | | 2 | — | — | — | — | — | — | 2 |
| EEPMOV | — | if R4L ≠ 0 Repeat @R5 → @R6 R5+1 → R5 R6+1 → R6 R4L-1 → R4L Until R4L = 0 else next; | | | | | | | | | | 4 | — | — | — | — | — | — | (4) |

Notes: * The number of execution states indicated here assumes that the operation code and operand data are in on-chip memory. For other cases, refer to section 2.5, Number of Execution States.

- (1) Set to 1 when there is a carry or borrow at bit 11; otherwise cleared to 0.
- (2) When the result is 0, the previous value remains unchanged; otherwise cleared to 0.
- (3) Set to 1 when there is a carry in the adjusted result; otherwise the previous value remains unchanged.
- (4) The number of execution states is $4n + 9$, with n being the value set in R4L.
- (5) Set to 1 when the divisor is negative; otherwise cleared to 0.
- (6) Set to 1 when the divisor is 0; otherwise cleared to 0.

2.5 Number of Execution States

The tables here can be used to calculate the number of states required for instruction execution.

Table 2-3 indicates the number of states required for each cycle (instruction fetch, branch address read, stack operation, byte data access, word data access, internal operation).

Table 2-4 indicates the number of cycles of each type occurring in each instruction. The total 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: When instruction is fetched from on-chip ROM, and an on-chip RAM is accessed.

1. BSET #0, @FF00

From table 2-4:

$$I = L = 2, J = K = M = N = 0$$

From table 2-3:

$$S I = 2, S L = 2$$

$$\text{Number of states required for execution} = 2 \times 2 + 2 \times 2 = 8$$

When instruction is fetched from on-chip ROM, branch address is read from on-chip ROM, and on-chip RAM is used for stack area.

2. JSR @@ 30

From table 2-4:

$$I = 2, J = K = 1, L = M = N = 0$$

From table 2-3:

$$S I = S J = S K = 2$$

$$\text{Number of states required for execution} = 2 \times 2 + 1 \times 2 + 1 \times 2 = 8$$

Table 2-3 Number of States Taken by Each Cycle in Instruction Execution

| Execution Status (instruction cycle) | | Access Location | |
|---|----------------|-----------------|---------------------------|
| | | On-Chip Memory | On-Chip Peripheral Module |
| Instruction fetch | S ^I | 2 | |
| Branch address read | S ^J | | |
| Stack operation | S ^K | | |
| Byte data access | S ^L | | 2 or 3* |
| Word data access | S ^M | | |
| Internal operation | S ^N | 1 | |

* Depends on which on-chip module is accessed. See the applicable hardware manual for details.

Table 2-4 Number of Cycles in Each Instruction

| Instruction | Mnemonic | Instruction Fetch | Branch Addr. 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 Rs, Rd | 1 | | | | | |
| ADDS | ADDS.W #1/2, Rd | 1 | | | | | |
| ADDX | ADDX.B #xx:8, Rd | 1 | | | | | |
| | ADDX.B Rs, Rd | 1 | | | | | |
| AND | AND.B #xx:8, Rd | 1 | | | | | |
| | AND.B Rs, Rd | 1 | | | | | |
| ANDC | ANDC #xx:8, CCR | 1 | | | | | |
| BAND | BAND #xx:3, Rd | 1 | | | | | |
| | BAND #xx:3, @Rd | 2 | | | 1 | | |
| | BAND #xx:3, @aa:8 | 2 | | | 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 | | | | | |

| Instruction | Mnemonic | Instruction | Branch | Stack | Byte | Word | Internal |
|-------------|--------------------|-------------|------------|-------|-------------|-------------|----------|
| | | Fetch | Addr. Read | | Data Access | Data Access | |
| | | I | J | K | L | M | N |
| BCLR | BCLR #xx:3, Rd | 1 | | | | | |
| | BCLR #xx:3, @Rd | 2 | | | 2 | | |
| | BCLR #xx:3, @aa:8 | 2 | | | 2 | | |
| | BCLR Rn, Rd | 1 | | | | | |
| | BCLR Rn, @Rd | 2 | | | 2 | | |
| | BCLR Rn, @aa:8 | 2 | | | 2 | | |
| BIAND | BIAND #xx:3, Rd | 1 | | | | | |
| | BIAND #xx:3, @Rd | 2 | | | 1 | | |
| | BIAND #xx:3, @aa:8 | 2 | | | 1 | | |
| BILD | BILD #xx:3, Rd | 1 | | | | | |
| | BILD #xx:3, @Rd | 2 | | | 1 | | |
| | BILD #xx:3, @aa:8 | 2 | | | 1 | | |
| BIOR | BIOR #xx:3, Rd | 1 | | | | | |
| | BIOR #xx:3, @Rd | 2 | | | 1 | | |
| | BIOR #xx:3, @aa:8 | 2 | | | 1 | | |
| BIST | BIST #xx:3, Rd | 1 | | | | | |
| | BIST #xx:3, @Rd | 2 | | | 2 | | |
| | BIST #xx:3, @aa:8 | 2 | | | 2 | | |
| BIXOR | BIXOR #xx:3, Rd | 1 | | | | | |
| | BIXOR #xx:3, @Rd | 2 | | | 1 | | |
| | BIXOR #xx:3, @aa:8 | 2 | | | 1 | | |
| BLD | BLD #xx:3, Rd | 1 | | | | | |
| | BLD #xx:3, @Rd | 2 | | | 1 | | |
| | BLD #xx:3, @aa:8 | 2 | | | 1 | | |
| BNOT | BNOT #xx:3, Rd | 1 | | | | | |
| | BNOT #xx:3, @Rd | 2 | | | 2 | | |
| | BNOT #xx:3, @aa:8 | 2 | | | 2 | | |
| | BNOT Rn, Rd | 1 | | | | | |
| | BNOT Rn, @Rd | 2 | | | 2 | | |
| | BNOT Rn, @aa:8 | 2 | | | 2 | | |

| Instruction | Mnemonic | Instruction Fetch | Branch Addr. Read | Stack Operation | Byte Data Access | Word Data Access | Internal Operation |
|-------------|-------------------|----------------------|-------------------------|--------------------|------------------------|------------------------|-----------------------|
| | | I | J | K | L | M | N |
| BOR | BOR #xx:3, Rd | 1 | | | | | |
| | BOR #xx:3, @Rd | 2 | | | 1 | | |
| | BOR #xx:3, @aa:8 | 2 | | | 1 | | |
| BSET | BSET #xx:3, Rd | 1 | | | | | |
| | BSET #xx:3, @Rd | 2 | | | 2 | | |
| | BSET #xx:3, @aa:8 | 2 | | | 2 | | |
| | BSET Rn, Rd | 1 | | | | | |
| | BSET Rn, @Rd | 2 | | | 2 | | |
| | BSET Rn, @aa:8 | 2 | | | 2 | | |
| BSR | BSR d:8 | 2 | | 1 | | | |
| BST | BST #xx:3, Rd | 1 | | | | | |
| | BST #xx:3, @Rd | 2 | | | 2 | | |
| | BST #xx:3, @aa:8 | 2 | | | 2 | | |
| BTST | BTST #xx:3, Rd | 1 | | | | | |
| | BTST #xx:3, @Rd | 2 | | | 1 | | |
| | BTST #xx:3, @aa:8 | 2 | | | 1 | | |
| | BTST Rn, Rd | 1 | | | | | |
| | BTST Rn, @Rd | 2 | | | 1 | | |
| | BTST Rn, @aa:8 | 2 | | | 1 | | |
| BXOR | BXOR #xx:3, Rd | 1 | | | | | |
| | BXOR #xx:3, @Rd | 2 | | | 1 | | |
| | BXOR #xx:3, @aa:8 | 2 | | | 1 | | |
| CMP | CMP. B #xx:8, Rd | 1 | | | | | |
| | CMP. B Rs, Rd | 1 | | | | | |
| | CMP.W Rs, Rd | 1 | | | | | |
| DAA | DAA.B Rd | 1 | | | | | |
| DAS | DAS.B Rd | 1 | | | | | |
| DEC | DEC.B Rd | 1 | | | | | |
| DIVXU | DIVXU.B Rs, Rd | 1 | | | | | 12 |
| EEPMOV | EEPMOV | 2 | | | 2n + 2* | | 1 |
| INC | INC.B Rd | 1 | | | | | |

| Instruction | Mnemonic | Instruction | Branch | Stack | Byte | Word | Internal |
|-------------|-----------------------|-------------|---------------|-------|----------------|----------------|----------|
| | | Fetch | Addr. Read | | Data Access | Data Access | |
| | | I | J | K | L | M | N |
| JMP | JMP @Rn | 2 | | | | | |
| | JMP @aa:16 | 2 | | | | | 2 |
| | JMP @@aa:8 | 2 | 1 | | | | 2 |
| JSR | JSR @Rn | 2 | | 1 | | | |
| | JSR @aa:16 | 2 | | 1 | | | 2 |
| | JSR @@aa:8 | 2 | 1 | 1 | | | |
| LDC | LDC #xx:8, CCR | 1 | | | | | |
| | LDC Rs, CCR | 1 | | | | | |
| MOV | MOV.B #xx:8, Rd | 1 | | | | | |
| | MOV.B Rs, Rd | 1 | | | | | |
| | MOV.B @Rs, Rd | 1 | | | 1 | | |
| | MOV.B @(d:16, Rs), Rd | 2 | | | 1 | | |
| | MOV.B @Rs+, Rd | 1 | | | 1 | | 2 |
| | MOV.B @aa:8, Rd | 1 | | | 1 | | |
| | MOV.B @aa:16, Rd | 2 | | | 1 | | |
| | MOV.B Rs, @Rd | 1 | | | 1 | | |
| | MOV.B Rs, @(d:16, Rd) | 2 | | | 1 | | |
| | MOV.B Rs, @-Rd | 1 | | | 1 | | 2 |
| | MOV.B Rs, @aa:8 | 1 | | | 1 | | |
| | MOV.B Rs, @aa:16 | 2 | | | 1 | | |
| | MOV.W #xx:16, Rd | 2 | | | | | |
| | MOV.W Rs, Rd | 1 | | | | | |
| | MOV.W @Rs, Rd | 1 | | | | 1 | |
| | MOV.W @(d:16, Rs), Rd | 2 | | | | 1 | |
| | MOV.W @Rs+, Rd | 1 | | | | 1 | 2 |
| | MOV.W @aa:16, Rd | 2 | | | | 1 | |
| | MOV.W Rs, @Rd | 1 | | | | 1 | |
| | MOV.W Rs, @(d:16, Rd) | 2 | | | | 1 | |
| | MOV.W Rs, @-Rd | 1 | | | | 1 | 2 |
| | MOV.W Rs, @aa:16 | 2 | | | | 1 | |
| MULXU | MULXU.B Rs, Rd | 1 | | | | | 12 |

| Instruction | Mnemonic | Instruction | Branch | Stack | Byte | Word | Internal |
|-------------|------------------|-------------|------------|-------|-------------|-------------|----------|
| | | Fetch | Addr. Read | | Data Access | Data Access | |
| | | I | J | K | L | M | N |
| NEG | NEG.B Rd | 1 | | | | | |
| NOP | NOP | 1 | | | | | |
| NOT | NOT.B Rd | 1 | | | | | |
| OR | OR.B #xx:8, Rd | 1 | | | | | |
| | OR.B Rs, Rd | 1 | | | | | |
| ORC | ORC #xx:8, CCR | 1 | | | | | |
| POP | POP Rd | 1 | | 1 | | | 2 |
| PUSH | PUSH Rs | 1 | | 1 | | | 2 |
| ROTL | ROTL.B Rd | 1 | | | | | |
| ROTR | ROTR.B Rd | 1 | | | | | |
| ROTXL | ROTXL.B Rd | 1 | | | | | |
| ROTXR | ROTXR.B Rd | 1 | | | | | |
| RTE | RTE | 2 | | 2 | | | 2 |
| RTS | RTS | 2 | | 1 | | | 2 |
| SHLL | SHLL.B Rd | 1 | | | | | |
| SHAL | SHAL.B Rd | 1 | | | | | |
| SHAR | SHAR.B Rd | 1 | | | | | |
| SHLR | SHLR.B Rd | 1 | | | | | |
| SLEEP | SLEEP | 1 | | | | | |
| STC | STC CCR, Rd | 1 | | | | | |
| SUB | SUB.B Rs, Rd | 1 | | | | | |
| | SUB.W Rs, Rd | 1 | | | | | |
| SUBS | SUBS.W #1/2, Rd | 1 | | | | | |
| SUBX | SUBX.B #xx:8, Rd | 1 | | | | | |
| | SUBX.B Rs, Rd | 1 | | | | | |
| XOR | XOR.B #xx:8, Rd | 1 | | | | | |
| | XOR.B Rs, Rd | 1 | | | | | |
| XORC | XORC #xx:8, CCR | 1 | | | | | |

* n: Initial value in R4L. The source and destination operands are accessed n + 1 times each.

Section 3 CPU Operation States

There are three CPU operation states, namely, program execution state, power-down state, and exception-handling state. In power-down state there are sleep mode, standby mode, and watch mode. These operation states are shown in figure 3-1. Figure 3-2 shows the state transitions.

For further details please refer to the applicable hardware manual.

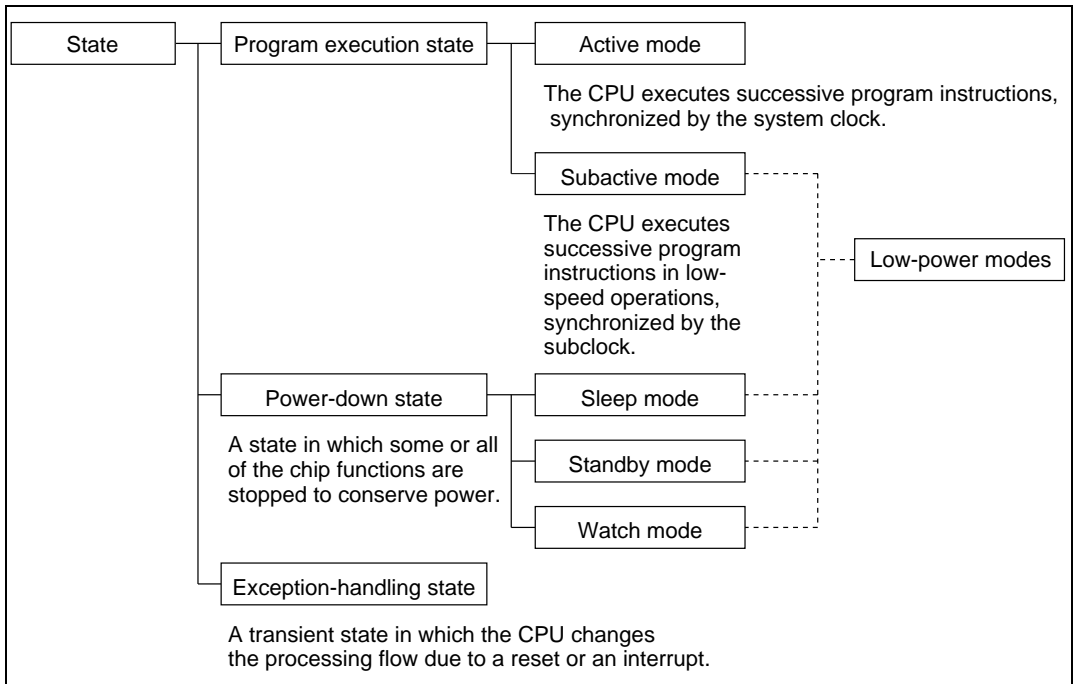
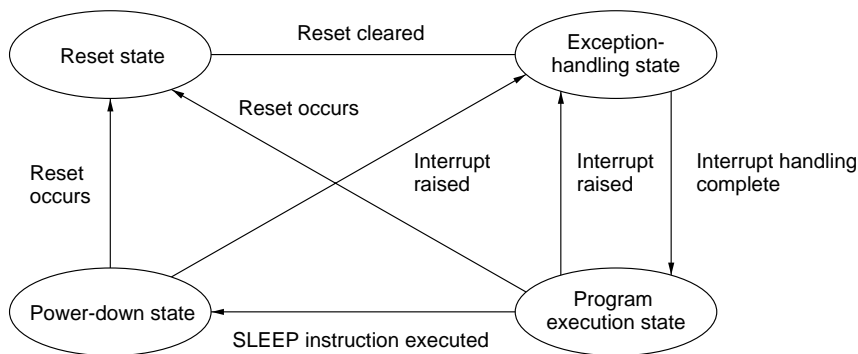


Figure 3-1 CPU Operation States



Note: On the transitions between modes, see the applicable hardware manual.

Figure 3-2 State Transitions

3.1 Program Execution State

In program execution state the CPU executes program instructions in sequence.

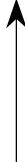
3.2 Exception Handling States

Exception-handling states are transient states occurring when exception handling is raised by a reset or interrupt, and the CPU changes its normal processing flow, branching to a start address acquired from a vector table. In exception handling caused by an interrupt, PC and CCR values are saved to the stack, with reference made to a stack pointer (R7).

3.2.1 Types and Priorities of Exception Handling

Exception handling includes processing of reset exceptions and of interrupts. Table 3-1 summarizes the factors causing each kind of exception, and their priorities. Reset exception handling has the highest priority.

Table 3-1 Types of Exception Handling and Priorities

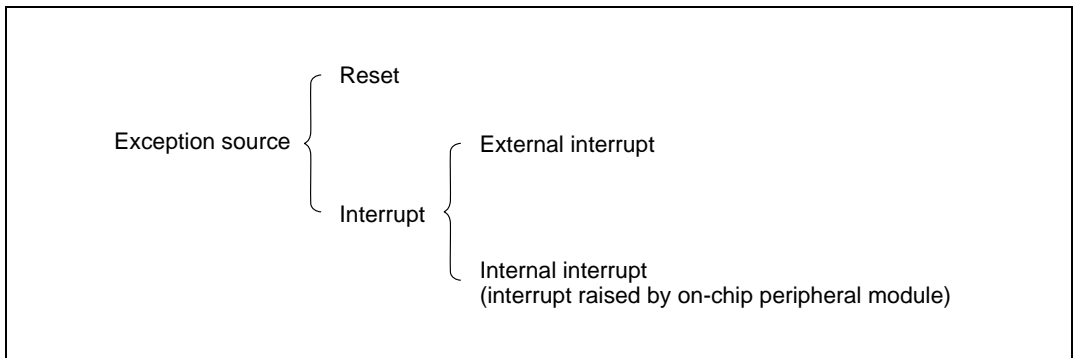
| Priority | Exception source | Detection timing | Timing for start of exception handling |
|---|------------------|---------------------|--|
| High  Low | Reset | Clock-synchronous | Reset exception handling starts as soon as RES pin changes from low to high. |
| | Interrupt | End of instruction* | When an interrupt request is made, execution* interrupt exception handling starts after execution of the present instruction is completed. |

Note: * Interrupt detection is not made upon completion of ANDC, ORC, XORC, and LDC instruction execution, nor upon completion of reset exception handling.

3.2.2 Exception Sources and Vector Table

The factors causing exception handling can be classified as in figure 3-3

For details of exception handling, the vector numbers of each source, and the vector addresses, see the applicable hardware manual.

**Figure 3-3 Classification of Exception Sources**

3.2.3 Outline of Exception Handling Operation

A reset has the highest priority of all exception handling. After the $\overline{\text{RES}}$ pin goes to low level putting the CPU in reset state, the $\overline{\text{RES}}$ pin is then put at high level, and reset exception handling is started at the point when the reset conditions are met. For details on reset conditions refer to the applicable hardware manual. When reset exception handling is started, the CPU gets a start address from the exception handling vector table, and starts executing the exception handling routine from that address. During execution of this routine and immediately after, all interrupts including NMI are masked.

When interrupt exception handling is started, the CPU refers to the stack pointer (R7) and pushes the PC and CCR contents to the stack. The CCR I bit is then set to 1, a start address is acquired from the exception handling vector table, and the interrupt exception handling routine is executed from this address. The stack state in this case is as shown in figure 3-4.

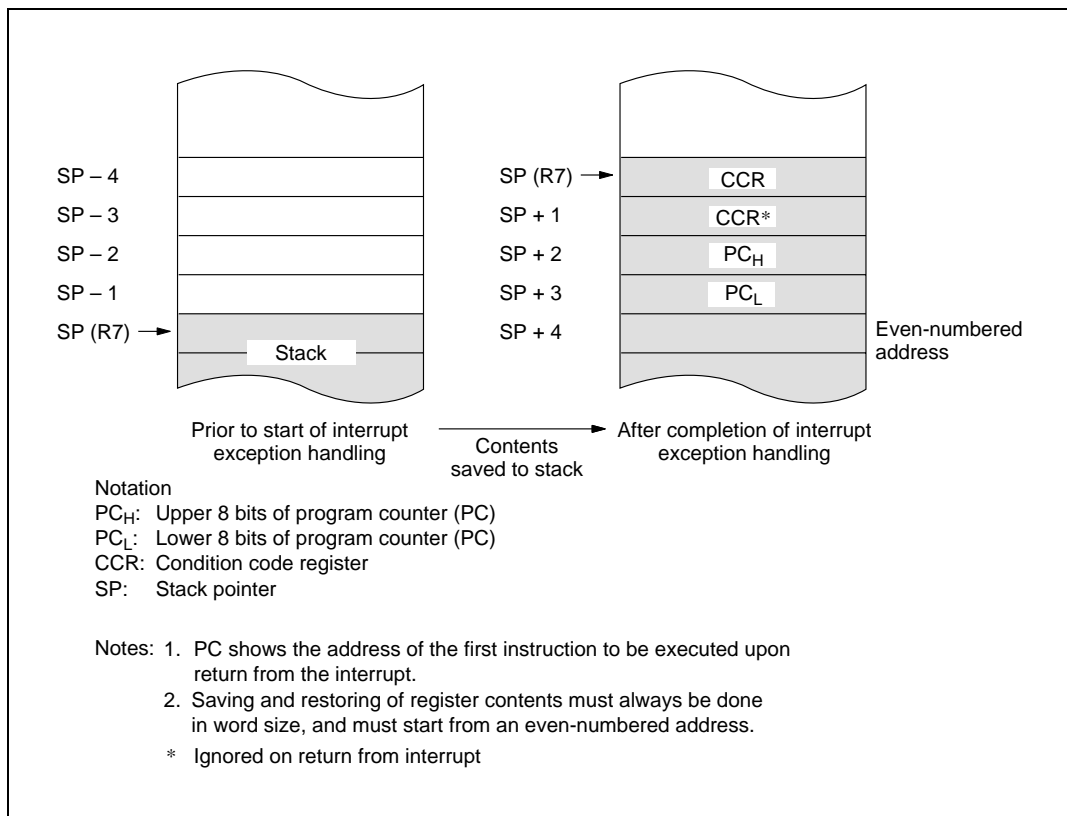


Figure 3-4 Stack State after Completion of Interrupt Exception Handling

3.3 Reset State

When the RES pin goes to low level, all processing stops and the system goes to reset state.

The I bit of the condition code register (CCR) is set, masking all interrupts.

After the RES pin is changed externally from low to high level, reset exception handling starts at the point when the reset conditions are met. For details on reset conditions refer to the applicable hardware manual.

3.4 Power-Down State

In power-down state the CPU operation is stopped, reducing power consumption. For details see the applicable hardware manual.

Section 4 Basic Operation Timing

CPU operation is synchronized by a clock (ϕ). The period from the rising edge of ϕ to the next rising edge is called one state. A memory cycle or bus cycle consists of two or three states.

For details on access to on-chip memory and to on-chip peripheral modules see the applicable hardware manual.

4.1 On-chip Memory (RAM, ROM)

Two-state access is employed for high-speed access to on-chip memory. The data bus width is 16 bits, allowing access in byte or word size. Figure 4-1 shows the on-chip memory access cycle.

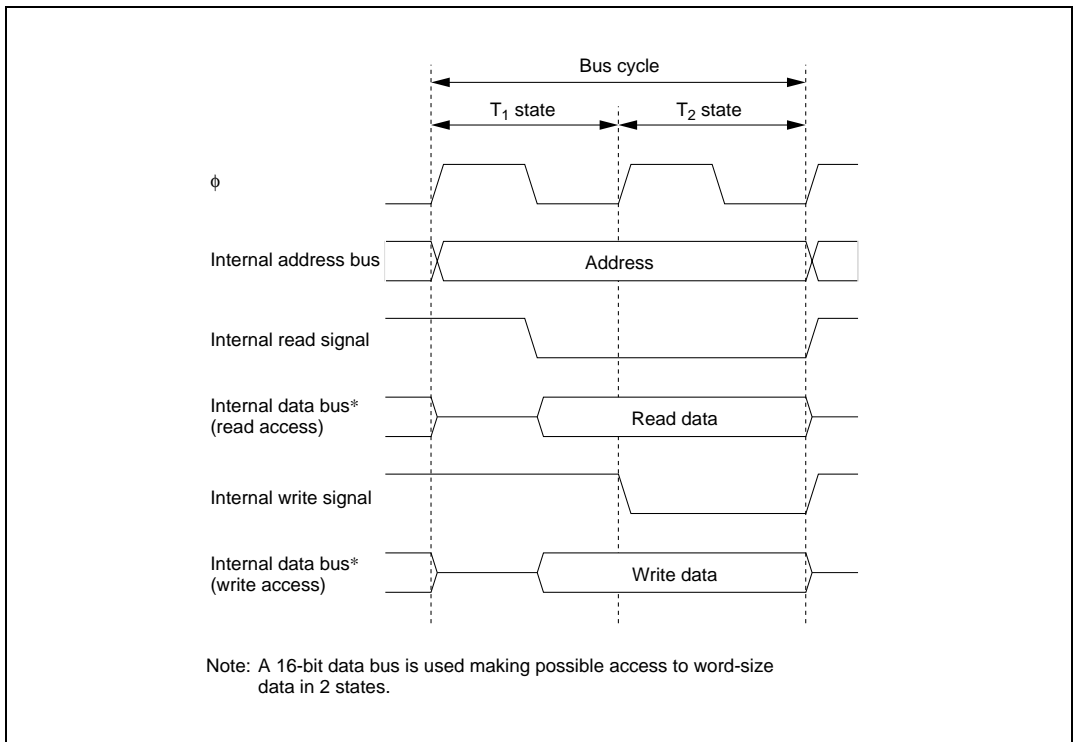


Figure 4-1 On-Chip Memory Access Cycle

4.2 On-chip Peripheral Modules and External Devices

On-chip peripheral modules are accessed in two or three states. The data bus width is 8 bits, so access is made in byte size only. Access to word data or instruction codes is not possible.

Figure 4-2 shows the on-chip peripheral module access cycle.

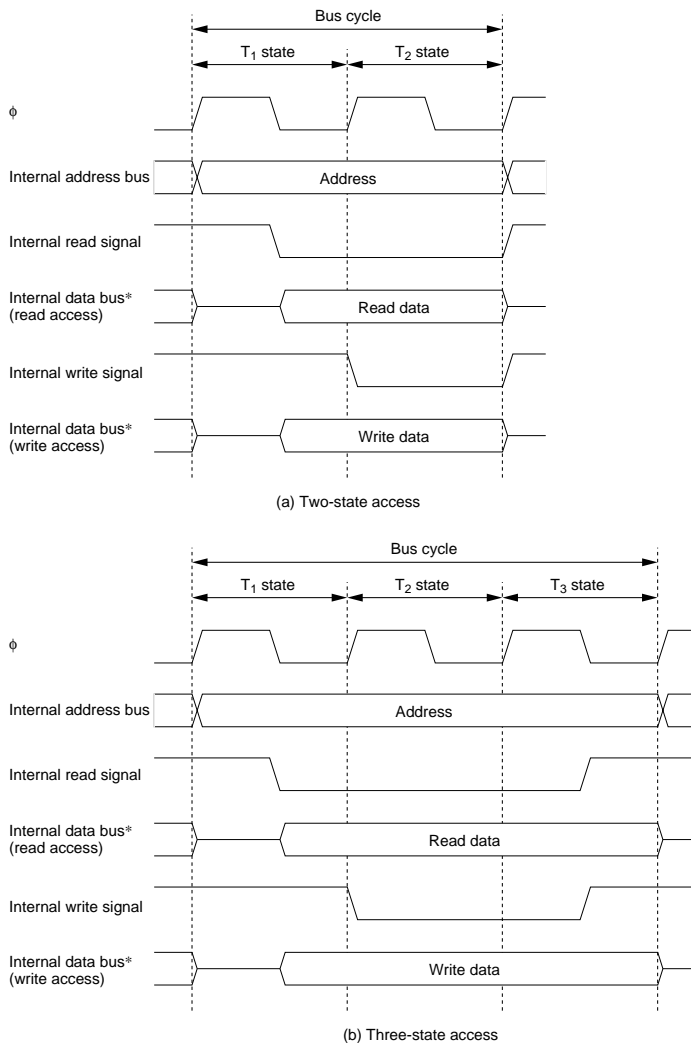


Figure 4-2 On-Chip Peripheral Module Access Cycle

Renesas 16-Bit Single-Chip Microcomputer
Software Manual
H8/300L Series

Publication Date: 1st Edition, December 1991

Rev.2.00, December 27, 2004

Published by: Sales Strategic Planning Div.

Renesas Technology Corp.

Edited by: Technical Documentation & Information Department

Renesas Koudaira Semiconductor Co., Ltd.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan



RENESAS SALES OFFICES

<http://www.renesas.com>

Refer to "<http://www.renesas.com/en/network>" for the latest and detailed information.

Renesas Technology America, Inc.

450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology Hong Kong Ltd.

7th Floor, North Tower, World Finance Centre, Harbour City, 1 Canton Road, Tsimshatsui, Kowloon, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2730-6071

Renesas Technology Taiwan Co., Ltd.

10th Floor, No.99, Fushing North Road, Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology (Shanghai) Co., Ltd.

Unit2607 Ruijing Building, No.205 Maoming Road (S), Shanghai 200020, China
Tel: <86> (21) 6472-1001, Fax: <86> (21) 6415-2952

Renesas Technology Singapore Pte. Ltd.

1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

H8/300L Series Software Manual



Renesas Technology Corp.

2-6-2, Ote-machi, Chiyoda-ku, Tokyo, 100-0004, Japan