To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

RENESAS
Renesas Technology Corp.

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation 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 Corporation 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 Corporation or a third party.
2. Renesas Technology Corporation 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 Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation 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 Corporation 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 Corporation by various means, including the Renesas Technology Corporation 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 Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation 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 Corporation or an authorized Renesas Technology Corporation 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 Corporation 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 Corporation for further details on these materials or the products contained therein.

User's Manual

RENESAS

H8/300

Programming Manual

Renesas Electronics
www.renesas.com

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

Preface

The H8/300 CPU forms the common core of all chips in the H8/300 Series. Featuring a Hitachi-original, high-speed, RISC-like architecture, it has eight 16-bit (or sixteen 8-bit) general registers and a concise, optimized instruction set.

This manual gives detailed descriptions of the H8/300 instructions. The descriptions apply to all chips in the H8/300 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.

Contents

Section 1. CPU

This document is a reference manual for programming the H8/300, a high-speed central processing unit with a Hitachi-original RISC-like architecture that is employed as a CPU core in a series of low-cost single-chip microcomputers intended for applications ranging from smart cards to office and factory automation.

The H8/300 features a concise instruction set in which most frequently-used instructions are two bytes long and execute in just two states (0.2µs with a 10MHz system clock). Its general registers can be accessed as 16-bit word registers or 8-bit byte registers. The instruction set includes both 8-bit and 16-bit instructions.

Section 1 of this manual summarizes the CPU architecture and instruction set. Section 2 gives detailed descriptions of the instructions. Appendices give an operation code map, a complete list of the instruction set, and tables for calculating instruction execution time. Programmers should also refer to the *User's Manual* of the chip being programmed for information on bus cycles, interrupt service, I/O ports, power-down modes, and on-chip facilities such as memory and timers, and for a memory map.

1.1 General CPU Architecture

1.1.1 Features

Table 1-1 summarizes the CPU architecture. Figures 1-1 and 1-2 show how data are stored in registers and memory.

Table 1-1. CPU Architecture

Item	Description	
Address space	64K bytes, H'0000 to H'FFFF	
Data types	Bit, 4-bit (packed BCD), byte, word (2 bytes)	
General registers	Sixteen 8-bit general registers (R0H, R0L, ..., R7H, R7L), also accessible as eight 16-bit general registers (R0 to R7)	
Control registers	Program counter (PC) Condition code register (CCR)	
Addressing modes	Rn	Register direct
	@Rn	Register indirect
	@(d:16, Rn)	Register indirect with 16-bit displacement
	@Rn+	Register indirect with post-increment
	@–Rn	Register indirect with pre-decrement
	@aa:8, @aa:16	Absolute address (8 or 16 bits)
	#xx:8, #xx:16	Immediate (8-, or 16-bit data)
	@(d:8, PC)	PC-relative (8-bit displacement)
	@@aa:8	Memory indirect
Instruction length	2 or 4 bytes	

Notes:
1. Word data stored in memory must be stored at an even address.
2. Instructions must be stored at even addresses.
3. General register R7 is used as the stack pointer (SP).

1.1.2 Data Structure

The H8/300 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, ..., 7) in a byte operand.
- All operational instructions except ADDS and SUBS can operate on byte 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.
- The MOV.W, ADD.W, SUB.W, CMP.W, ADDS, SUBS, MULXU (8 bits × 8 bits), and DIVXU (16 bits ÷ 8 bits) instructions operate on word data.

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	7 … 0: [7 6 5 4 3 2 1 0] [Don't-care]
1-Bit data	RnL	7 … 0: [Don't-care] [7 6 5 4 3 2 1 0]
Byte data	RnH	7 … 0: [MSB … LSB] [Don't-care]
Byte data	RnL	7 … 0: [Don't-care] [MSB … LSB]
Word data	Rn	15 … 0: [MSB … LSB]
4-Bit BCD data	RnH	7 4 3 0: [Upper digit] [Lower digit] [Don't-care]
4-Bit BCD data	RnL	7 4 3 0: [Don't-care] [Upper digit] [Lower digit]

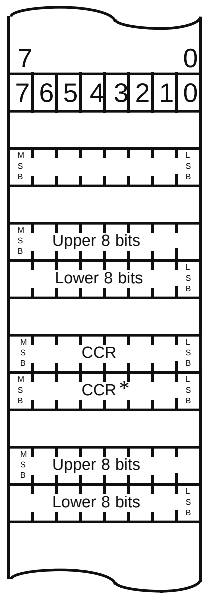
RnH: Upper 8 bits of General Register
RnL: Lower 8 bits of General Register
MSB: Most Significant Bit
LSB: Least Significant Bit

Figure 1-1. Register Data Structure

Memory Data Structure: Figure 1-2 indicates the data structure in memory.

Word data stored in memory must always begin at an even address. In word access the least significant bit of the address is regarded as "0." If an odd address is specified, no address error occurs but the access is performed at the preceding even address. This rule affects MOV.W instructions and branching instructions, and implies that only even addresses should be stored in the vector table.

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 return.

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 H8/300 CPU supports a 64K-byte address space. The memory map differs depending on the particular chip in the H8/300 Series and its operating mode. See the *Hardware Manual* of the chip for details.

1.2 Registers

Figure 1-3 shows the register structure of the H8/300 CPU. There are sixteen 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).

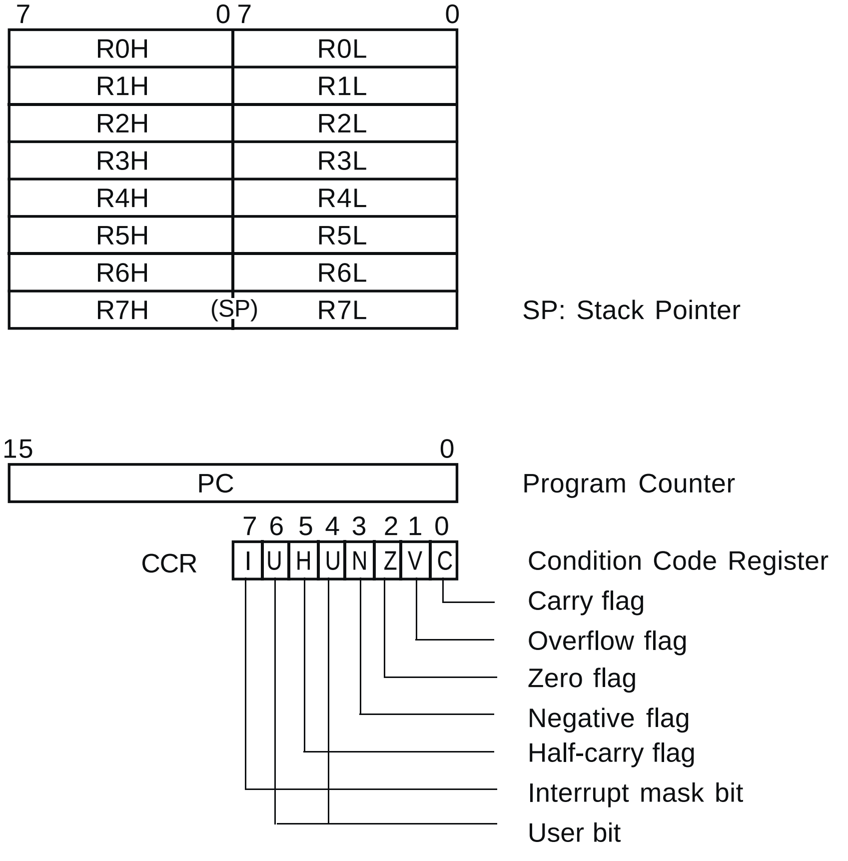


Figure 1-3. CPU Registers

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.

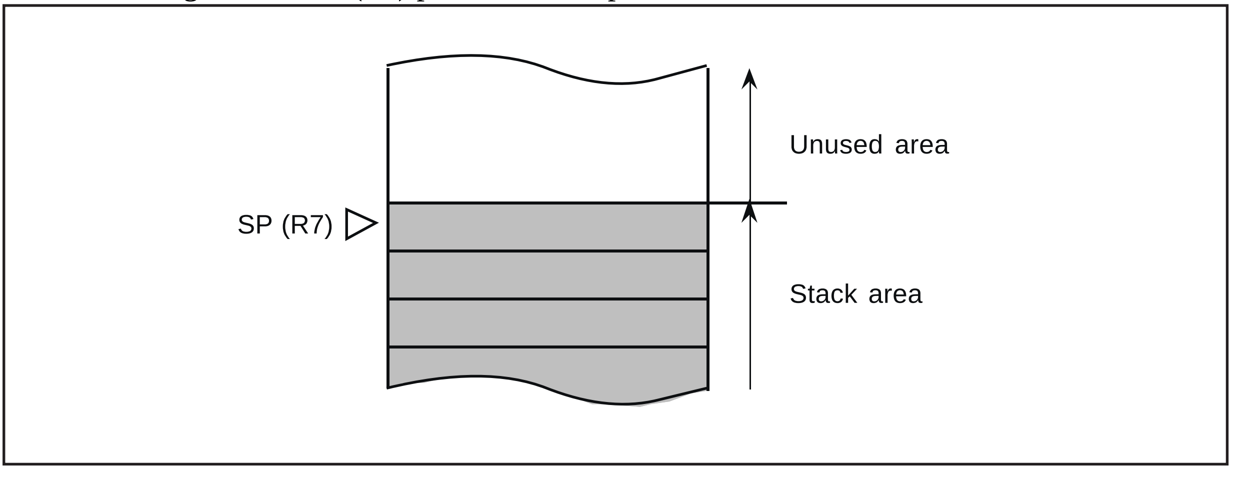


Figure 1-4. Stack Pointer

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	1	*	*	*	*	*	*	*
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

* Undetermined

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 by a reset and 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.

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 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

System control instructions can load and store the CCR, and perform logic operations to set, clear, or toggle selected bits.

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 the CCR is set to “1.” The other CCR bits and the general registers are not initialized.

In particular, the stack pointer (R7) is not initialized. 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 H8/300 has a concise set of 57 RISC-like instructions.
- Arithmetic and logic are performed as register-to-register operations, or with immediate data.
- All instructions are 2 or 4 bytes long.
- Fast multiply/divide instructions; extensive bit manipulation instructions.
- Eight addressing modes.

1.3.1 Types of Instructions

Table 1-2 classifies the H8/300 instructions by type. Tables 1-3 to 1-10 briefly describe their functions. Section 2, Instruction Set, gives detailed descriptions.

Table 1-2. Instruction Classification

Function	Instructions	Types
Data transfer	MOV, MOVFPE, MOVTPE, POP*, PUSH*	3
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, BIAND, 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	57

* 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 1-3 to 1-10 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
#xx:3	3-Bit immediate data
#xx:8	8-Bit immediate data
#xx:16	16-Bit 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 1-3. Data Transfer Instructions

Instruction	Size*	Function
MOV	B/W	(EAs) → Rd, Rs → (EAd) Moves data between two general registers or between a general register and memory, or moves immediate data to a general register. 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. The @–R7 and @R7+ modes require word operands. Do not specify byte size for these two modes.
MOVFPE	B	(EAs) → Rd Transfers data from memory to a general register in synchronization with the E clock.
MOVTPE	B	Rs → (EAd) Transfers data from a general register to memory in synchronization with the E clock.
POP	W	@SP+ → Rn Pops a 16-bit general register from the stack. Equivalent to MOV.W @SP+, Rn.
PUSH	W	Rn → @–SP Pushes a 16-bit general register onto the stack. Equivalent to MOV.W Rn, @-SP.

* Size: Operand size
B: Byte
W: Word

Table 1-4. Arithmetic Instructions

Instruction	Size*	Function
ADD SUB	B/W	Rd ± Rs → Rd, Rd + #Imm → Rd 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 SUBX	B	Rd ± Rs ± C → Rd, Rd ± #Imm ± C → Rd 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 DEC	B	Rd ± 1 → Rd Increments or decrements a general register.
ADDS SUBS	W	Rd ± 1 → Rd, Rd ± 2 → Rd Adds or subtracts immediate data to or from data in a general register. The immediate data must be 1 or 2.
DAA DAS	B	Rd decimal adjust → Rd Decimal-adjusts (adjusts to packed BCD) an addition or subtraction result in a general register by referring to the CCR.
MULXU	B	Rd × Rs → Rd Performs 8-bit × 8-bit unsigned multiplication on data in two general registers, providing a 16-bit result.
DIVXU	B	Rd ÷ Rs → Rd Performs 16-bit ÷ 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 → Rd Obtains the two's complement (arithmetic complement) of data in a general register.

* Size: Operand size
B: Byte
W: Word

Table 1-5. 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 1-6. Shift Instructions

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

* Size: Operand size
B: Byte

Table 1-7. Bit-Manipulation Instructions

Instruction	Size*	Function
BSET	B	1 → (<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 → (<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	¬(<bit-No.> of <EAd>) → (<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	¬ (<bit-No.> of <EAd>) → 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 ∧ (<bit-No.> of <EAd>) → C ANDs the C flag with a specified bit in a general register or memory.
BIAND	B	C ∧ [¬ (<bit-No.> of <EAd>)] → 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 ∨ (<bit-No.> of <EAd>) → C ORs the C flag with a specified bit in a general register or memory.
BIOR	B	C ∨ [¬ (<bit-No.> of <EAd>)] → 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 1-7. Bit-Manipulation Instructions (Cont.)

Instruction	Size*	Function
BXOR	B	C ⊕ (<bit-No.> of <EAd>) → C Exclusive-ORs the C flag with a specified bit in a general register or memory.
BIXOR	B	C ⊕ [¬ (<bit-No.> of <EAd>)] → C 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	(<bit-No.> of <EAd>) → C Copies a specified bit in a general register or memory to the C flag.
BILD	B	¬ (<bit-No.> of <EAd>) → 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 → (<bit-No.> of <EAd>) Copies the C flag to a specified bit in a general register or memory.
BIST	B	¬ C → (<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 1-8. Branching Instructions

Instruction	Size	Function
`Bcc`	—	Branches if condition cc is true.

Mnemonic	cc Field	Description	Condition
`BRA (BT)`	0 0 0 0	Always (True)	Always
`BRN (BF)`	0 0 0 1	Never (False)	Never
`BHI`	0 0 1 0	High	$C \vee Z = 0$
`BLS`	0 0 1 1	Low or Same	$C \vee Z = 1$
`BCC (BHS)`	0 1 0 0	Carry Clear (High or Same)	$C = 0$
`BCS (BLO)`	0 1 0 1	Carry Set (Low)	$C = 1$
`BNE`	0 1 1 0	Not Equal	$Z = 0$
`BEQ`	0 1 1 1	Equal	$Z = 1$
`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	$N \oplus V = 0$
`BLT`	1 1 0 1	Less Than	$N \oplus V = 1$
`BGT`	1 1 1 0	Greater Than	$Z \vee (N \oplus V) = 0$
`BLE`	1 1 1 1	Less or Equal	$Z \vee (N \oplus V) = 1$

Instruction	Size	Function
`JMP`	—	Branches unconditionally to a specified address.
`BSR`	—	Branches to a subroutine at a specified address.
`JSR`	—	Branches to a subroutine at a specified displacement from the current address.
`RTS`	—	Returns from a subroutine.

Table 1-9. System Control Instructions

Instruction	Size*	Function
RTE	—	Returns from an exception-handling routine.
SLEEP	—	Causes a transition to the power-down state.
LDC	B	Rs $\rightarrow$ CCR, #Imm $\rightarrow$ CCR Moves immediate data or general register contents to the condition code register.
STC	B	CCR $\rightarrow$ Rd 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 1-10. Block Data Transfer Instruction

Instruction	Size	Function
EEPMOV	—	if R4L $\neq$ 0 then repeat @R5+ $\rightarrow$ @R6+ R4L – 1 $\rightarrow$ R4L until R4L = 0 else next; Moves a data block according to parameters set in general registers R4L, R5, and R6. R4L: size of block (bytes) R5: starting source address R6: starting destination address Execution of the next instruction starts as soon as the block transfer is completed.

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 the port 4 data direction register (P4DDR) under the following conditions.

$P4_7$: Input pin, Low, MOS pull-up transistor on

$P4_6$: Input pin, High, MOS pull-up transistor off

$P4_5$ – $P4_0$: Output pins, Low

The intended purpose of this BCLR instruction is to switch $P4_0$ from output to input.

Before Execution of BCLR Instruction

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
Input/output	Input	Input	Output	Output	Output	Output	Output	Output
Pin state	Low	High	Low	Low	Low	Low	Low	Low
DDR	0	0	1	1	1	1	1	1
DR	1	0	0	0	0	0	0	0
Pull-up	On	Off	Off	Off	Off	Off	Off	Off

Execution of BCLR Instruction

```
BCLR  #0  @P4DDR     ;clear bit 0 in data direction register
```

After Execution of BCLR Instruction

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
Input/output	Output	Output	Output	Output	Output	Output	Output	Input
Pin state	Low	High	Low	Low	Low	Low	Low	High
DDR	1	1	1	1	1	1	1	0
DR	1	0	0	0	0	0	0	0
Pull-up	Off	Off	Off	Off	Off	Off	Off	Off

Explanation: To execute the BCLR instruction, the CPU begins by reading P4DDR. Since P4DDR 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 P4DDR to complete the BCLR instruction.

As a result, $P4_0DDR$ is cleared to "0," making $P4_0$ an input pin. In addition, $P4_7DDR$ and $P4_6DDR$ are set to "1," making $P4_7$ and $P4_6$ output pins.

Example 2: BSET is executed to set bit 0 in the port 4 data register (P4DR) under the following conditions.

$P4_7$: Input pin, Low, MOS pull-up transistor on

$P4_6$: Input pin, High, MOS pull-up transistor off

$P4_5$ – $P4_0$: Output pins, Low

The intended purpose of this BSET instruction is to switch the output level at $P4_0$ from Low to High.

Before Execution of BSET Instruction

	$P4_7$	**$P4_6$**	**$P4_5$**	**$P4_4$**	**$P4_3$**	**$P4_2$**	**$P4_1$**	**$P4_0$**
Input/output	Input	Input	Output	Output	Output	Output	Output	Output
Pin state	Low	High	Low	Low	Low	Low	Low	Low
DDR	0	0	1	1	1	1	1	1
DR	1	0	0	0	0	0	0	0
Pull-up	On	Off	Off	Off	Off	Off	Off	Off

Execution of BSET Instruction

```
BSET  #0  @PORT4      ;set bit 0 in port-4 data register
```

After Execution of BSET Instruction

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
Input/output	Input	Input	Output	Output	Output	Output	Output	Output
Pin state	Low	High	Low	Low	Low	Low	Low	High
DDR	0	0	1	1	1	1	1	1
DR	0	1	0	0	0	0	0	1
Pull-up	Off	On	Off	Off	Off	Off	Off	Off

Explanation: To execute the BSET instruction, the CPU begins by reading port 4. Since $P4_7$ and $P4_6$ are input pins, the CPU reads the level of these pins directly, not the value in the data register. It reads $P4_7$ as Low ("0") and $P4_6$ as High ("1").

Since $P4_5$ to $P4_0$ are output pins, for these pins the CPU reads the value in the data register ("0"). The CPU therefore reads the value of port 4 as H'40, although the actual value in P4DR 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 P4DR to complete the BSET instruction.

As a result, bit $P4_0$ is set to "1," switching pin $P4_0$ to High output. In addition, bits $P4_7$ and $P4_6$ are both modified, changing the on/off settings of the MOS pull-up transistors of pins $P4_7$ and $P4_6$.

Programming Solution: The switching of the pull-ups for $P4_7$ and $P4_6$ in example 2 can be avoided by storing the same data in both the port-4 data register and in a work area in RAM. Bit manipulations are performed on the data in the work area, after which the result is moved into the port-4 data register. In the following example RAM0 is a symbol for the user-selected address of the work area.

Before Execution of BSET Instruction

```
MOV.B  #80   R0L     ;write data (H'80) for data register
MOV.B  R0L   @RAM0   ;write to DR work area (RAM0)
MOV.B  R0L   @PORT4  ;write to DR
```

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
Input/output	Input	Input	Output	Output	Output	Output	Output	Output
Pin state	Low	High	Low	Low	Low	Low	Low	Low
DDR	0	0	1	1	1	1	1	1
DR	1	0	0	0	0	0	0	0
Pull-up	On	Off	Off	Off	Off	Off	Off	Off
RAM0	1	0	0	0	0	0	0	0

Execution of BSET Instruction

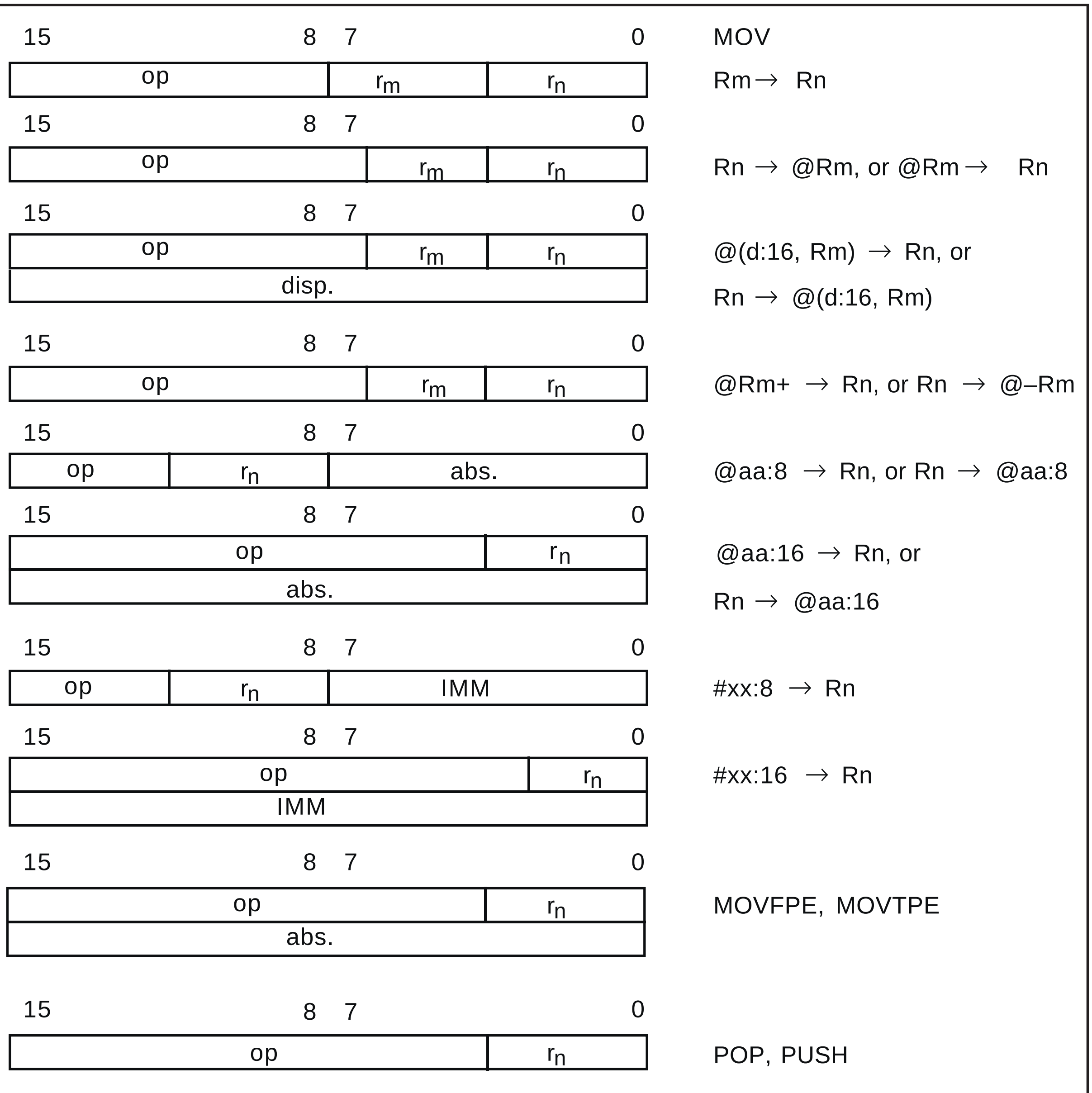
```
BSET  #0  @RAM0      ;set bit 0 in DR work area (RAM0)
```

After Execution of BSET Instruction

```
MOV.B  @RAM0  R0L    ;get value in work area (RAM0)
MOV.B  R0L    @PORT4 ;write value to DR
```

	$P4_7$	$P4_6$	$P4_5$	$P4_4$	$P4_3$	$P4_2$	$P4_1$	$P4_0$
Input/output	Input	Input	Output	Output	Output	Output	Output	Output
Pin state	Low	High	Low	Low	Low	Low	Low	High
DDR	0	0	1	1	1	1	1	1
DR	1	0	0	0	0	0	0	1
Pull-up	On	Off	Off	Off	Off	Off	Off	Off
RAM0	1	0	0	0	0	0	0	1

1.3.3 Machine-Language Coding



Notation

op:	Operation field
r_m, r_n:	Register field
disp:	Displacement
abs.:	Absolute address
IMM:	Immediate data

Figure 1-5. Machine-Language Coding of Data Transfer Instructions

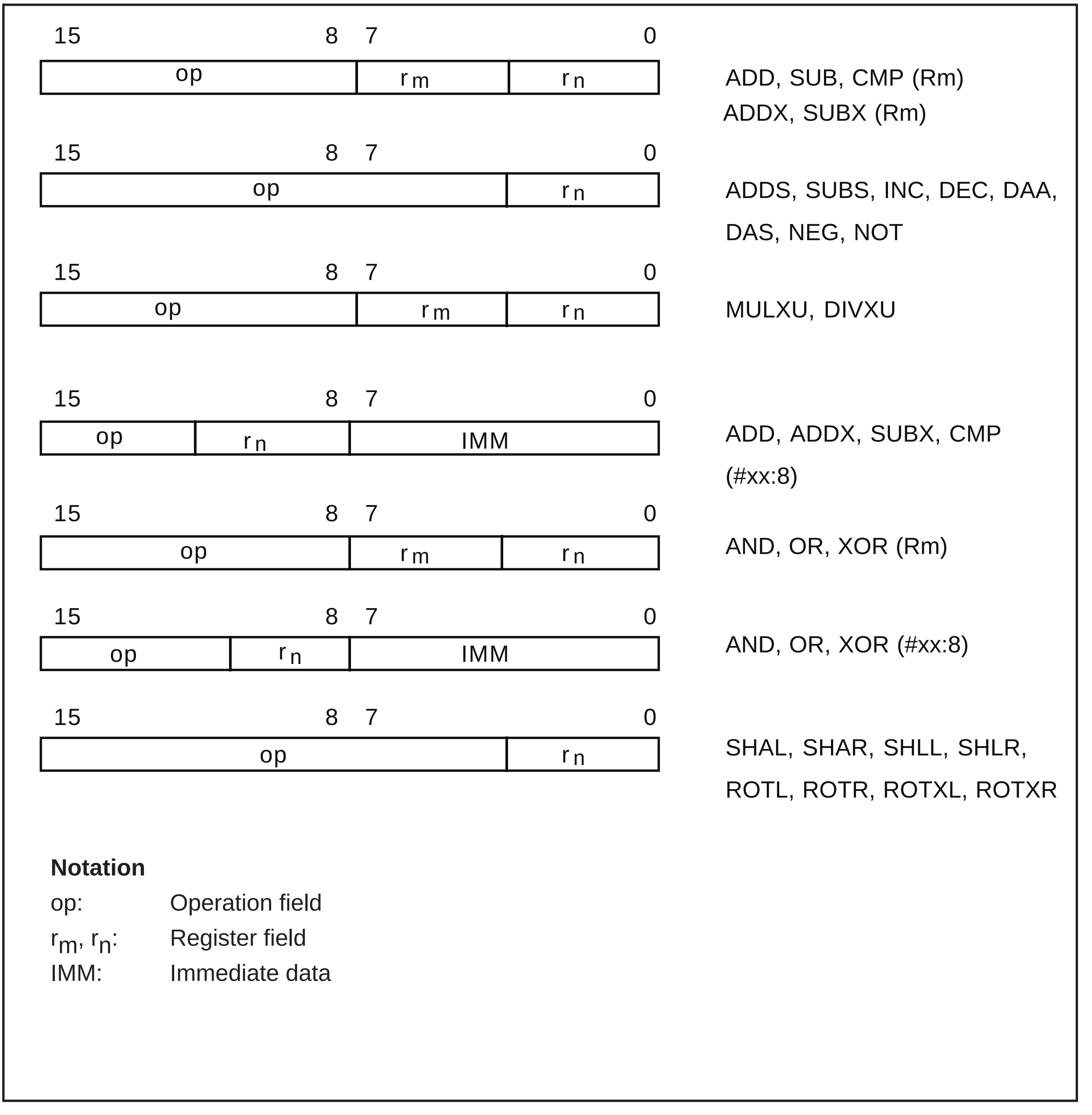
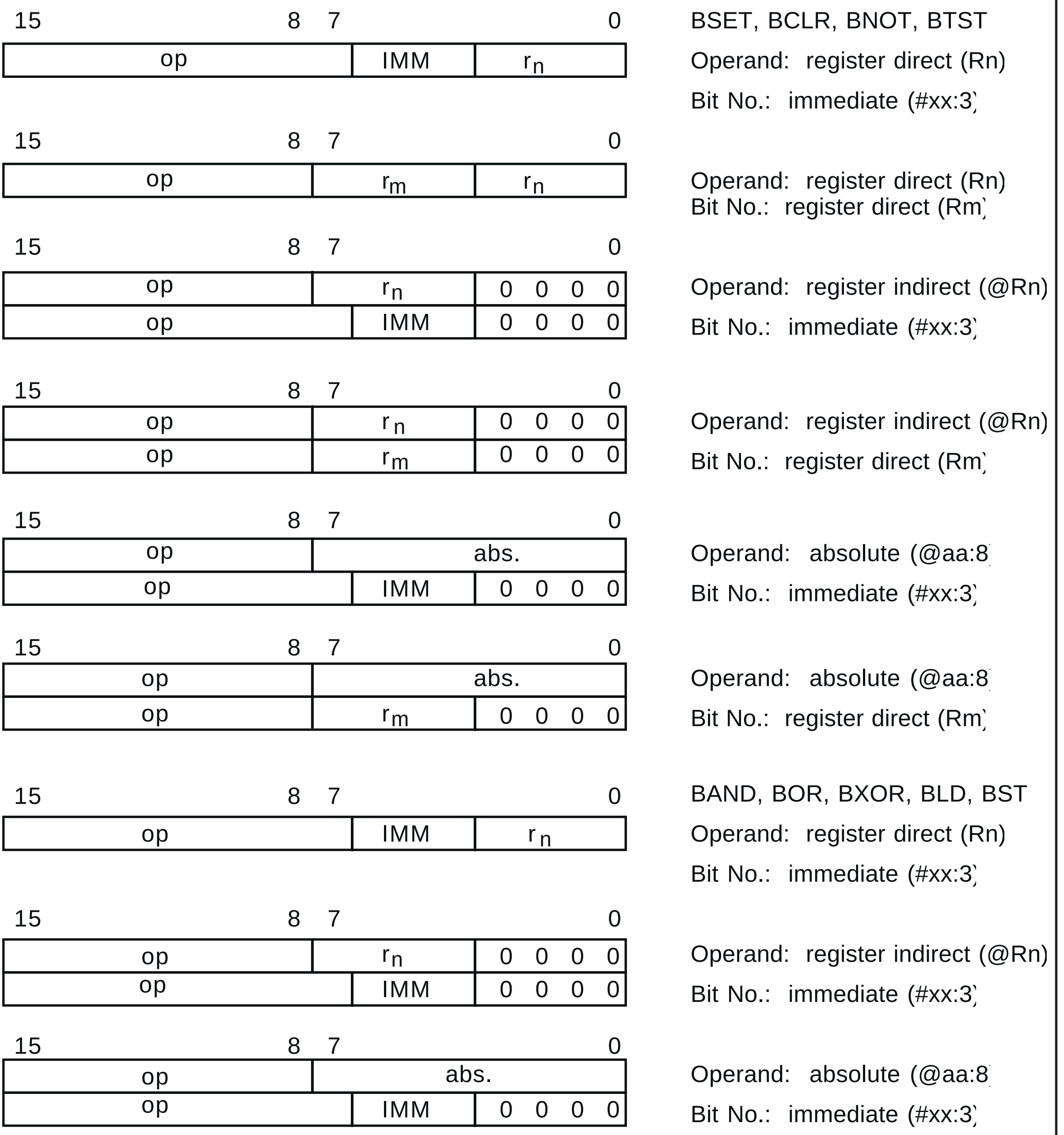


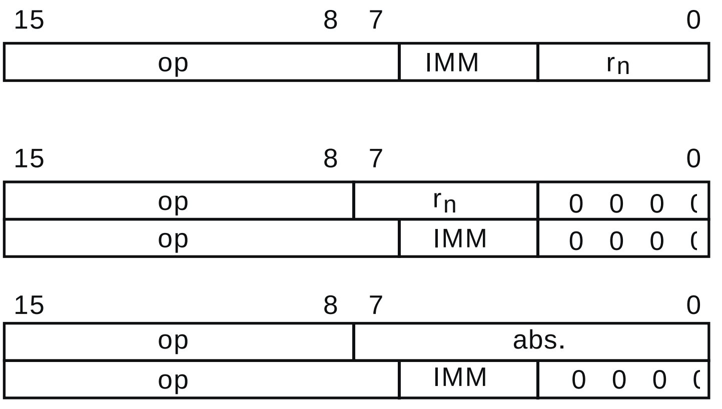
Figure 1-6. Machine-Language Coding of Arithmetic, Logic, and Shift Instruction Codes



Notation

op: Operation field
r_m, r_n: Register field
abs.: Absolute address
IMM: Immediate data

Figure 1-7. Machine-Language Coding of Bit Manipulation Instructions



BIAND, BIOR, BIXOR, BILD, BIST

Operand: register direct (Rn)
Bit No.: immediate (#xx:3)

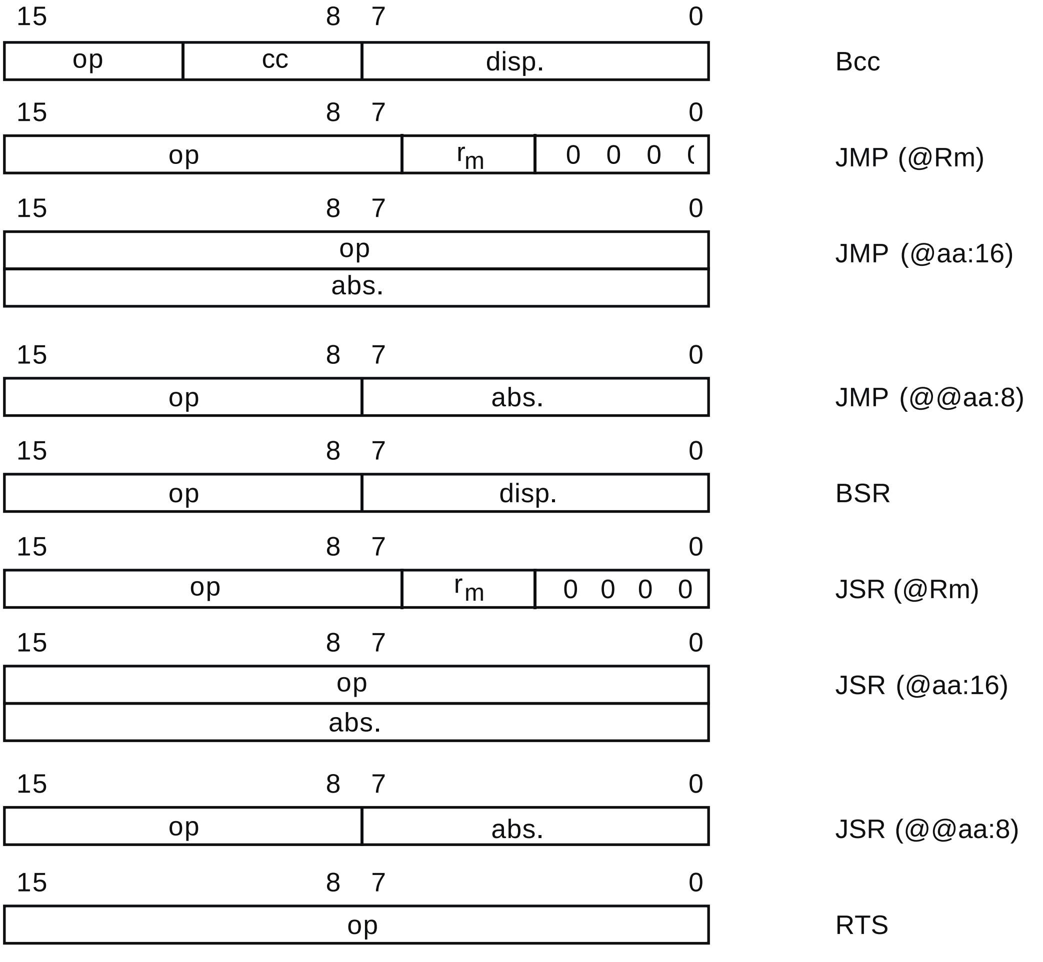
Operand: register indirect (@Rn)
Bit No.: immediate (#xx:3)

Operand: absolute (@aa:8)
Bit No.: immediate (#xx:3)

Notation

op: Operation field
r_m, r_n: Register field
abs.: Absolute address
IMM: Immediate data

Figure 1-7. Machine-Language Coding of Bit Manipulation Instructions (Cont.)



Notation

op:	Operation field
cc:	Condition field
r_m:	Register field
disp.:	Displacement
abs.:	Absolute address

and to the I/O port registers.

Example 1: BCLR is executed to clear bit 0 in the port 4 data direction register (P4DDR) under the following conditions.

$P4_7$:	Input pin, Low, MOS pull-up transistor on
$P4_6$:	Input pin, High, MOS pull-up transistor off
$P4_5$ – $P4_0$:	Output pins, Low

The intended purpose of this BCLR instruction is to switch $P4_0$ from output to input.

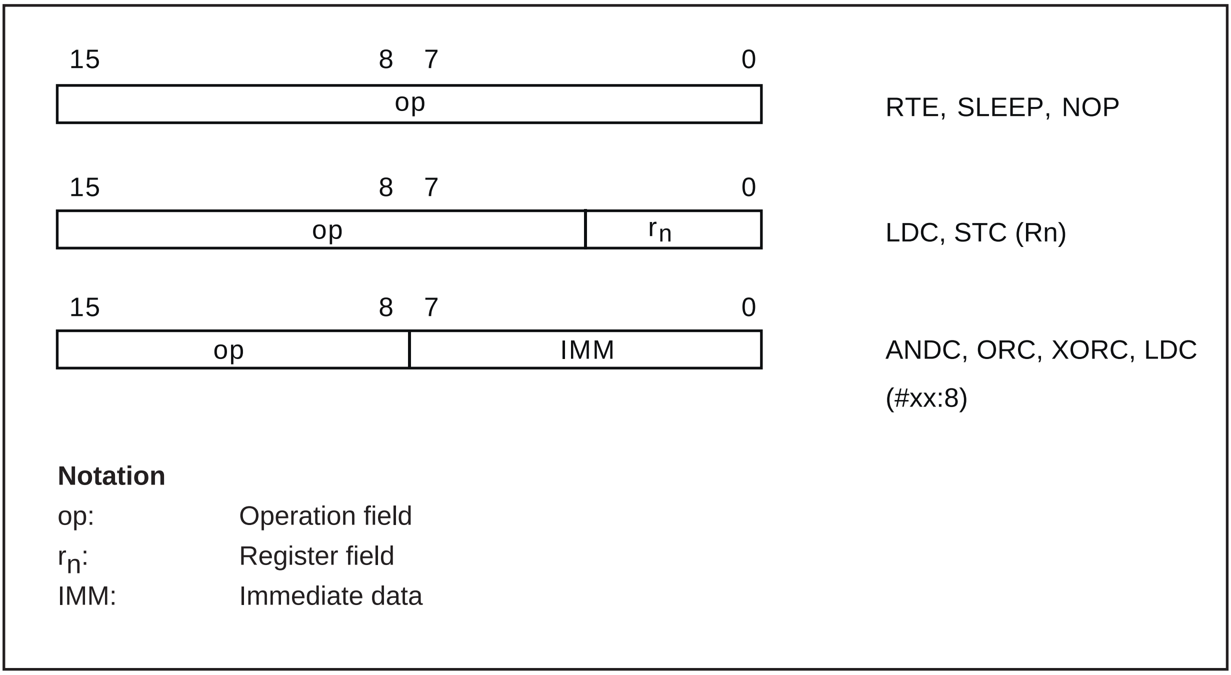


Figure 1-9. Machine-Language Coding of System Control Instructions

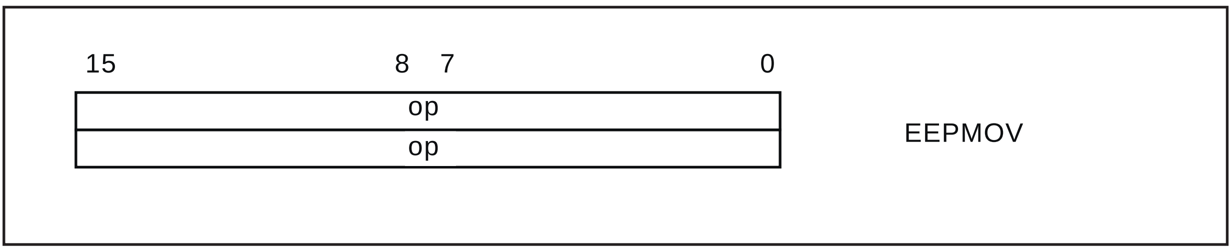


Figure 1-10. Machine-Language Coding of Block Data Transfer Instruction

1.3.4 Addressing Modes and Effective Address Calculation

Table 1-11 lists the eight addressing modes and their assembly-language notation. Each instruction can use a specific subset of these addressing modes.

Table 1-11. 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 register 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 registers 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 hardware manual of the specific chip 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.

Calculation of Effective Address: Table 1-12 shows how the H8/300 calculates effective addresses in each addressing mode.

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.

Effective Address Calculation

Table 1-12 explains how the effective address is calculated in each addressing mode.

Table 1-12, Effective Address Calculation (1)

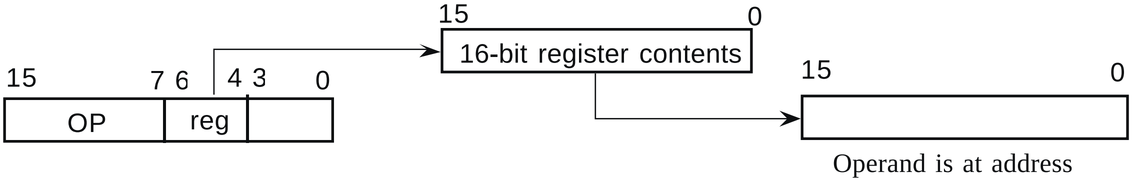
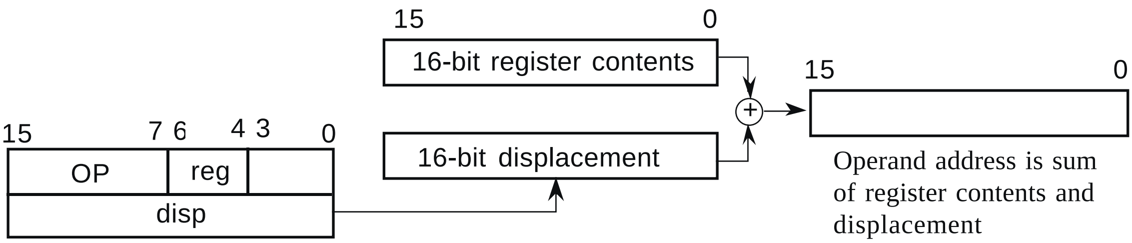
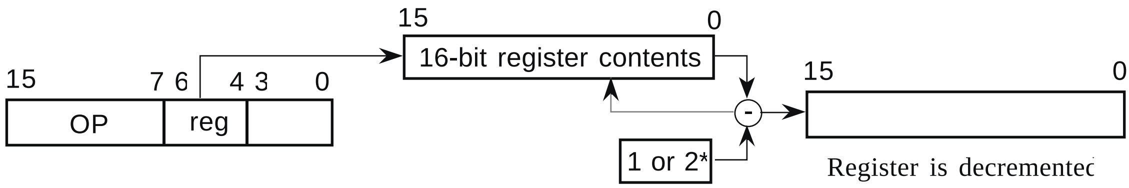
No.	Addressing mode, instruction format	Effective address calculation	Effective address
1	Register direct Rn. 	None	Operand are contained in registers m and n
2	Register indirect @Rn 		Operand is at address indicated by register

Table 1-12, Effective Address Calculation (2)

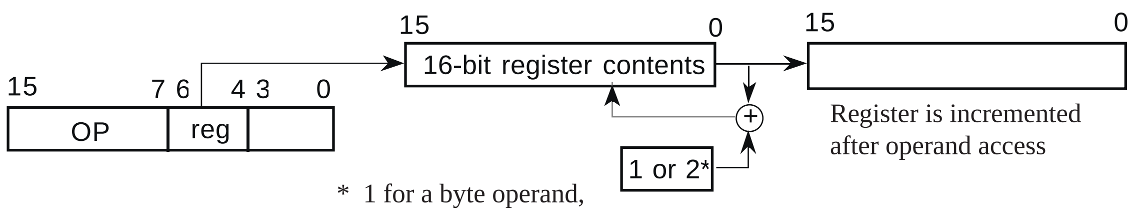
No.	Addressing mode, instruction format	Effective address calculation	Effective address
3	Register indirect with displacement @(d:16, Rn)		Operand address is sum of register contents and displacement



No.	Addressing mode, instruction format	Effective address calculation	Effective address
4	Register indirect with pre-decrement @-Rn		Register is decremented before operand access

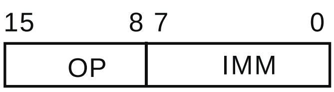


No.	Addressing mode, instruction format	Effective address calculation	Effective address
	Register indirect with post-increment @Rn+		Register is incremented after operand access



* 1 for a byte operand, 2 for a word operand

No.	Addressing mode, instruction format	Effective address calculation	Effective address
5	Immediate #xx:8.	None	Operand is 1-byte immediate data



No.	Addressing mode, instruction format	Effective address calculation	Effective address
	Immediate #xx:16	None	Operand is 2-byte immediate data

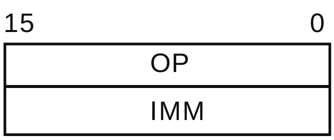
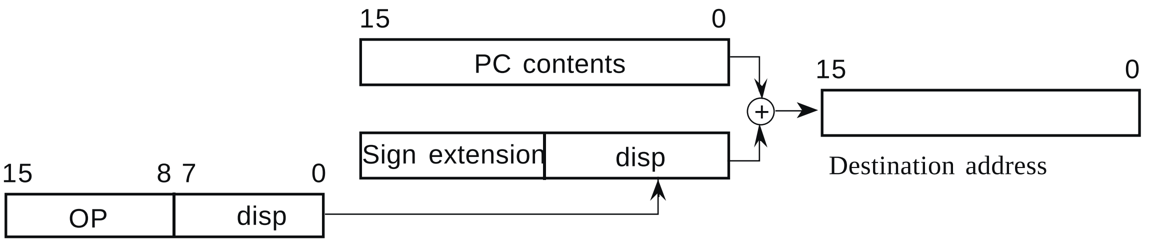


Table 1-12, Effective Address Calculation (3)

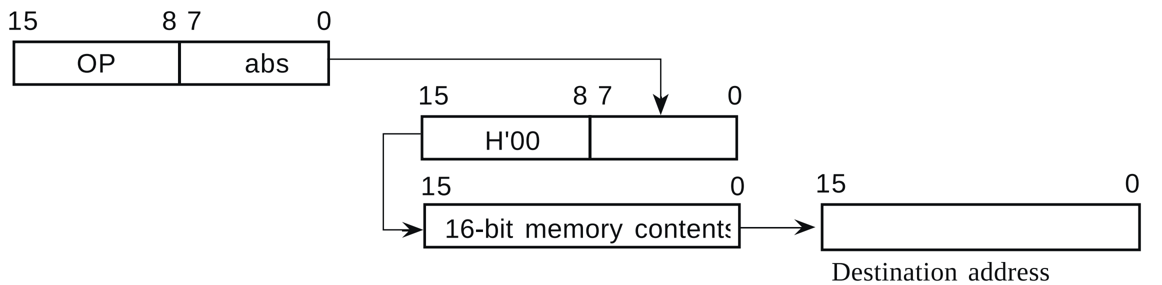
No.	Addressing mode, instruction format	Effective address calculation	Effective address
6	Absolute address @aa:8	None	

Absolute address
@aa:16

No.	Addressing mode, instruction format	Effective address calculation	Effective address
7	PC-relative @(d:8, PC)		



No.	Addressing mode, instruction format	Effective address calculation	Effective address
8	Memory indirect @@aa:8		



reg, regm, regn: General register
op: Operation field
disp: Displacement
abs: Absolute address
IMM: Immediate data

Section 2. Instruction Set

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

ADD (ADD binary) (byte) **ADD**

<Operation>

Rd + (EAs) → Rd

<Assembly-Language Format>

ADD.B <EAs>, Rd

<Examples>

```
ADD.B R0H, R1H
ADD.B #H'64, R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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 8-bit general register .

The source operand can be an 8-bit register value or immediate byte data.

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

<Instruction Formats>

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

Operation: Describes the instruction in symbolic notation. The following symbols are used.

Symbol	Meaning
(EAs)	Source operand
(EAd)	Destination operand
Rs, Rd, Rn	8-bit or 16-bit general register (s—source; d—destination)
#xx:3, #xx:8, #xx:16	3-bit, 8-bit, or 16-bit immediate data
d:8, d:16	8-bit or 16-bit displacement
PC	Program counter
SP	Stack pointer
CCR	Condition code register
Z	Zero flag in CCR
C	Carry flag in CCR
$\rightarrow$	The result of the operation on the left is assigned to the operand on the right (For compare instructions, the resulting condition code is assigned.)
$+$	Addition
$-$	Subtraction
$\times$	Multiplication
$\div$	Division
$\wedge$	AND logical
$\vee$	OR logical
$\oplus$	Exclusive OR logical
$\leftrightarrow$	Exchange
$\neg$	Not

Assembly-Language Format: The assembly-language coding of the instruction. An example is:

ADD. B <EAs>, Rd

Mnemonic Size Source Destination

The operand size is indicated by the letter B (byte) or W (word). The size is indicated explicitly in this manual, but for instructions that permit only one size, the size designation can be omitted in source-program coding.

The abbreviation EAs or EAd (effective address of source or destination) is used for operands that permit more than one addressing mode.

Examples: Examples of the assembly-language coding of the instruction are given.

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 the 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.
*	Undetermined; the flag is left in an unpredictable state.

Description: A detailed explanation is given of the action of the instruction.

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

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

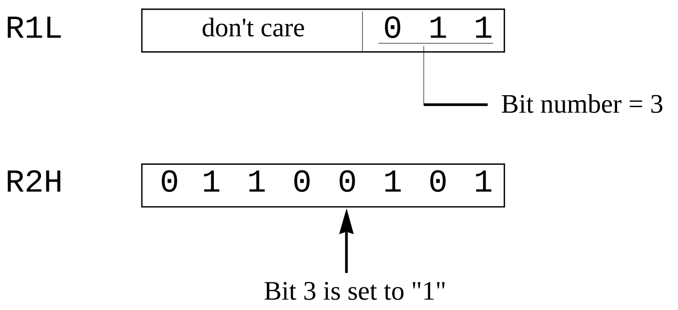
r^s, r^d, or r^n	**Register**
0 0 0	R0
0 0 1	R1
:	:
1 1 1	R7

8-Bit registers

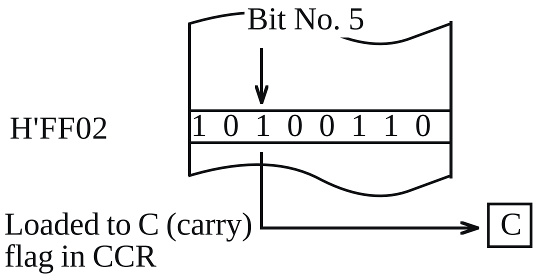
r^s, r^d, or r^n	**Register**
0 0 0 0	R0H
0 0 0 1	R1H
:	:
0 1 1 1	R7H
1 0 0 0	R0L
1 0 0 1	R1L
:	:
1 1 1 1	R7L

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 Appendix C.

<Operation>

Rd + (EAs) → Rd

<Assembly-Language Format>

`ADD.B` <EAs>, Rd

<Examples>

```
ADD.B R0H, R1H
ADD.B #H'64, R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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 8-bit general register .

The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

Rd + Rs → Rd

<Assembly-Language Format>

`ADD.W` Rs, Rd

<Examples>

```
ADD.W R0, R1
```

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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>

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

<Operation>

Rd + 1 → Rd

Rd + 2 → Rd

<Assembly-Language Format>

`ADDS #1,` Rd

`ADDS #2,` Rd

<Examples>

```
ADDS #1, R4
ADDS #2, R5
```

<Operand Size>

Word

<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: Previous value remains unchanged.

<Description>

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

<Instruction Formats>

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.

<Operation>

Rd + (EAs) + C → Rd

<Assembly-Language Format>

`ADDX` <EAs>, Rd

<Examples>

```
ADDX R0L, R1L
ADDX #H'0A, R2H
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

I: Previous value remains unchanged.

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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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 8-bit general register.

The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

Rd $\wedge$ (EAs) $\rightarrow$ Rd

<Assembly-Language Format>

`AND` <EAs>, Rd

<Examples>

```
AND R6H, R6L
AND #H'FD, R0H
```

<Operand Size>

Byte

<Condition Code>

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

The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

$CCR \wedge \#IMM \rightarrow CCR$

<Assembly-Language Format>

`ANDC` #xx:8, CCR

<Examples>

```
ANDC #H'7F, CCR
```

<Operand Size>

Byte

<Condition Code>

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>

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

BAND (Bit AND)

<Operation>

C ∧ (<Bit No.> of <EAd>) → C

<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: ANDed with the specified bit.

<Assembly-Language Format>

BAND #xx:3, <EAd>

<Examples>

```
BAND #0, R1L
BAND #4, @R3
BAND #7, @H'FFE0:8
```

<Operand Size>

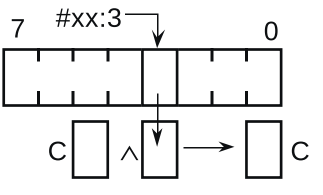
Byte

<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



The value of the specified bit is not changed.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code 1st byte	2nd byte	3rd byte	4th byte	No. of states
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.

<Operation>

If cc then

PC + d:8 → PC

else next;

<Assembly-Language Format>

`Bcc` d:8

└→ Condition code field

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

<Examples>

```
BHI H'42
BEQ H'–7E
```

<Operand Size>

—

<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: Previous value remains unchanged.

<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 available 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 ≤ Y (Unsigned)
BCC (BHS)	0 1 0 0	Carry Clear (High or Same)	C = 0	X ≥ 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 ≠ 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 ≥ 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 ≤ Y (Signed)

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

<Instruction Formats>

Adressing 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.

<Operation>

0 → (<Bit No.> of <EAd>)

<Assembly-Language Format>

BCLR #xx:3, <EAd>

BCLR Rn, <EAd>

<Examples>

```
BCLR #0, ROL
BCLR #1, @R5
BCLR R6L, @H'FFCO:8
```

<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.

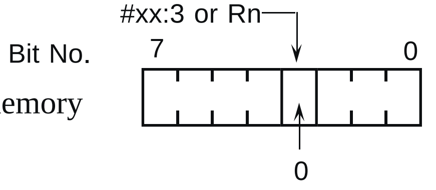
C: Previous value remains unchanged.

<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.

<EAd>*→ Byte data in register or memory



*Register direct, register indirect, or absolute addressing.

<Instruction Formats>

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

<Operation>

C ∧ [¬ (<Bit No.> of <EAd>)] → C

<Assembly-Language Format>

`BIAND` #xx:3, <EAd>

<Examples>

```
BIAND #0, R1H
BIAND #2, @R5
BIAND #4, @H'FFDE:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

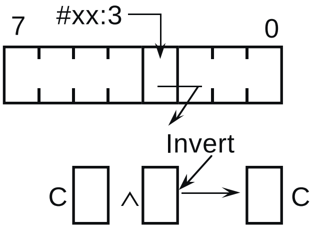
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: ANDed with the inverse of the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code 1st byte	2nd byte	3rd byte	4th byte	No. of states
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.

<Operation>

¬ (<Bit No.> of <EAd>) → C

<Assembly-Language Format>

`BILD` #xx:3, <EAd>

<Examples>

```
BILD #3, R4L
BILD #5, @R5
BILD #7, @H'FFA2:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

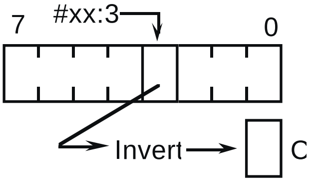
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Loaded with the inverse of the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

<Instruction Formats>

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.

<Operation>

C ∨ [¬ (<Bit No.> of <EAd>)] → C

<Assembly-Language Format>

BIOR #xx:3, <EAd>

<Examples>

```
BIOR #6, R1H
BIOR #3, @R2
BIOR #0, @H'FFF0:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

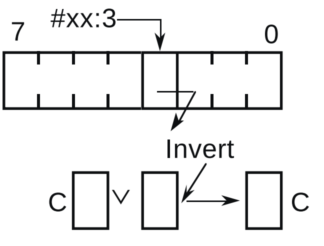
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: ORed with the inverse of the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code: 1st byte		2nd byte		3rd byte		4th byte		No. of states
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.

<Operation>

¬ C → (<Bit No.> of <EAd>)

<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: Previous value remains unchanged.

<Assembly-Language Format>

`BIST` #xx:3, <EAd>

<Examples>

```
BIST #0, R0L
BIST #6, @R3
BIST #7, @H'FFBB:8
```

<Operand Size>

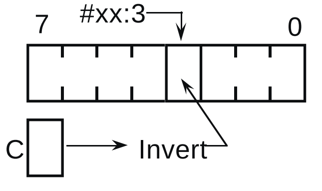
Byte

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The values of the unspecified bits are not changed.

<Instruction Formats>

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.

<Operation>

C ⊕ [¬ (<Bit No.> of <EAd>)] → C

<Assembly-Language Format>

`BIXOR` #xx:3, <EAd>

<Examples>

```
BIXOR #1, R4L
BIXOR #2, @R5
BIXOR #3, @H'FF60:8
```

<Operand Size>

Byte

<Condition Code>

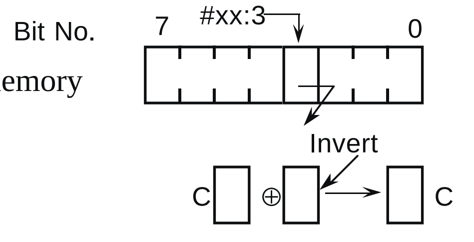
I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

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

<Description>

This instruction exclusive-ORs the inverse of a specified bitwith 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.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code 1st byte	2nd byte	3rd byte	4th byte	No. of states
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.

BLD (Bit LoaD) BLD

<Operation>

(<Bit No.> of <EAd>) → C

<Assembly-Language Format>

```
BLD #xx:3, <EAd>
```

<Examples>

```
BLD #1, R3H
BLD #2, @R2
BLD #4, @H'FF90:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

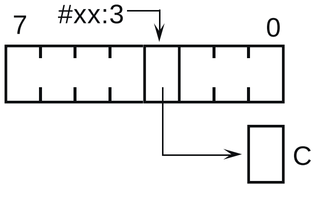
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Loaded with the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code: 1st byte		2nd byte		3rd byte		4th byte		No. of states
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.

<Operation>

¬ (<Bit No.> of <EAd>)
→ (<Bit No.> of <EAd>)

<Assembly-Language Format>

`BNOT` #xx:3, <EAd>

`BNOT` Rn, <EAd>

<Examples>

```
BNOT #7, R1H
BNOT R1L, @R6
BNOT #3, @H'FFB4:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	—

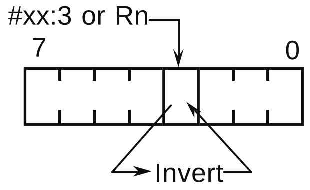
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



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

*Register direct, register indirect, or absolute addressing.

<Instruction Formats>

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

<Operation>

C $\vee$ (<Bit No.> of <EAd>) → C

<Assembly-Language Format>

`BOR` #xx:3, <EAd>

<Examples>

```
BOR #5, R2H
BOR #4, @R1
BOR #5, @H'FFB6:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

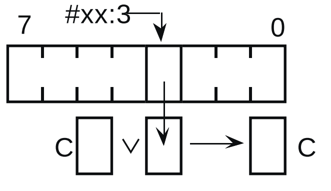
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: ORed with the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

*Register direct, register indirect, or absolute addressing.

<Instruction Formats>

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

<Operation>

1 → (<Bit No.> of <EAd>)

<Assembly-Language Format>

BSET #xx:3,<EAd>

BSET Rn,<EAd>

<Examples>

```
BSET #3, R2L
BSET R2H, @R6
BSET #7, @H'FFE4:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	—

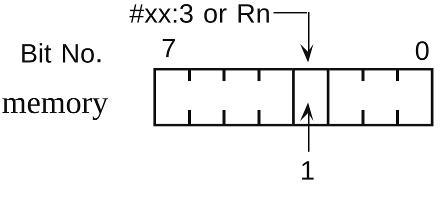
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

<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.

<EAd>*→ Byte data in register or memory



*Register direct, register indirect, or absolute addressing.

<Instruction Formats>

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

<Operation>

PC → @–SP

PC + d:8 → PC

<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: Previous value remains unchanged.

<Assembly-Language Format>

BSR d:8

<Examples>

```
BSR H'76
```

<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>

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

<Operation>

C → (<Bit No.> of <EAd>)

<Assembly-Language Format>

BST #xx:3, <EAd>

<Examples>

```
BST #7, R4L
BST #2, @R3
BST #6, @H'FFD1:8
```

<Operand Size>

Byte

<Condition Code>

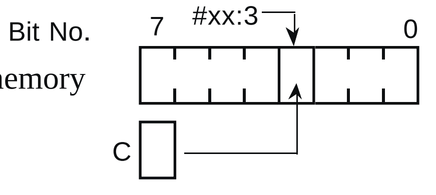
I		H		N	Z	V	C
—	—	—	—	—	—	—	—

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: 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.

<EAd>*→ Byte data in register or memory



<Instruction Formats>

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.

<Operation>

¬ (<Bit No.> of <EAd>) → Z

<Assembly-Language Format>

BTST #xx:3, <EAd>

BTST Rn, <EAd>

<Examples>

```
BTST #4, R6L
BTST R1H, @R5
BTST #7, @H'FF6C:8
```

<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.

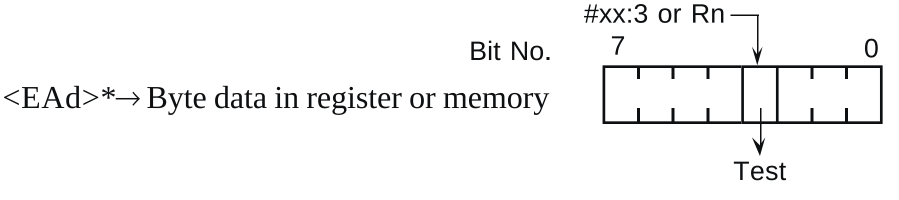
Z: See to "1" if the specified bit is zero; otherwise cleared to "0".

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.

*Register direct, register indirect, or absolute addressing.

<Instruction Formats>

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

<Operation>

C ⊕ (<Bit No.> of <EAd>) → C

<Assembly-Language Format>

`BXOR` #xx:3, <EAd>

<Examples>

```
BXOR #4, R6H
BXOR #2, @R0
BXOR #1, @H'FFA0:8
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	—	—	—	↕

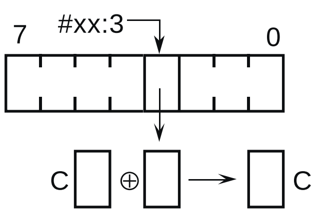
I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Exclusive-ORed with the specified bit.

<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.

Bit No.

<EAd>*→ Byte data in register or memory



The value of the specified bit is not changed.

*Register direct, register indirect, or absolute addressing.

BXOR (Bit eXclusive OR) BXOR

<Instruction Formats>

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

<Operation>

Rd – (EAs); set condition code

<Assembly-Language Format>

`CMP.B` <EAs>, Rd

<Examples>

```
CMP.B #H'E5, R1H
CMP.B R3L, R4L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs; otherwise cleared to "0."
C: Set to "1" if 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>

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

<Operation>

Rd – Rs; set condition code

<Assembly-Language Format>

`CMP.W` Rs, Rd

<Examples>

```
CMP.W R5, R6
```

<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" if an overflow occurs; otherwise cleared to "0."
C: Set to "1" if 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>

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

<Operation>

Rd (decimal adjust) → Rd

<Assembly-Language Format>

`DAA` Rd

<Examples>

`DAA R5L`

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	*	—	↕	↕	*	↕

I: Previous value remains unchanged.

H: Unpredictable.

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

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

V: Unpredictable.

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

<Description>

Given that 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

<Instruction Formats>

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

<Operation>

Rd (decimal adjust) → Rd

<Assembly-Language Format>

`DAS` Rd

<Examples>

`DAS R0H`

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	*	—	↕	↕	*	—

I: Previous value remains unchanged.

H: Unpredictable.

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

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

V: Unpredictable.

C: Previous value remains unchanged.

<Description>

Given that 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>

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

<Operation>

Rd – 1 → Rd

<Assembly-Language Format>

`DEC` Rd

<Examples>

```
DEC R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	↕	—

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

V: Set to "1" if an overflow occurs (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 8-bit general register.

<Instruction Formats>

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

<Operation>

Rd ÷ Rs → Rd

<Assembly-Language Format>

`DIVXU` Rs, Rd

<Examples>

```
DIVXU R0L, R1
```

<Operand Size>

Byte

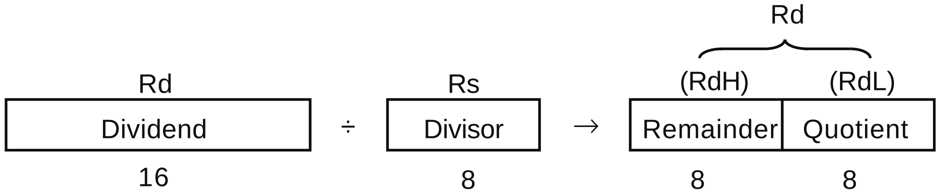
<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	—	—

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

<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 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>

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

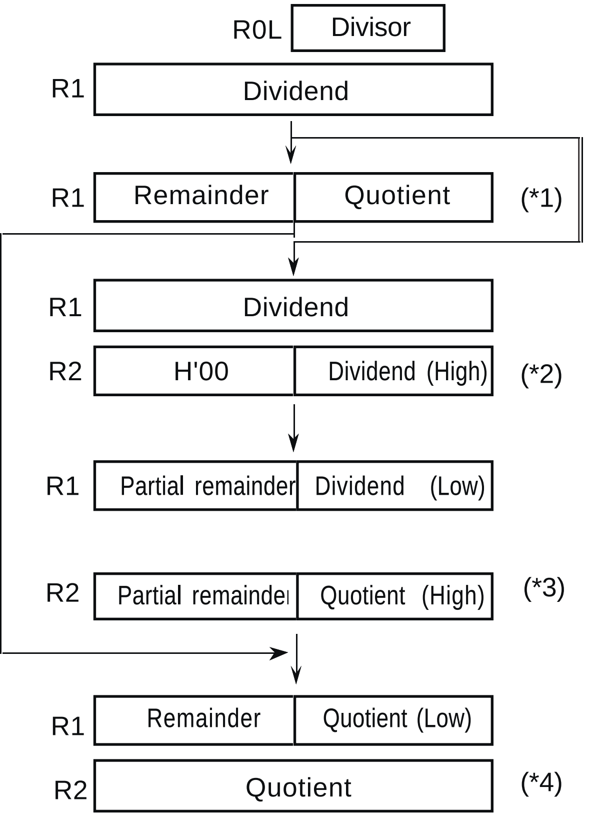
<Note: DIVXU Overflow>

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)
```



<Operation>

if R4L ≠ 0 then
 repeat $@R5+ \rightarrow @R6+$
 $R4L - 1 \rightarrow R4L$
 until R4L = 0
else next;

<Assembly-Language Format>

```
EEPMOV
```

<Examples>

```
MOV.B #H'20, R4L
MOV.W #H'FEC0, R5
MOV.W #H'6000, R6
EEPMOV
```

<Operand Size>

—

<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: 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.

Chips in the H8/300 Series having large on-chip EEPROM memories use this instruction to write data in the EEPROM. For details, see the hardware manual for the particular chip.

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

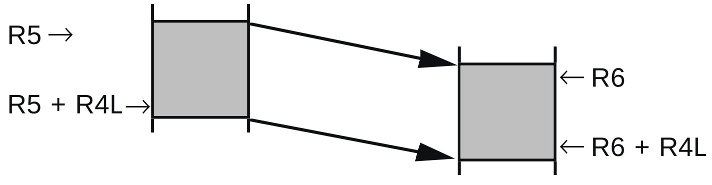
<Instruction Formats>

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	8+4n*

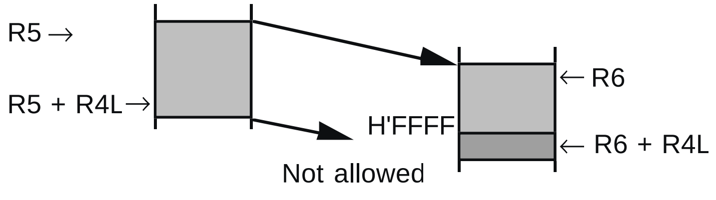
* n is the initial value in R4L ($0 \leq n \leq 255$). Although n bytes of data are transferred, memory is accessed 2(n+1) times, requiring 4(n+1) states.

Notes on EEPMOV Instruction

1. The EEPMOV instruction is a block data transfer instruction. It moves the number of bytes specified by R4L from the address specified by R5 to the address specified by R6.



2. When setting R4L and R6, make sure that the final destination address (R6 + R4L) does not exceed H'FFFF. The value in R6 must not change from H'FFFF to H'0000 during execution of the instruction.



<Operation>

Rd + 1 → Rd

<Assembly-Language Format>

`INC` Rd

<Examples>

```
INC R3L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	↕	—

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

V: Set to "1" if an overflow occurs (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 8-bit general register.

<Instruction Formats>

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

<Operation>

(EAd) → PC

<Assembly-Language Format>

JMP <EA>

<Examples>

```
JMP @R6
JMP @H'2000
JMP @@H'9A
```

<Operand Size>

—

<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: Previous value remains unchanged.

<Description>

This instruction branches unconditionally to a specified destination address.

The destination address must be even.

<Instruction Formats>

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

<Operation>

PC → @-SP

(EAd) → PC

<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: Previous value remains unchanged.

<Assembly-Language Format>

JSR <EA>

<Examples>

```
JSR @R3
JSR @H'1D26
JSR @@H'F0
```

<Operand Size>

—

<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>

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

<Operation>

(EAs) → CCR

<Assembly-Language Format>

LDC <EAs>, CCR

<Examples>

```
LDC #H'80, CCR
LDC R4H, CCR
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Loaded from the source operand.
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). The source operand can be an 8-bit general register or 8-bit immediate data. Bits 4 and 6 are loaded 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>

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

<Operation>

Rs → Rd

<Assembly-Language Format>

MOV.B Rs, Rd

<Examples>

MOV.B R1L, R2H

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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>

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

<Operation>

Rs → Rd

<Assembly-Language Format>

`MOV.W` Rs, Rd

<Examples>

```
MOV.W R3, R4
```

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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 register to a destination register and sets condition code flags according to the data value.

<Instruction Formats>

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

<Operation>

(EAs) → Rd

<Assembly-Language Format>

MOV.B <EAs>, Rd

<Examples>

```
MOV.B @R1, R2H
MOV.B @R5+, R0L
MOV.B @H'FFF1, R1H
MOV.B #H'A5, R3L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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 source operand can be memory contents or immediate data.

The MOV.B @R7+, Rd instruction should never be used, because it leaves an odd value in the stack pointer. This may result in loss of data, since the stack is always accessed a word at a time at an even address.

<Instruction Formats>

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	disp.		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	abs.		6

<Operation>

(EAs) → Rd

<Assembly-Language Format>

`MOV.W` <EAs>, Rd

<Examples>

```
MOV.W @R3, R4
MOV.W @(H'0004,R5), R6
MOV.W @R7+, R0
MOV.W #H'B00A, R1
```

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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.

<Instruction Formats>

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	IMM		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	disp.		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	abs.		6

<Operation>

Rs → (EAd)

<Assembly-Language Format>

`MOV.B` Rs, <EAd>

<Examples>

```
MOV.B R1L, @R0
MOV.B R3H, @(H'8001, R0)
MOV.B R5H, @–R4
MOV.B R6L, @H'FE77
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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. This may result in loss of data, since the stack is always accessed a word at a time at an even address.

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>

Addressing mode	Mnem.	Operands	Instruction code: 1st byte	2nd byte	3rd byte	4th byte	No. of states
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 | rs | 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

<Operation>

Rs → (EAd)

<Assembly-Language Format>

MOV.W Rs, <EAd>

<Examples>

```
MOV.W R3, @R4
MOV.W R2, @(H,0030,R5)
MOV.W R1, @–R7
MOV.W R0, @H'FED6
```

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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 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.

<Instruction Formats>

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

<Operation>

synchronization with the E clock
(EAs) → Rd

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

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

<Assembly-Language Format>

MOVFPE @aa:16, Rd

<Examples>

```
MOVFPE @H'FF81, R0H
```

<Operand Size>

Byte

<Description>

This instruction moves one byte of data from an absolute address location to a destination register, and sets the condition code flags according to the data value. The transfer is performed in synchronization with the E (enable) clock used by peripheral devices. The transfer requires 9 to 16 states, so the execution time is variable. For further information on basic timing, See the each *Hardware Manuals* .
This instruction should not be used with chips not having an E clock output pin or in single-chip mode.
When the source operand is located in on-chip memory or the on-chip register field, the MOVFPE instruction is identical in operation to MOV.B @aa:16, Rd.

Note that only 16-bit absolute addressing can be used, and word data cannot be transferred.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code						No. of states
			1st byte		2nd byte		3rd byte	4th byte	
Absolute address	MOVFPE	@aa:16, Rd	6	A	4	rd	abs.		13-20

<Operation>

synchronization with the E clock
Rs → (EAd)

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

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

<Assembly-Language Format>

MOVTPE Rs, @aa:16

<Examples>

```
MOVTPE R2L, @H'FF8D
```

<Operand Size>

Byte

<Description>

This instruction moves one byte of data from a source register to an absolute address location, and sets the condition code flags according to the data value. The transfer is performed in synchronization with the E (enable) clock used by peripheral devices. The transfer requires 9 to 16 states, so the execution time is variable. For further information on basic timing, see the each *Hardware Manuals* .

This instruction should not be used with chips not having an E clock output pin or in single-chip mode.

When the destination operand is located in on-chip memory or the on-chip register field, the MOVTPE instruction is identical in operation to MOV.B Rs, @aa:16.

Note that only 16-bit absolute addressing can be used, and word data cannot be transferred.

<Instruction Formats>

Addressing mode	Mnem.	Operands	Instruction code: 1st byte		2nd byte		3rd byte	4th byte	No. of states
Absolute address	MOVTPE	Rs, @aa:16	6	A	C	rs	abs.		13-20

<Operation>

Rd × Rs → Rd

<Assembly-Language Format>

`MULXU` Rs, Rd

<Examples>

```
MULXU R0H, R3
```

<Operand Size>

Byte

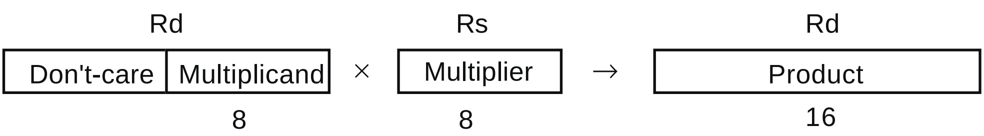
<Condition Code>

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: Previous value remains unchanged.

<Description>

This instruction performs 8-bit × 8-bit → 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>

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

<Operation>

0 – Rd → Rd

<Assembly-Language Format>

NEG Rd

<Examples>

NEG R0L

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs (the previous contents of the destination register was H'80); otherwise cleared to "0."

C: Set to "1" if 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>

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

<Operation>

PC + 2 → PC

<Assembly-Language Format>

NOP

<Examples>

NOP

<Operand Size>

—

<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: 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.
The NOP instruction can be used to fill in gaps in programs, or for software synchronization.

<Instruction Formats>

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

<Operation>

¬ Rd → Rd

<Assembly-Language Format>

`NOT` Rd

<Examples>

```
NOT R4L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

V: 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>

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

<Operation>

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

<Assembly-Language Format>

OR <EAs>, Rd

<Examples>

```
OR R2H, R3H
OR #H'C0, R0H
```

<Operand Size>

Byte

<Condition Code>

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 .

The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

$CCR \vee \#IMM \rightarrow CCR$

<Assembly-Language Format>

ORC #xx:8, CCR

<Examples>

```
ORC #H'80, CCR
```

<Operand Size>

Byte

<Condition Code>

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, including the nonmaskable interrupt (NMI), are deferred until after the next instruction.

<Instruction Formats>

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

<Operation>

@SP+ → Rn

<Assembly-Language Format>

`POP` Rn

<Examples>

`POP R1`

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

Z: Set to "1" if 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>

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

<Operation>

Rn → @–SP

<Assembly-Language Format>

PUSH Rn

<Examples>

PUSH R2

<Operand Size>

Word

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to "1" if the data value is negative; otherwise cleared to "0."
Z: Set to "1" if 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>

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

<Operation>

Rd (rotated left) → Rd

<Assembly-Language Format>

`ROTL` Rd

<Examples>

```
ROTL R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

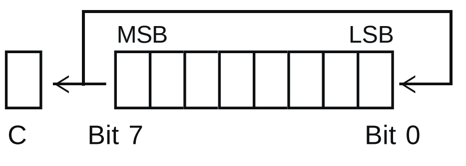
V: Cleared to "0."

C: Receives the previous value in bit 7.

<Description>

This instruction rotates an 8-bit general register one bit to the left. 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>

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

<Operation>

Rd (rotated right) → Rd

<Assembly-Language Format>

`ROTR` Rd

<Examples>

`ROTR R5L`

<Operand Size>

Byte

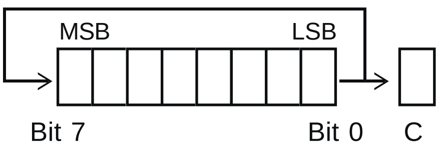
<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

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

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

<Operation>

Rd (rotated with carry left) → Rd

<Assembly-Language Format>

`ROTXL` Rd

<Examples>

`ROTXL R1H`

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

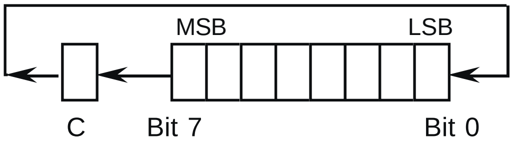
V: Cleared to "0."

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>

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

<Operation>

Rd (rotated with carry right) → Rd

<Assembly-Language Format>

ROTXR Rd

<Examples>

```
ROTXR R5L
```

<Operand Size>

Byte

<Condition Code>

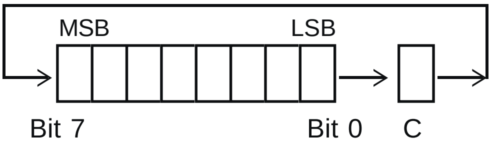
I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

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

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

<Operation>

@SP+ → CCR

@SP+ → PC

<Assembly-Language Format>

RTE

<Examples>

RTE

<Operand Size>

—

<Condition Code>

I		H		N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

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 interrupt-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>

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

<Operation>

@SP+ → PC

<Assembly-Language Format>

RTS

<Examples>

RTS

<Operand Size>

—

<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: 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>

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

<Operation>

Rd (shifted arithmetic left) → Rd

<Assembly-Language Format>

SHAL Rd

<Examples>

SHAL R5H

<Operand Size>

Byte

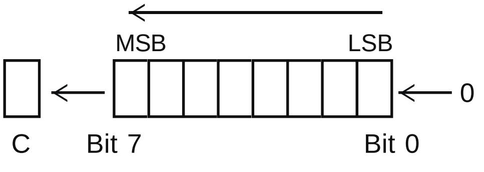
<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	↕	↕

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

<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>

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

<Operation>

Rd (shifted arithmetic right) → Rd

<Assembly-Language Format>

SHAR Rd

<Examples>

SHAR R5H

<Operand Size>

Byte

<Condition Code>

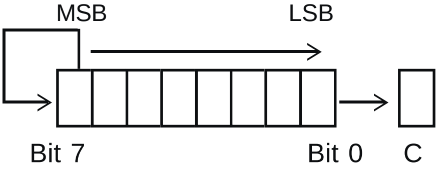
I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

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

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

<Operation>

Rd (shifted logical left) → Rd

<Assembly-Language Format>

`SHLL` Rd

<Examples>

```
SHLL R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

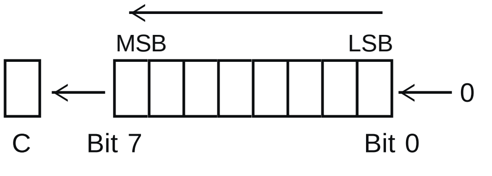
V: Cleared to "0."

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>

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

SHLR (SHift Logical Right) SHLR

<Operation>

Rd (shifted logical right) → Rd

<Assembly-Language Format>

`SHLR` Rd

<Examples>

`SHLR R3L`

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	—	—	↕	↕	0	↕

I: Previous value remains unchanged.

H: Previous value remains unchanged.

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

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

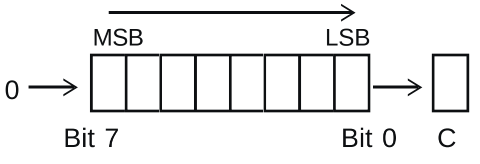
V: Cleared to "0."

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>

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

<Operation>

Program execution state → power-down mode

<Assembly-Language Format>

SLEEP

<Examples>

SLEEP

<Operand Size>

—

<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: 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 *Hardware Manual* for the particular chip.

<Instruction Formats>

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

<Operation>

CCR → Rd

<Assembly-Language Format>

STC CCR, Rd

<Examples>

```
STC CCR, R6H
```

<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: 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>

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

<Operation>

Rd – Rs → Rd

<Assembly-Language Format>

`SUB.B` Rs, Rd

<Examples>

```
SUB.B R0L, R2L
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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`
`SUBX` #(Imm – 1), Rd

(2) `ADD` #(0 – Imm), Rd
`XORC #H'01, CCR`

<Instruction Formats>

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

<Operation>

`Rd - Rs → Rd`

<Assembly-Language Format>

`SUB.W Rs, Rd`

<Examples>

`SUB.W R0, R1`

<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" if an overflow occurs; otherwise cleared to "0."

C: Set to "1" if 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>

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

<Operation>

Rd – 1 → Rd

Rd – 2 → Rd

<Assembly-Language Format>

`SUBS #1,` Rd

`SUBS #2,` Rd

<Examples>

```
SUBS #1, R3
SUBS #2, R5
```

<Operand Size>

Word

<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: Previous value remains unchanged.

<Description>

This instruction subtracts the immediate value 1 or 2 from word data in a general register. Differing from the SUB instruction, it does not affect the condition code flags.
The SUBS instruction does not permit byte operands.

<Instruction Formats>

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

<Operation>

Rd – (EAs) – C $\rightarrow$ Rd

<Assembly-Language Format>

SUBX <EAs>, Rd

<Examples>

```
SUBX R0L, R3L
SUBX #H'32, R5H
```

<Operand Size>

Byte

<Condition Code>

I		H		N	Z	V	C
—	—	$\updownarrow$	—	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$

I: Previous value remains unchanged.

H: Set to "1" if 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: Previous value remains unchanged when the result is zero; otherwise cleared to "0."

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

C: Set to "1" if there is a borrow from bit 7; 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.
The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

$Rd \oplus (EAs) \rightarrow Rd$

<Assembly-Language Format>

`XOR` <EAs>, Rd

<Examples>

```
XOR R0H, R1H
XOR #H'F0, R2L
```

<Operand Size>

Byte

<Condition Code>

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

The source operand can be an 8-bit register value or immediate byte data.

<Instruction Formats>

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

<Operation>

CCR $\oplus$ #IMM $\rightarrow$ CCR

<Assembly-Language Format>

XORC #xx:8, CCR

<Examples>

```
XORC #H'50, CCR
```

<Operand Size>

Byte

<Condition Code>

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>

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

Appendix A. Operation Code Map

The table in this appendix is a map of the operation codes contained in the first byte of the instruction code (bits 15 to 8 of the first instruction word).

Some pairs of instructions have indentical first bytes. These instructions are differentiated by the first bit of the second byte (bit 7 of the first instruction word).

Instruction when first bit of byte 2 (bit 7 of first instruction word) is "0."
Instruction when first bit of byte 2 (bit 7 of first instruction word) is "1."

Operation Code Map

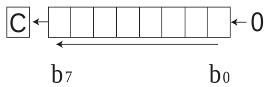
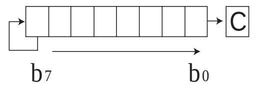
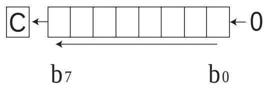
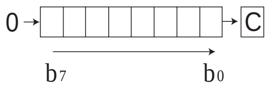
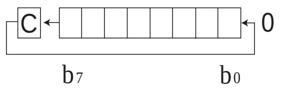
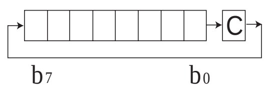
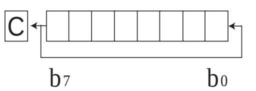
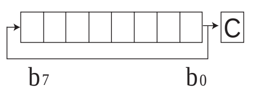
LO \ HI	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	SLEEP	STC	LDC	ORC	XORC	ANDC	LDC	ADD		INC	ADDS	MOV		ADDX	DAA
1	SHLL / SHAL	SHLR / SHAR	ROTXL / ROTL	ROTXR / ROTR	OR	XOR	AND	NOT / NEG	SUB		DEC	SUBS	CMP		SUBX	DAS
2	MOV															
3																
4	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT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

* The MOVFPE and MOVTPE instructions are identical to MOV instructions in the first byte and first bit of the second byte (bits 15 to 7 of the instruction word). They are differentiated in the next bit (bit 6 of the instruction word). See section 2.

Appendix. B Instruction Set List

Mnemonic	Operand size	Operation	Addressing mode/ instruction length									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	Rs16–2→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
MOVFPE @aa:16,Rd	B	@aa:16→Rd (Synchronization with E clock)						4				–	–	↕	↕	0	–	⑤
MOVTPE Rs,@aa:16	B	Rs→@aa:16 (Synchronization with E clock)						4				–	–	↕	↕	0	–	⑤

Mnemonic	Operand size	Operation	Addressing mode/ instruction length								Condition code						No. of states
			#xx: 8/16	Rn	@Rn	@(d:16, Rn)	@–Rn/@Rn+	@aa: 8/16	@(d:8, PC)	@@aa	I	H	N	Z	V	C	
ADD.B #xx:8,Rd	B	Rd8+#xx:8 →Rd8	2								–	↕	↕	↕	↕	↕	2
ADD.B Rs,Rd	B	Rs8+Rd8 →Rd8		2							–	↕	↕	↕	↕	↕	2
ADD.W Rs,Rd	W	Rs16+Rd16 →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							–	–	⑥	⑦	–	–	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	$\overline{Rd}$ →Rd		2							–	–	↕	↕	0	–	2

Mnemonic	Operand size	Operation	Addressing mode/ instruction length								Condition code						No. of states
			#xx: 8/16	Rn	@Rn	@(d:16, Rn)	@–Rn/@Rn+	@aa: 8/16	@(d:8, PC)	@@aa	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
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) ← $\overline{(\#xx{:}3 \text{ of } Rd8)}$		2							–	–	–	–	–	–	2
BNOT #xx:3,@Rd	B	(#xx:3 of @Rd16) ← $\overline{(\#xx{:}3 \text{ of } @Rd16)}$			4						–	–	–	–	–	–	8
BNOT #xx:3,@aa:8	B	(#xx:3 of @aa:8) ← $\overline{(\#xx{:}3 \text{ of } @aa{:}8)}$						4			–	–	–	–	–	–	8

Mnemonic	Operand size	Operation	Addressing mode/ instruction length								Condition code						No. of states
			#xx: 8/16	Rn	@Rn	@(d:16, Rn)	@–Rn/@Rn+	@aa: 8/16	@(d:8, PC)	@@aa	I	H	N	Z	V	C	
BNOT Rn,Rd	B	(Rn8 of Rd8) ← $\overline{\text{(Rn8 of Rd8)}}$		2							–	–	–	–	–	–	2
BNOT Rn,@Rd	B	(Rn8 of @Rd16) ← $\overline{\text{(Rn8 of @Rd16)}}$			4						–	–	–	–	–	–	8
BNOT Rn,@aa:8	B	(Rn8 of @aa:8) ← $\overline{\text{(Rn8 of @aa:8)}}$						4			–	–	–	–	–	–	8
BTST #xx:3,Rd	B	$\overline{\text{(\#xx:3 of Rd8)}}$ → Z		2							–	–	–	↕	–	–	2
BTST #xx:3,@Rd	B	$\overline{\text{(\#xx:3 of @Rd16)}}$ → Z			4						–	–	–	↕	–	–	6
BTST #xx:3,@aa:8	B	$\overline{\text{(\#xx:3 of @aa:8)}}$ → Z						4			–	–	–	↕	–	–	6
BTST Rn,Rd	B	$\overline{\text{(Rn8 of Rd8)}}$ → Z		2							–	–	–	↕	–	–	2
BTST Rn,@Rd	B	$\overline{\text{(Rn8 of @Rd16)}}$ → Z			4						–	–	–	↕	–	–	6
BTST Rn,@aa:8	B	$\overline{\text{(Rn8 of @aa:8)}}$ → 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	$\overline{\text{(\#xx:3 of Rd8)}}$ → C		2							–	–	–	–	–	↕	2
BILD #xx:3,@Rd	B	$\overline{\text{(\#xx:3 of @Rd16)}}$ → C			4						–	–	–	–	–	↕	6
BILD #xx:3,@aa:8	B	$\overline{\text{(\#xx:3 of @aa:8)}}$ → 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
BIST #xx:3,Rd	B	$\overline{\text{C}}$ → (#xx:3 of Rd8)		2							–	–	–	–	–	–	2
BIST #xx:3,@Rd	B	$\overline{\text{C}}$ → (#xx:3 of @Rd16)			4						–	–	–	–	–	–	8
BIST #xx:3,@aa:8	B	$\overline{\text{C}}$ → (#xx:3 of @aa:8)						4			–	–	–	–	–	–	8
BAND #xx:3,Rd	B	C ∧ (#xx:3 of Rd8) → C		2							–	–	–	–	–	↕	2
BAND #xx:3,@Rd	B	C ∧ (#xx:3 of @Rd16) → C			4						–	–	–	–	–	↕	6
BAND #xx:3,@aa:8	B	C ∧ (#xx:3 of @aa:8) → C						4			–	–	–	–	–	↕	6
BIAND #xx:3,Rd	B	C ∧ $\overline{\text{(\#xx:3 of Rd8)}}$ → C		2							–	–	–	–	–	↕	2
BIAND #xx:3,@Rd	B	C ∧ $\overline{\text{(\#xx:3 of @Rd16)}}$ → C			4						–	–	–	–	–	↕	6
BIAND #xx:3,@aa:8	B	C ∧ $\overline{\text{(\#xx:3 of @aa:8)}}$ → C						4			–	–	–	–	–	↕	6
BOR #xx:3,Rd	B	C ∨ (#xx:3 of Rd8) → C		2							–	–	–	–	–	↕	2
BOR #xx:3,@Rd	B	C ∨ (#xx:3 of @Rd16) → C			4						–	–	–	–	–	↕	6
BOR #xx:3,@aa:8	B	C ∨ (#xx:3 of @aa:8) → C						4			–	–	–	–	–	↕	6
BIOR #xx:3,Rd	B	C ∨ $\overline{\text{(\#xx:3 of Rd8)}}$ → C		2							–	–	–	–	–	↕	2

Mnemonic	Operand size	Operation		#xx: 8/16	Rn	@Rn	@(d:16, Rn)	@–Rn/@Rn+	@aa: 8/16	@(d:8, PC)	@@aa	I	H	N	Z	V	C	No. of states
				Addressing mode/ instruction length								Condition code						
BIOR #xx:3,@Rd	B	$C \vee \overline{(\#xx:3 \text{ of } @Rd16)} \rightarrow C$				4						–	–	–	–	–	↕	6
BIOR #xx:3,@aa:8	B	$C \vee \overline{(\#xx:3 \text{ of } @aa:8)} \rightarrow C$							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	$C \oplus \overline{(\#xx:3 \text{ of } Rd8)} \rightarrow C$			2							–	–	–	–	–	↕	2
BIXOR #xx:3,@Rd	B	$C \oplus \overline{(\#xx:3 \text{ of } @Rd16)} \rightarrow C$				4						–	–	–	–	–	↕	6
BIXOR #xx:3,@aa:8	B	$C \oplus \overline{(\#xx:3 \text{ of } @aa:8)} \rightarrow C$							4			–	–	–	–	–	↕	6
BRA d:8 (BTd:8)	–	PC←PC+d:8								2		–	–	–	–	–	–	4
BRN d:8 (BFd:8)	–	PC←PC+2								2		–	–	–	–	–	–	4
BHI d:8	–	if true then PC←PC+d:8 else next	C∨Z=0							2		–	–	–	–	–	–	4
BLS d:8	–		C∨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
BPL d:8	–		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	–	SP–2→SP PC→@SP PC←PC+d:8								2		–	–	–	–	–	–	6
JSR @Rn	–	SP–2→SP PC→@SP PC←Rn16				2						–	–	–	–	–	–	6

Mnemonic	Operand size	Operation	Addressing mode/ instruction length									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	
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
XORC #xx:8,CCR	B	CCR⊕#xx:8→CCR	2									↕	↕	↕	↕	↕	↕	2
NOP	–	PC←PC+2									2	–	–	–	–	–	–	2
EEPMOV	–	if R4L≠0 then Repeat @R5→@R6 R5+1→R5 R6+1→R6 R4L–1→R4L Until R4L=0 else next									4	–	–	–	–	–	–	④

Notes: The number of states is the number of states required for execution when the instructions and its operands are located in on-chip memory.

① Set to "1" when there is a carry or borrow from bit 11; otherwise cleared to "0."

② If the result is zero, the previous value of the flag is retained; otherwise the flag is cleared to "0."

③ Set to "1" if decimal adjustment produces a carry; otherwise cleared to "0."

④ The number of states is 4n+8 (n: initial value of R4L)

⑤ These instructions transfer data in synchronization with the E clock. The number of states varies depending on the synchronization delay.

⑥ Set to "1" if the divisor is negative; otherwise cleared to "0."

⑦ Set to "1" if the divisor is zero; otherwise cleared to "0."

Appendix C. Number of Execution States

The tables in this appendix can be used to calculate the number of states required for instruction execution. Table C-1 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 C-2 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:

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: Mode 1 (on-chip ROM disabled), stack located in external memory, 1 wait state inserted in external memory access.

1. BSET #0, @'FFC7
   From table C-2:
   $I = L = 2$, $J = K = M = N = 0$
   From table C-1:
   $S^I = 8$, $S^L = 3$
   Number of states required for execution = $2 \times 8 + 2 \times 3 = 22$

2. JSR @@ 30
   From table C-2:
   $I = 2$, $J = K = 1$, $L = M = N = 0$
   From table C-1:
   $S^I = S^J = S^K = 8$
   Number of states required for execution = $2 \times 8 + 1 \times 8 + 1 \times 8 = 32$

Table C-1. Number of States Taken by Each Cycle in Instruction Execution

Execution Status (instruction cycle)		Access Location		
		On-Chip Memory	On-Chip Reg. Field	External Memory
Instruction fetch	S_I	2	6	6 + 2m
Branch address read	S_J			
Stack operation	S_K			
Byte data access	S_L		3	3 + m
Word data access	S_M		6	6 + 2m
Internal operation	S_N	2		

Notes: 1. m: Number of wait states inserted in access to external device.

2. The byte data access cycle to an external device by the MOVFPE and MOVTPE instructions requires 9 to 16 states since it is synchronized with the E clock. See the *Hardware Manual* for timing details.

Table C-2. 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					
BCLR	BCLR #xx:3, Rd	1					
	BCLR #xx:3, @Rd	2			2		
	BCLR #xx:3, @aa:8	2			2		
	BCLR Rn, Rd	1					

Instruction	Mnemonic	Instruction Fetch	Branch Addr. Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
BCLR	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		
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		

Instruction	Mnemonic	Instruction Fetch	Branch Addr. Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
BSET	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					6
EEPMOV	EEPMOV	2			2n+2*1		
INC	INC.B Rd	1					
JMP	JMP @Rn	2					
	JMP @aa:16	2					1
	JMP @@aa:8	2	1				1
JSR	JSR @Rn	2		1			
	JSR @aa:16	2		1			1
	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		

Instruction	Mnemonic	Instruction Fetch	Branch Addr. Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
MOV	MOV.B @(d:16, Rs), Rd	2			1		
	MOV.B @Rs+, Rd	1			1		1
	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		1
	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	1
	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	1
	MOV.W Rs, @aa:16	2				1	
MOVFPE	MOVFPE @aa:16, Rd	2			1*2		
MOVTPE	MOVTPE Rs, @aa:16	2			1*2		
MULXU	MULXU.B Rs, Rd	1					6
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					
ROTL	ROTL.B Rd	1					
ROTR	ROTR.B Rd	1					
ROTXL	ROTXL.B Rd	1					
ROTXR	ROTXR.B Rd	1					
RTE	RTE	2		2			1
RTS	RTS	2		1			1

Instruction	Mnemonic	Instruction Fetch	Branch Addr. Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
		I	J	K	L	M	N
SHAL	SHAL.B Rd	1					
SHAR	SHAR.B Rd	1					
SHLL	SHLL.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					

Notes:

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

*2 Data access requires 9 to 16 states.

H8/300 Programming Manual

Publication Date: 1st Edition, December 1989
Published by: Business Planning Division
Semiconductor & Integrated Circuits
Hitachi, Ltd.
Edited by: Technical Documentation Group
Hitachi Kodaira Semiconductor Co., Ltd.