FUJITSU MICROELECTRONICS
CONTROLLER MANUAL

CM44-00201-3E

F^2MC-16LX

16-BIT MICROCONTROLLER

PROGRAMMING MANUAL

FUJITSU

F^2MC-16LX

16-BIT MICROCONTROLLER

PROGRAMMING MANUAL

FUJITSU MICROELECTRONICS LIMITED

PREFACE

■ Objectives and Intended Readership

The F^2MC-16LX series products are original 16-bit one-chip microcontrollers that support application specific ICs (ASICs). They are suitable for use in various types of industrial equipment, office-automation equipment, on-vehicle equipment, and other equipment that is required to operate at high speed in real-time mode.

Note: F^2MC is the abbreviation of FUJITSU Flexible Microcontroller.

■ Trademark

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

■ Configuration of this Manual

This manual contains the following 9 chapters and appendix.

CHAPTER 1 OVERVIEW OF THE F^2MC-16LX CPU CORE AND SAMPLE CONFIGURATION INCLUDING IT

This chapter briefly describes the configuration of the F^2MC-16LX CPU core, and presents a sample configuration of a device incorporating it.

CHAPTER 2 MEMORY SPACE

This chapter describes the memory spaces of the F^2MC-16LX CPU.

CHAPTER 3 DEDICATED REGISTERS

This chapter describes the dedicated registers of the F^2MC-16LX CPU.

CHAPTER 4 GENERAL-PURPOSE REGISTERS

This chapter describes the general-purpose registers of the F^2MC-16LX CPU.

CHAPTER 5 PREFIX CODES

The operation of an instruction can be modified by prefixing it with prefix code. This chapter explains the prefix codes.

CHAPTER 6 INTERRUPT HANDLING

This chapter describes the F^2MC-16LX interrupt handling functions and their operations.

CHAPTER 7 ADDRESSING

This chapter explains the addressing mode for each instruction of the F^2MC-16LX.

CHAPTER 8 INSTRUCTION OVERVIEW

This chapter explains the meanings of items and symbols used in explanations in "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS".

CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS

This chapter describes each execution instruction used in the assembler in a reference format.

APPENDIX

The appendix section includes lists of instructions used in the F^2MC-16LX, as well as the related instruction maps.

■ References

The following manuals should be referred along with this manual:

- F^2MC-16LX/16L/16/16H/16F Assembler Manual
- F^2MC-16LX Model-Specific Hardware Manual

READING THIS MANUAL

■ Page Layout

In this manual, an entire section is presented on a single page or spread whenever possible. You can thus view a section without having to flip pages.

The content of each section is summarized immediately below the title. You can obtain a rough overview of this product by reading through these summaries.

Also, higher level section headings are given in lower sections so that you can know to which section the text your are currently reading belongs.

CONTENTS

CHAPTER 1 OVERVIEW OF THE F^{2}MC-16LX CPU CORE AND SAMPLE CONFIGURATION INCLUDING IT 1

1.1 Overview of the F^{2}MC-16LX CPU Core 2
1.2 Sample Configuration of an F^{2}MC-16LX Device 3

CHAPTER 2 MEMORY SPACE 5

2.1 CPU Memory Space 6
2.2 Linear Addressing Mode 7
2.3 Bank Addressing Mode 9
2.4 Memory Space Divided into Banks and Value in Each Bank Register 11
2.5 Data Configuration of and Access to Multi-byte Data in Memory 12

CHAPTER 3 DEDICATED REGISTERS 15

3.1 F^{2}MC-16LX Dedicated Register Types 16
3.2 Accumulator (A) 18
3.3 User Stack Pointer (USP) and System Stack Pointer (SSP) 20
3.4 Processor Status (PS) 22
3.4.1 Interrupt Level Mask Register (ILM) 23
3.4.2 Register Bank Pointer (RP) 24
3.4.3 Condition Code Register (CCR) 25
3.5 Program Counter (PC) 26
3.6 Direct Page Register (DPR) 27
3.7 Bank Registers 28

CHAPTER 4 GENERAL-PURPOSE REGISTERS 29

4.1 Register Banks in RAM 30
4.2 Calling General-purpose Registers in RAM 31

CHAPTER 5 PREFIX CODES 33

5.1 Bank Select Prefix 34
5.2 Common Register Bank Prefix (CMR) 36
5.3 Flag Change Inhibit Prefix Code (NCC) 37
5.4 Constraints Related to the Prefix Codes 38

CHAPTER 6 INTERRUPT HANDLING 41

6.1 Interrupt Handling 42
6.2 Hardware Interrupt Operation Flow 44
6.3 Interrupt Handling Flowchart and Saving the Contents of Registers 45
6.4 Interrupt Vectors 47
6.5 Extended Intelligent I/O Service 48
6.5.1 Flowchart of Extended Iintelligent I/O Service Operation 50
6.5.2 Flowchart of Extended Intelligent I/O Service Application Procedure 51
6.6 Interrupt Control Register (ICR) 52

6.7 Meanings of the Bits of Interrupt Control Register (ICR) 53
6.8 Extended Intelligent I/O Service Descriptor (ISD) 55
6.9 Registers of Extended Intelligent I/O Service Descriptor 56
6.10 Exception Processing 58

CHAPTER 7 ADDRESSING 59
7.1 Effective Address Field 60
7.2 Direct Addressing 61
7.3 Indirect Addressing 63

CHAPTER 8 INSTRUCTION OVERVIEW 67
8.1 Instruction Overview 68
8.2 Symbols (Abbreviations) Used in Detailed Execution Instructions 70
8.3 Effective Address Field 72
8.4 Execution Cycles 73

CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS 77
9.1 Detailed Execution Instructions 78
9.1.1 ADD (Add Byte Data of Destination and Source to Destination) 79
9.1.2 ADDC (Add Byte Data of AL and AH with Carry to AL) 81
9.1.3 ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator) 82
9.1.4 ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator) 84
9.1.5 ADDDC (Add Decimal Data of AL and AH with Carry to AL) 86
9.1.6 ADDL (Add Long Word Data of Destination and Source to Destination) 87
9.1.7 ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer) 89
9.1.8 ADDW (Add Word Data of AL and AH to AL) 90
9.1.9 ADDW (Add Word Data of Destination and Source to Destination) 91
9.1.10 AND (And Byte Data of Destination and Source to Destination) 93
9.1.11 AND (And Byte Data of Immediate Data and Condition Code Register) 95
9.1.12 ANDL (And Long Word Data of Destination and Source to Destination) 97
9.1.13 ANDW (And Word Data of AH and AL to AL) 99
9.1.14 ANDW (And Word Data of Destination and Source to Destination) 100
9.1.15 ASR (Arithmetic Shift Byte Data of Accumulator to Right) 102
9.1.16 ASRL (Arithmetic Shift Long Word Data of Accumulator to Right) 104
9.1.17 ASRW (Arithmetic Shift Word Data of Accumulator to Right) 106
9.1.18 ASRW (Arithmetic Shift Word Data of Accumulator to Right) 108
9.1.19 BBcc (Branch if Bit Condition satisfied) 110
9.1.20 Bcc (Branch relative if Condition satisfied) 112
9.1.21 CALL (Call Subroutine) 114
9.1.22 CALLP (Call Physical Address) 116
9.1.23 CALLV (Call Vectored Subroutine) 118
9.1.24 CBNE (Compare Byte Data and Branch if not equal) 120
9.1.25 CLRB (Clear Bit) 122
9.1.26 CMP (Compare Byte Data of Destination and Source) 123
9.1.27 CMPL (Compare Long Word Data of Destination and Source) 125
9.1.28 CMPW (Compare Word Data of Destination and Source) 127

9.1.29 CWBNE (Compare Word Data and Branch if not Equal) 129
9.1.30 DBNZ (Decrement Byte Data and Branch if not "0") 131
9.1.31 DEC (Decrement Byte Data) 133
9.1.32 DECL (Decrement Long Word Data) 134
9.1.33 DECW (Decrement Word Data) 135
9.1.34 DIV (Divide Word Data by Byte Data) 137
9.1.35 DIVW (Divide Long Word Data by Word Data) 139
9.1.36 DIVU (Divide unsigned Word Data by unsigned Byte Data) 141
9.1.37 DIVUW (Divide unsigned Long Word Data by unsigned Word Data) 143
9.1.38 DWBNZ (Decrement Word Data and Branch if not Zero) 145
9.1.39 EXT (Sign Extend from Byte Data to Word Data) 147
9.1.40 EXTW (Sign Extend from Word Data to Long Word Data) 148
9.1.41 FILS (Fill String Byte) 149
9.1.42 FILSW (Fill String Word) 151
9.1.43 INC (Increment Byte Data (Address Specification)) 153
9.1.44 INCL (Increment Long Word Data) 154
9.1.45 INCW (Increment Word Data) 155
9.1.46 INT (Software Interrupt) 157
9.1.47 INT (Software Interrupt (Vector Specification)) 159
9.1.48 INT9 (Software Interrupt) 161
9.1.49 INTP (Software Interrupt) 163
9.1.50 JCTX (Jump Context) 165
9.1.51 JMP (Jump Destination Address) 167
9.1.52 JMPP (Jump Destination Physical Address) 168
9.1.53 LINK (Link and create new stack frame) 169
9.1.54 LSL (Logical Shift Byte Data of Accumulator to Left) 170
9.1.55 LSLL (Logical Shift Long Word Data of Accumulator to Left) 171
9.1.56 LSLW (Logical Shift Word Data of Accumulator to Left) 172
9.1.57 LSLW (Logical Shift Word Data of Accumulator to Left) 173
9.1.58 LSR (Logical Shift Byte Data of Accumulator to Right) 174
9.1.59 LSRL (Logical Shift Long Word Data of Accumulator to Right) 176
9.1.60 LSRW (Logical Shift Word Data of Accumulator to Right) 178
9.1.61 LSRW (Logical Shift Word Data of Accumulator to Right) 179
9.1.62 MOV (Move Byte Data from Source to Accumulator) 181
9.1.63 MOV (Move Byte Data from Accumulator to Destination) 183
9.1.64 MOV (Move Byte Immediate Data to Destination) 184
9.1.65 MOV (Move Byte Data from Source to Destination) 186
9.1.66 MOV (Move Byte Data from AH to Memory) 188
9.1.67 MOVB (Move Bit Data from Bit Address to Accumulator) 189
9.1.68 MOVB (Move Bit Data from Accumulator to Bit Address) 191
9.1.69 MOVEA (Move Effective Address to Destination) 193
9.1.70 MOVL (Move Long Word Data from Source to Accumulator) 194
9.1.71 MOVL (Move Long Word Data from Accumulator to Destination) 195
9.1.72 MOVN (Move Immediate Nibble Data to Accumulator) 196
9.1.73 MOVS (Move String Byte) 197
9.1.74 MOVSW (Move String Word) 199
9.1.75 MOVW (Move Word Data from Source to Accumulator) 201

9.1.76 MOVW (Move Word Data from Accumulator to Destination) 203
9.1.77 MOVW (Move Immediate Word Data to Destination) 205
9.1.78 MOVW (Move Word Data from Source to Destination) 207
9.1.79 MOVW (Move Immediate Word Data to io) 209
9.1.80 MOVW (Move Word Data from AH to Memory) 210
9.1.81 MOVX (Move Byte Data with Sign Extension from Source to Accumulator) 211
9.1.82 MUL (Multiply Byte Data of Accumulator) 213
9.1.83 MUL (Multiply Byte Data of Accumulator and Effective Address) 214
9.1.84 MULW (Multiply Word Data of Accumulator) 215
9.1.85 MULW (Multiply Word Data of Accumulator and Effective Address) 216
9.1.86 MULU (Multiply Unsigned Byte Data of Accumulator) 217
9.1.87 MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218
9.1.88 MULUW (Multiply Unsigned Word Data of Accumulator) 219
9.1.89 MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220
9.1.90 NEG (Negate Byte Data of Destination) 221
9.1.91 NEGW (Negate Word Data of Destination) 222
9.1.92 NOP (No Operation) 223
9.1.93 NOT (Not Byte Data of Destination) 224
9.1.94 NOTW (Not Word Data of Destination) 226
9.1.95 NRML (NORMALIZE Long Word) 227
9.1.96 OR (Or Byte Data of Destination and Source to Destination) 228
9.1.97 OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230
9.1.98 ORL (Or Long Word Data of Destination and Source to Destination) 232
9.1.99 ORW (Or Word Data of AH and AL to AL) 234
9.1.100 ORW (Or Word Data of Destination and Source to Destination) 235
9.1.101 POPW (Pop Word Data of Accumulator from Stack Memory) 237
9.1.102 POPW (Pop Word Data of AH from Stack Memory) 239
9.1.103 POPW (Pop Word Data of Program Status from Stack Memory) 240
9.1.104 POPW (Pop Registers from Stack Memory) 242
9.1.105 PUSHW (Push Word Data of Inherent Register to Stack Memory) 244
9.1.106 PUSHW (Push Registers to Stack Memory) 246
9.1.107 RET (Return from Subroutine) 248
9.1.108 RETI (Return from Interrupt) 249
9.1.109 RETP (Return from Physical Address) 251
9.1.110 ROLC (Rotate Byte Data of Accumulator with Carry to Left) 253
9.1.111 RORC (Rotate Byte Data of Accumulator with Carry to Right) 255
9.1.112 SBBS (Set Bit and Branch if Bit Set) 257
9.1.113 SCEQ (Scan String Byte Until Equal) 259
9.1.114 SCWEQ (Scan String Word Until Equal) 261
9.1.115 SETB (Set Bit) 263
9.1.116 SUB (Subtract Byte Data of Source from Destination to Destination) 264
9.1.117 SUBC (Subtract Byte Data of AL from AH with Carry to AL) 266
9.1.118 SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
9.1.119 SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269
9.1.120 SUBDC (Subtract Decimal Data of AL from AH with Carry to AL) 271

9.1.121 SUBL (Subtract Long Word Data of Source from Destination to Destination) 272
9.1.122 SUBW (Subtract Word Data of Source from Destination to Destination) 274
9.1.123 SUBW (Subtract Word Data of AL from AH to AL) 276
9.1.124 SWAP (Swap Byte Data of Accumulator) 277
9.1.125 SWAPW (Swap Word Data of Accumulator) 278
9.1.126 UNLINK (Unlink and Create New Stack Frame) 279
9.1.127 WBTc (Wait until Bit Condition Satisfied) 280
9.1.128 XCH (Exchange Byte Data of Source to Destination) 282
9.1.129 XCHW (Exchange Word Data of Source to Destination) 284
9.1.130 XOR (Exclusive Or Byte Data of Destination and Source to Destination) 286
9.1.131 XORL (Exclusive Or Long Word Data of Destination and Source to Destination) 288
9.1.132 XORW (Exclusive Or Word Data of AH and AL to AL) 290
9.1.133 XORW (Exclusive Or Word Data of Destination and Source to Destination) 291
9.1.134 ZEXT (Zero Extend from Byte Data to Word Data) 293
9.1.135 ZEXTW (Zero Extend from Word Data to Long Word Data) 294

APPENDIX 295

APPENDIX A Explanation of Instruction Lists 296
A.1 Items Used in Instruction Lists 297
A.2 Symbols Used in Instruction Lists 299
A.3 Effective Address Field 301
A.4 Calculating the Number of Execution Cycles 302
APPENDIX B F^{2}MC-16LX Instruction Lists (351 Instructions) 304
B.1 Transfer Instructions 305
B.2 Numeric Data Operation Instructions 307
B.3 Logical Data Operation Instruction 311
B.4 Shift Instruction 313
B.5 Branch Instructions 314
B.6 Other Instructions 316
APPENDIX C F^{2}MC-16LX Instruction Maps 319
C.1 Structure of the Instruction Map 320
C.2 Basic Page Map 322
C.3 Bit Operation Instruction Map 324
C.4 Character String Operation Instruction Map 326
C.5 2-byte Instruction Map 328
C.6 ea-type Instruction Map 330
C.7 MOVEA RWi, ea Instruction Map 340
C.8 MOV Ri, ea Instruction Map 342
C.9 MOVW RWi, ea Instruction Map 344
C.10 MOV ea, Ri Instruction Map 346
C.11 MOVW ea, RWi Instruction Map 348
C.12 XCH Ri, ea Instruction Map 350
C.13 XCHW RWi, ea Instruction Map 352

INDEX.......... 355

Main changes in this edition

Page	Changes (For details, refer to main body.)	
188	CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS 9.1.66 MOV (Move Byte Data from AH to Memory)	"● Assembler format:" is changed. (MOV @AL,AH / MOV @A,T → MOV @AL,AH)
210	CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS 9.1.80 MOVW (Move Word Data from AH to Memory)	"● Assembler format:" is changed. (MOVW @AL,AH / MOV @A,T → MOVW @AL,AH)
278	CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS 9.1.125 SWAPW (Swap Word Data of Accumulator)	"● Assembler format:" is changed. (SWAPW /XCHW A,T → SWAPW)
305	B.1 Transfer Instructions	"Table B-1 Transfer Instruction (Byte): 41 Instructions" is changed. (MOV @AL,AH / MOV @A,T → MOV @AL,AH)
306		"Table B-2 Transfer Instruction (Word/Long-word): 38 Instructions" is changed. (MOVW @AL,AH / MOVW @A,T → MOVW @AL,AH)
315	B.5 Branch Instructions	CBNE Instruction in "Table B-14 Branch 2: 19 Instructions" is changed. (CBNE ear,#imm8,rel byte(A) not equal to imm8 CBNE ear,#imm8,rel byte(A) not equal to imm8 → CBNE ear,#imm8,rel byte(ear) not equal to imm8 CBNE eam,#imm8,rel byte(eam) not equal to imm8)
		CWBNE Instruction in "Table B-14 Branch 2: 19 Instructions" is changed. (CWBNE ear,#imm16,rel CWBNE ear,#imm16,rel → CWBNE ear,#imm16,rel CWBNE eam,#imm16,rel)

Page	Changes (For details, refer to main body.)	
316	B.6 Other Instructions	"Table B-15 Other Control Systems (Byte/Word/Long-word): 28 Instructions" is changed. (+&→ *5)
		"Table B-15 Other Control Systems (Byte/Word/Long-word): 28 Instructions" is changed. (*5: (number of POP operations), or (number of PUSH operations))
317		"Table B-17 Accumulator Operation Instruction (Byte/Word): 6 Instructions" is changed. (SWAPW / XCHW A,T → SWAPW)
318		"Table B-18 String Instruction : 10 Instructions" is changed. (+& → *5), (+) → *8)

The vertical lines marked in the left side of the page show the changes.

CHAPTER 1 OVERVIEW OF THE F^{2}MC-16LX CPU CORE AND SAMPLE CONFIGURATION INCLUDING IT

This chapter briefly describes the configuration of the F^{2}MC-16LX CPU core, and presents a sample configuration of a device incorporating it.

1.1 Overview of the F^{2}MC-16LX CPU Core

1.2 Sample Configuration of an F^{2}MC-16LX Device

1.1 Overview of the F^{2}MC-16LX CPU Core

The F^{2}MC-16LX CPU core is an advanced 16-bit CPU designed for use in various types of industrial equipment, office automation equipment, on-vehicle equipment, and other equipment required to operate at high speed in real-time mode.

■ Overview of the F^{2}MC-16LX CPU Core

The F^{2}MC-16LX CPU core is an advanced 16-bit CPU designed for use in various types of industrial equipment, office automation equipment, on-vehicle equipment, and other equipment required to operate at high speed in real-time mode. The design of the F^{2}MC-16LX instruction set is optimized for use in controllers. The instructions can perform various types of control at high speed and at high efficiency. The F^{2}MC-16LX is a suitable CPU for processing 16-bit data. Some of its instructions can be used also for 32-bit data processing, because its CPU incorporates a 32-bit accumulator. The memory space of the F^{2}MC-16LX can be expanded up to 16 Mbytes. Each location in the memory space can be accessed using either a linear pointer or a bank method. The instruction set is based on the F^{2}MC-8 A-T architecture, but has been enhanced by adding instructions that support high-level language, extending the addressing mode, improving the multiplication and division instructions, and augmenting bit manipulation.

■ Features of the F^{2}MC-16LX CPU Core

● Minimum instruction execution time: 62.5 ns (with internal clock at 16 MHz)

● Memory space: Up to 16 Mbytes, accessible using either a linear or bank mode

● Instruction set optimized for use in controllers

- Cornucopia of data types: Bit, byte, word, and long word
- Extended addressing mode: 23 types
- High code efficiency
- Reinforcement of high-precision calculation (32-bit length) by means of a 32-bit accumulator

● Powerful interrupt functions

Interrupt priority levels: 8 levels (programmable)

● CPU-independent automatic transfer function

● Extended intelligent I/O service: Up to 16 channels

● Instruction supporting high-level language (C language) and multitasking

- Use of a system stack pointer
- Various pointers
- High symmetry of the instruction set
- Barrel shift instruction

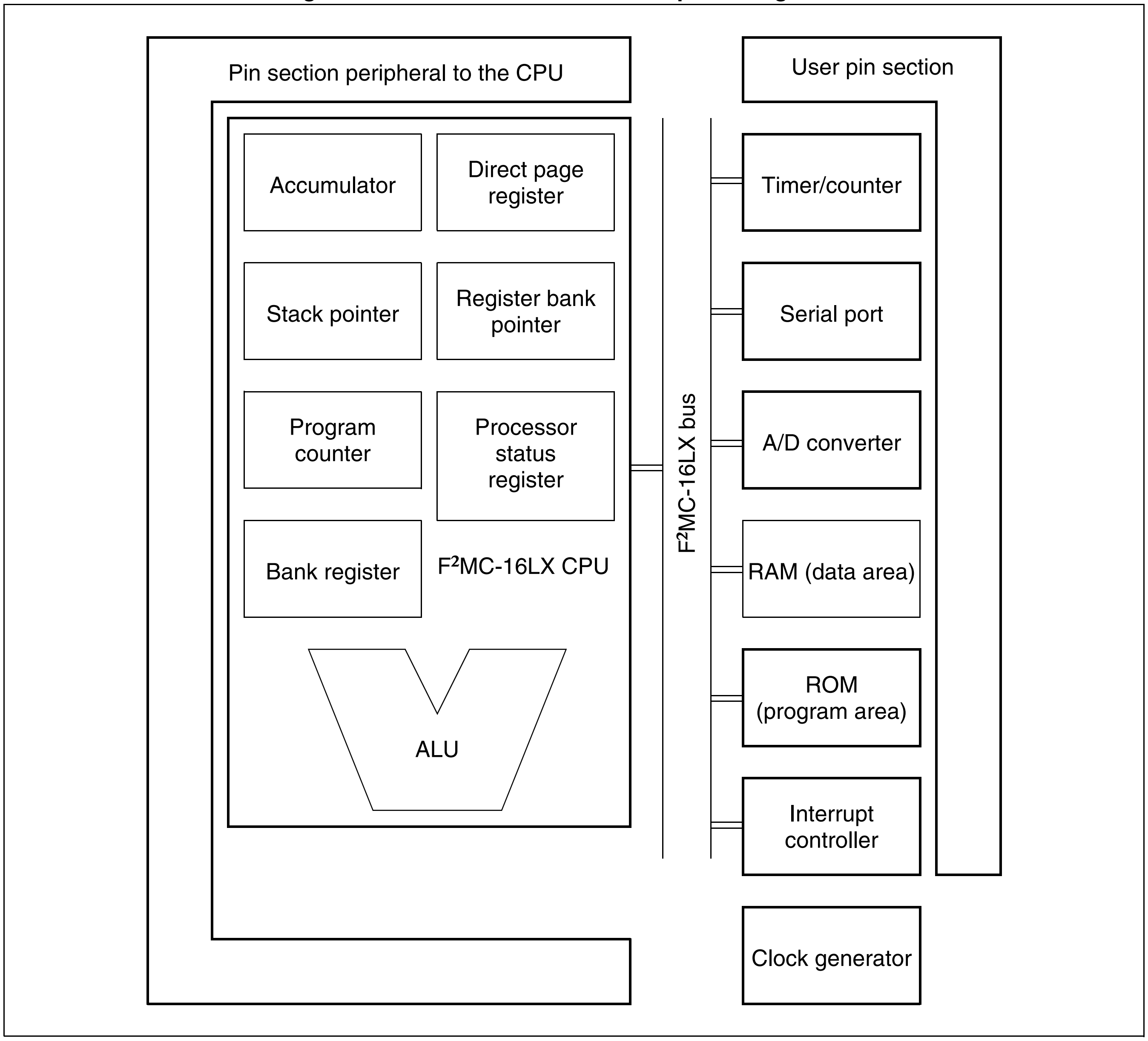
● Increased execution speed: Use of a 4-byte queue for waiting of instructions

1.2 Sample Configuration of an F^{2}MC-16LX Device

Figure 1.2-1 shows a sample configuration of an F^{2}MC-16LX device.

■ Sample Configuration of an F^{2}MC-16LX Device

Figure 1.2-1 F^{2}MC-16LX Device Sample Configuration



CHAPTER 2
MEMORY SPACE

This chapter describes memory spaces in the F^2MC-16LX CPU.

2.1 CPU Memory Space

2.2 Linear Addressing Mode

2.3 Bank Addressing Mode

2.4 Memory Space Divided into Banks and Value in Each Bank Register

2.5 Data Configuration of and Access to Multi-byte Data in Memory

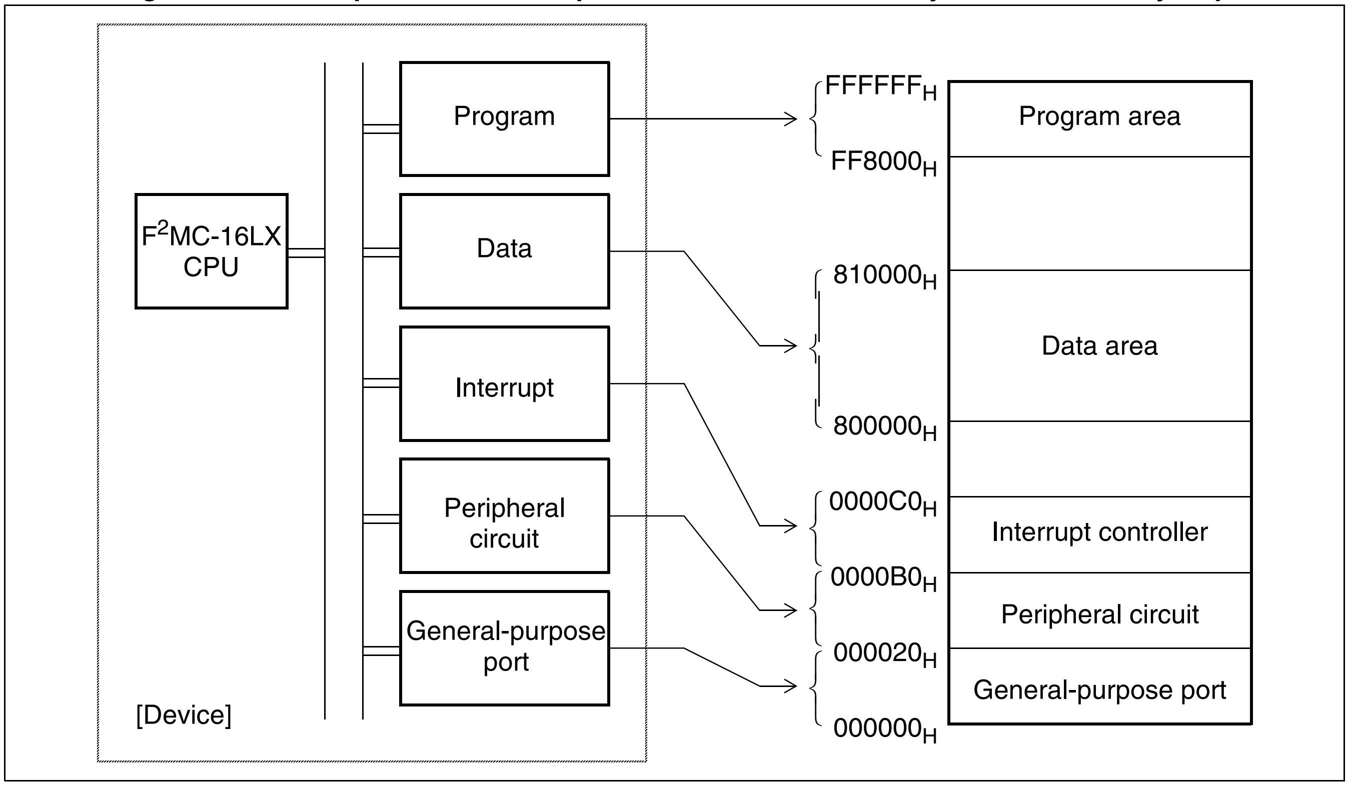
2.1 CPU Memory Space

All data, programs, and I/O areas managed in the F^{2}MC-16LX CPU are allocated in its 16-Mbyte memory space. The CPU can access these resources using an address on the 24-bit address bus (see Figure 2.1-1).

The F^{2}MC-16LX addressing mode can be classified either as a linear or bank mode. The linear mode specifies an entire 24-bit address using a instruction. The bank mode specifies the upper 8 bits of each address using a bank register, and the remaining 16-bit address using an instruction.

■ CPU Memory Space

Figure 2.1-1 Example of Relationship between the F^{2}MC-16LX System and Memory Map



2.2 Linear Addressing Mode

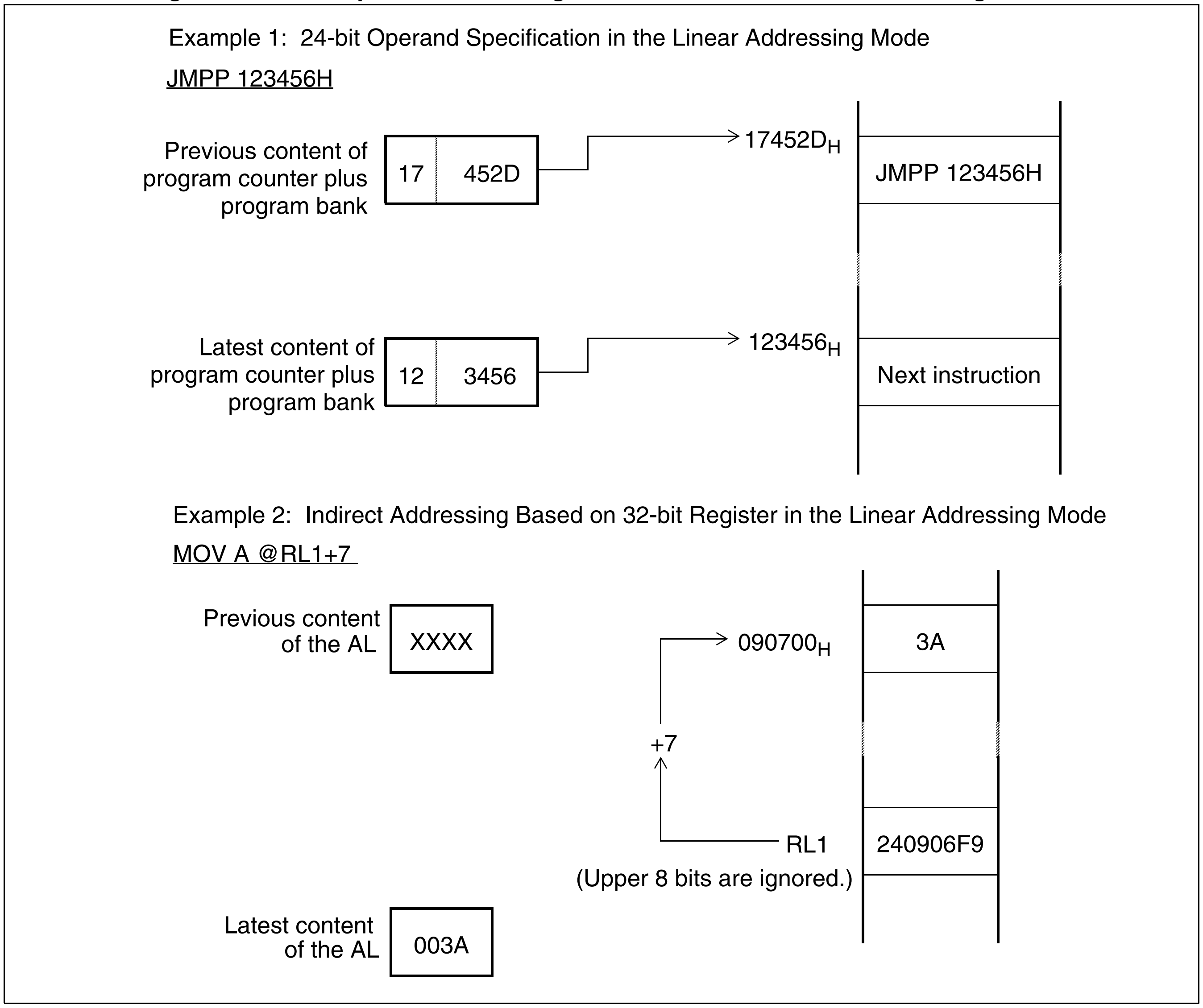
The linear addressing mode of the F^2MC-16LX specifies an entire 24-bit address using an instruction.
The linear addressing mode can operate in two different ways. In the first way, an operand of an instruction directly specifies an entire 24-bit address. In the second way, the lower 24-bit of a 32-bit general-purpose register is referred as an address.

■ Linear Addressing Mode

The linear addressing mode of the F^2MC-16LX specifies an entire 24-bit address using an instruction. The address mode of the F^2MC-16LX is determined according to the specification of the effective address or instruction code (implied) of an instruction.

The linear addressing mode can operate in two different ways. In the first way, an operand of an instruction directly specifies an entire 24-bit address. In the second way, the lower 24-bit of a 32-bit general-purpose register is referred as an address (see Figure 2.2-1).

Figure 2.2-1 Examples of Generating an Address in the Linear Addressing Mode


Example 1: 24-bit Operand Specification in the Linear Addressing Mode
JMPP 123456H
Previous content of program counter plus program bank
17
452D
17452D$_H$
JMPP 123456H
Latest content of program counter plus program bank
12
3456
123456$_H$
Next instruction
Example 2: Indirect Addressing Based on 32-bit Register in the Linear Addressing Mode
MOV A @RL1+7
Previous content of the AL
XXXX
090700$_H$
3A
+7
RL1
240906F9
(Upper 8 bits are ignored.)
Latest content of the AL
003A

2.3 Bank Addressing Mode

The bank addressing mode of the F^{2}MC-16LX specifies the upper 8 bits of an address using a bank register for use, and the remaining 16 bits using an instruction.

■ Bank Addressing Mode

In the bank addressing mode, the 16-Mbyte memory space is divided into 256 banks of 64-Kbyte, and the corresponding bank to each space is specified by the following 4 bank registers.

● Program bank register (PCB)

A 64-Kbyte bank specified using the PCB register is called a program (PC) space. It is used to hold mainly instruction codes, vector tables, and immediate data.

● Data bank register (DTB)

A 64-Kbyte bank specified using the DTB register is called a data (DT) space. It is used to hold mainly readable/writable data and control/data registers for internal and external resources.

● User stack bank register (USB) and system stack bank register (SSB)

A 64-Kbyte bank specified using the USB or SSB register is called a stack (SP) space. It is accessed when the execution of a push or pop instruction or interrupt handling is performed and which to be used, the USB or SSB register, is determined according to the S flag in the condition code register to save register contents and a stack access occurs.

● Additional data bank register (ADB)

A 64-Kbyte bank specified using the ADB register is called an additional (AD) space. It is used to hold mainly data overflowing from the DT space.

Each instruction is assigned with one of the default spaces by each addressing listed in Table 2.3-1 to improve instruction code efficiency.

Table 2.3-1 Default Spaces

Default space	Addressing
Program space	PC-indirect, program access, branch type
Data space	@A, addr16, dir, or addressing using @RW0, @RW1, @RW4, or @RW5
Stack space	Addressing using PUSHW, POPW, @RW3, @RW7, or @SP
Additional space	Addressing using @RW2 or @RW6

If a space other than a default space is used, an arbitrary bank space corresponding to a prefix code can be accessed by specifying the prefix code before the instruction.

Table 2.3-2 lists bank select prefixes and the memory space selected using each prefix.

Table 2.3-2 Bank Selection Prefix

Bank select prefix	Selected space
PCB	Program space
DTB	Data space
ADB	Additional space
SPB	System or user stack space depending on the contents of the selected stack flag

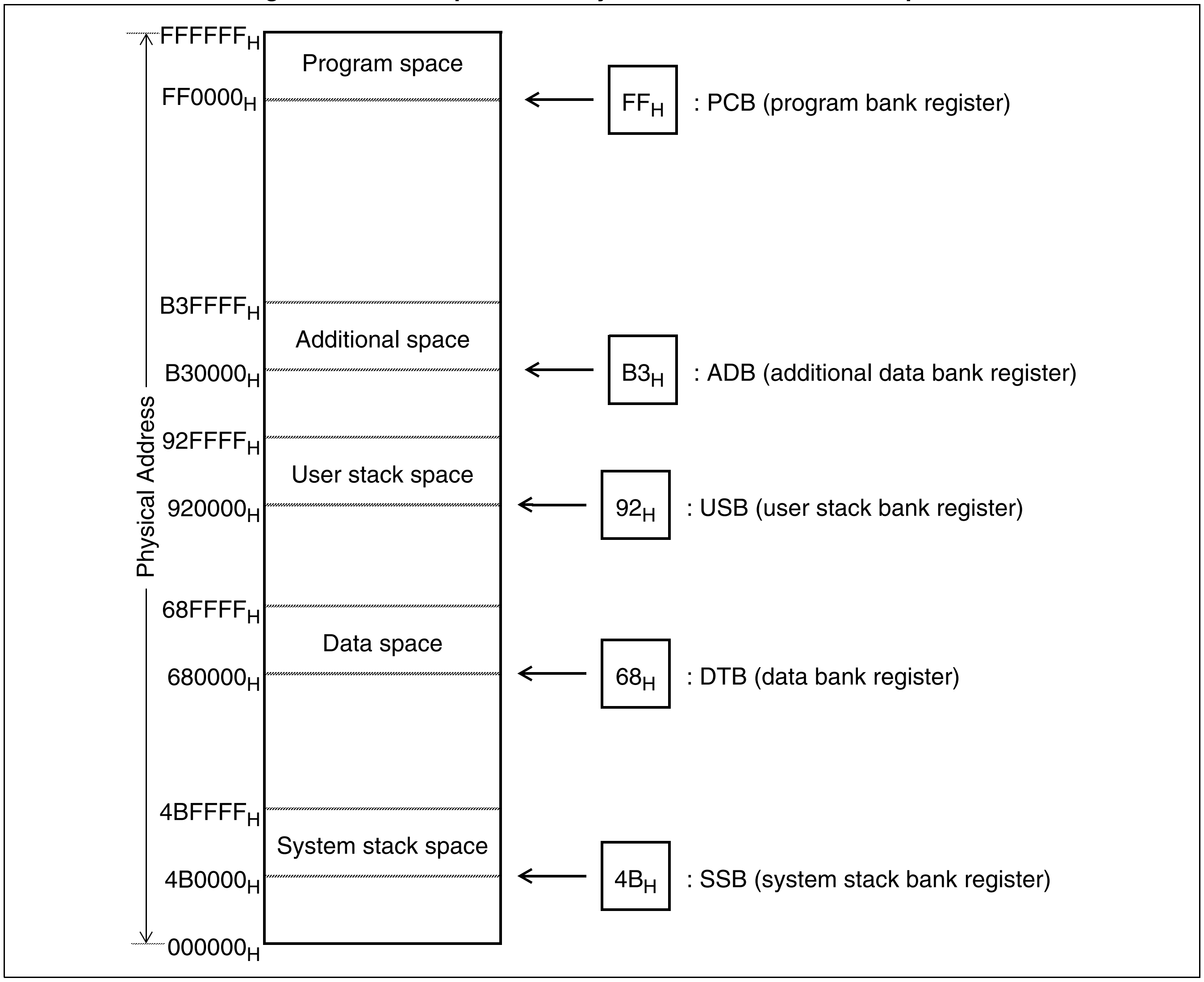
The DTB, USB, SSB, and ADB registers are initialized to "00_H" at a reset. The PCB register is initialized to "FF_H" at a reset. After a reset, the data, stack, and additional spaces are allocated in bank 00_H (000000_H to $00FFFF_H$), and the program space is allocated in bank FF_H ($FF0000_H$ to $FFFFFF_H$).

2.4 Memory Space Divided into Banks and Value in Each Bank Register

Figure 2.4-1 shows an example of a memory space divided into banks and a value in each register bank.

■ Memory Space Divided into Banks and Values in Each Register Bank

Figure 2.4-1 Example of the Physical Addresses of Each Space



2.5 Data Configuration of and Access to Multi-byte Data in Memory

Multi-byte data is written to memory starting at the lowest address. If the multi-byte data is 32-bit long, the lower 16 bits are written to memory first and then upper 16 bits.

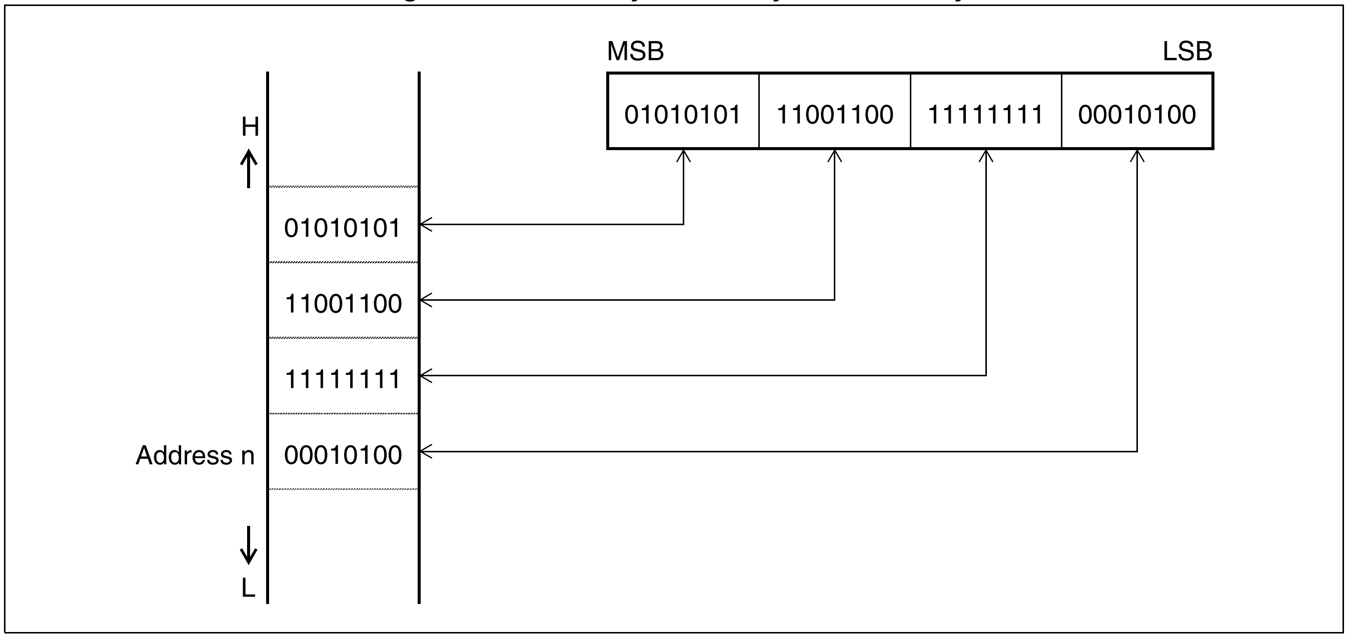
■ Multi-byte Data Layout in a Memory Space

Multi-byte data is written to memory starting at the lowest address. If the multi-byte data is 32-bit length, the lower 16 bits are written to memory first and then upper 16 bits.

If a reset signal is input immediately after the low-order data is written to memory, the high-order data may not be written. To keep the data in integrity, it is necessary to input a reset signal after the high-order data is written.

Figure 2.5-1 shows the layout of multi-byte data in memory. The lower 8 bits are placed at address n, the next lower 8 bits are placed at address n + 1, and the next lower 8 bits are placed at address n + 2, and so on.

Figure 2.5-1 Multi-byte Data Layout in Memory

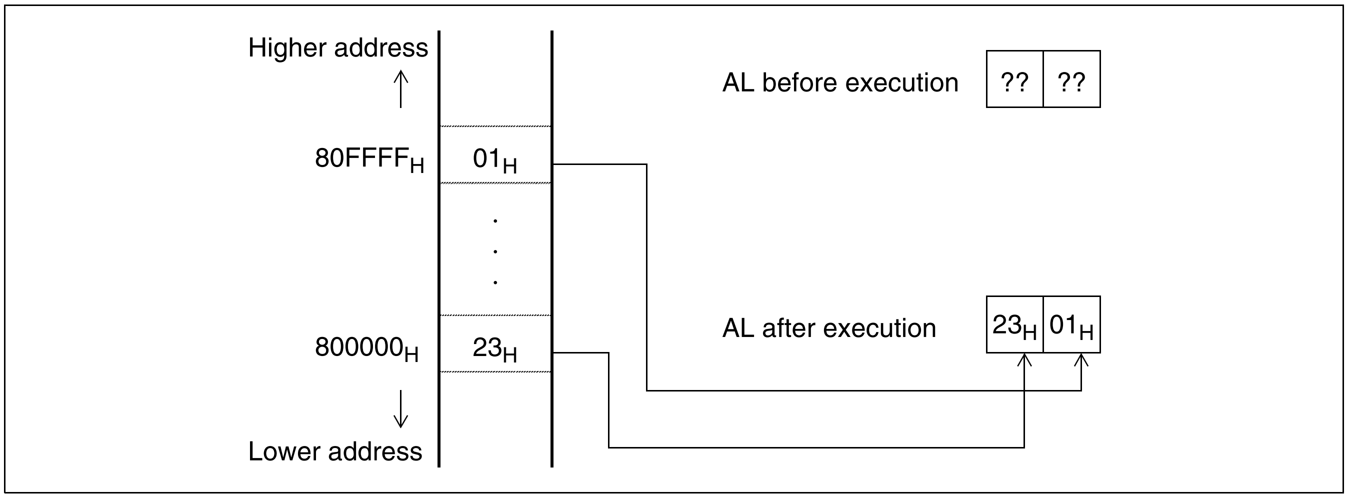


■ Access to Multi-byte Data

When multi-byte data is accessed, it is assumed that all parts of the multi-byte data are within a single bank. To put it another way, an instruction accessing multi-byte data assumes that an address that follows address $FFFF_H$ is 0000_H in the same bank as for $FFFF_H$.

Figure 2.5-2 shows an execution example of an instruction accessing multi-byte data.

Figure 2.5-2 Execution Example of an Instruction (MOVPW A, $080FFFF_H$) Accessing Multi-byte Data



CHAPTER 3
DEDICATED REGISTERS

The registers of the F^{2}MC-16LX can be grouped into two major categories: dedicated registers in the CPU and general-purpose registers allocated in memory.

This chapter describes the F^{2}MC-16LX dedicated registers. These registers are the dedicated hardware in the CPU. Their use is limited due to the architecture of the CPU.

3.1 F^{2}MC-16LX Dedicated Register Types

3.2 Accumulator (A)

3.3 User Stack Pointer (USP) and System Stack Pointer (SSP)

3.4 Processor Status (PS)

3.5 Program Counter (PC)

3.6 Direct Page Register (DPR)

3.7 Bank Registers

3.1 F^{2}MC-16LX Dedicated Register Types

There are 11 dedicated registers in the F^{2}MC-16LX.

- **Accumulator (A=AH:AL)**
- **System stack pointer (SSP)**
- **Program counter (PC)**
- **Data bank register (DTB)**
- **System stack bank register (SSB)**
- **Direct page register (DPR)**
- **User stack pointer (USP)**
- **Processor status (PS)**
- **Program bank register (PCB)**
- **User stack bank register (USB)**
- **Additional data bank register (ADB)**

■ F^{2}MC-16LX Dedicated Register Types

● Accumulator (A=AH:AL)

This is a set of two 16-bit accumulators. It can be used as a single 32-bit accumulator.

● User stack pointer (USP)

This is a 16-bit pointer indicating a user stack area.

● System stack pointer (SSP)

This is a 16-bit pointer indicating a system stack area.

● Processor status (PS)

This is a 16-bit register indicating the status of the system.

● Program counter (PC)

This is a 16-bit register to hold an address where the next instruction to be executed is stored.

● Program bank register (PCB)

This is an 8-bit register indicating the program space.

● Data bank register (DTB)

This is an 8-bit register indicating the data space.

● User stack bank register (USB)

This is an 8-bit register indicating the user stack space.

● System stack bank register (SSB)

This is an 8-bit register indicating the system stack space.

● Additional data bank register (ADB)

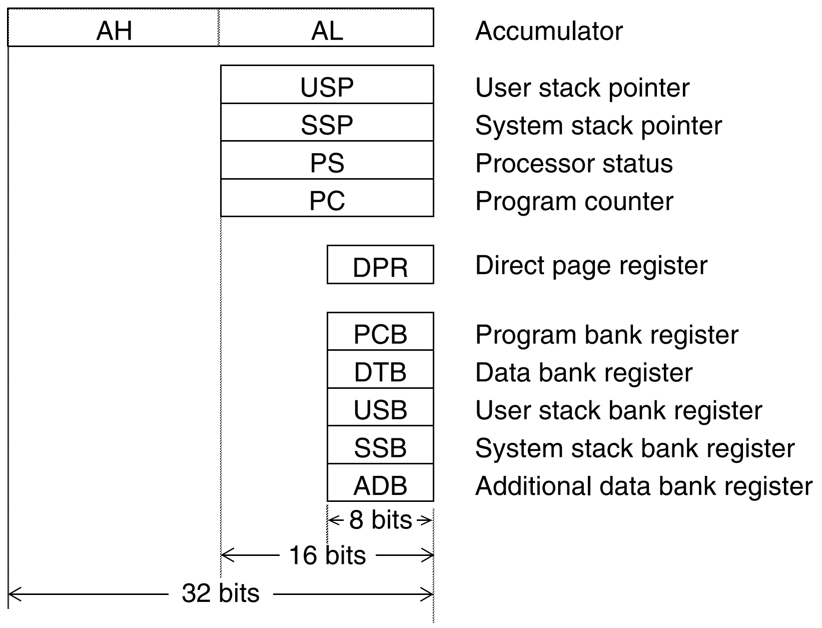
This is an 8-bit register indicating the additional space.

● Direct page register (DPR)

This is an 8-bit register indicating the direct page.

Figure 3.1-1 shows an image of the dedicated registers.

Figure 3.1-1 Dedicated Registers



3.2 Accumulator (A)

The accumulator (A) consists of two 16-bit length operation registers (AH and AL), is used for temporary storage of the results for an operation or of data to be transferred.

■ Accumulator (A)

The accumulator (A) consists of two 16-bit length operation registers (AH and AL), used for temporary storage of the results for an operation or of data to be transferred. To process 32-bit data, the AH and AL registers are concatenated (see Figure 3.2-1). To process 16-bit data (used in word-unit processing) or 8-bit data (used in byte-unit processing), only the AL register is used (see Figure 3.2-2). Various types of arithmetic and logical operations can be performed between data in the accumulator (A) and data in memory or a register (such as Ri, RWi, or RLi). Similarly to the F^{2}MC-8, the F^{2}MC-16LX automatically transfers data from the AL register to the AH register, if it receives new data at the AL register and the new data is not larger than a word (data preservation function). Use of this data preservation function and a function to perform arithmetic and logical operations between the AL and AH registers makes various types of processing more efficient (see Figure 3.2-2).

If data transferred to the AL register is not larger than a byte, the data is sign- or zero-extended to 16 bits and it is stored in the AL register. The data in the AL register can be handled as either a word or a byte. If a byte-unit arithmetic operation is performed on the AL register, the upper 8 bits of data that have been previously set in the AL register are ignored and reset to all "0"s.

Figure 3.2-1 Example of Transferring 32-bit Data

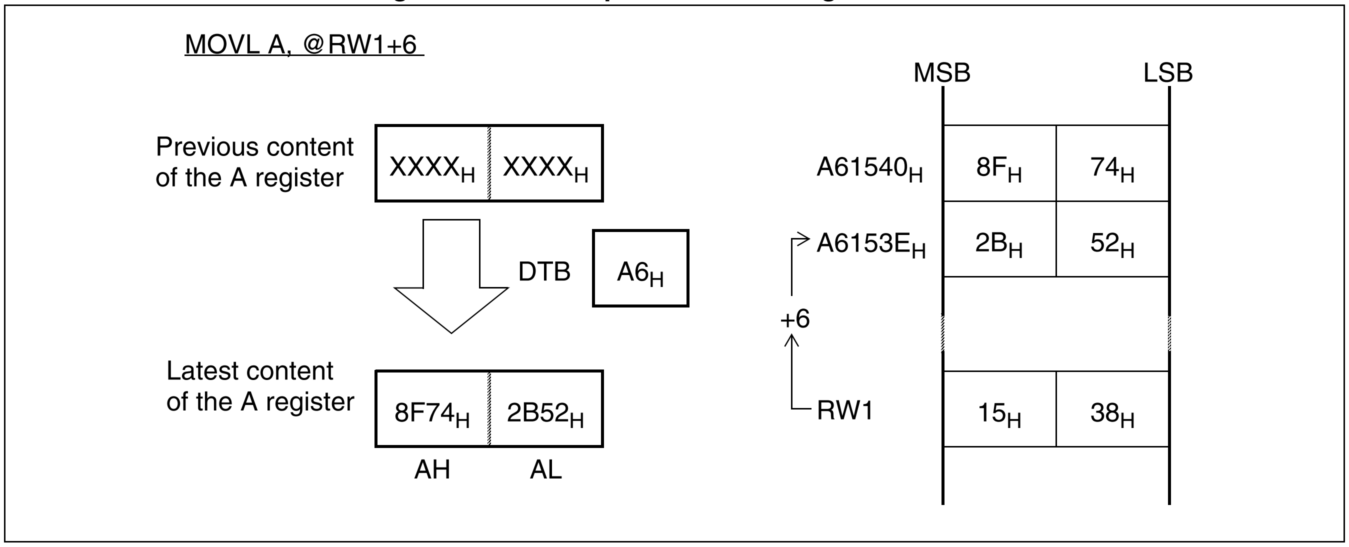
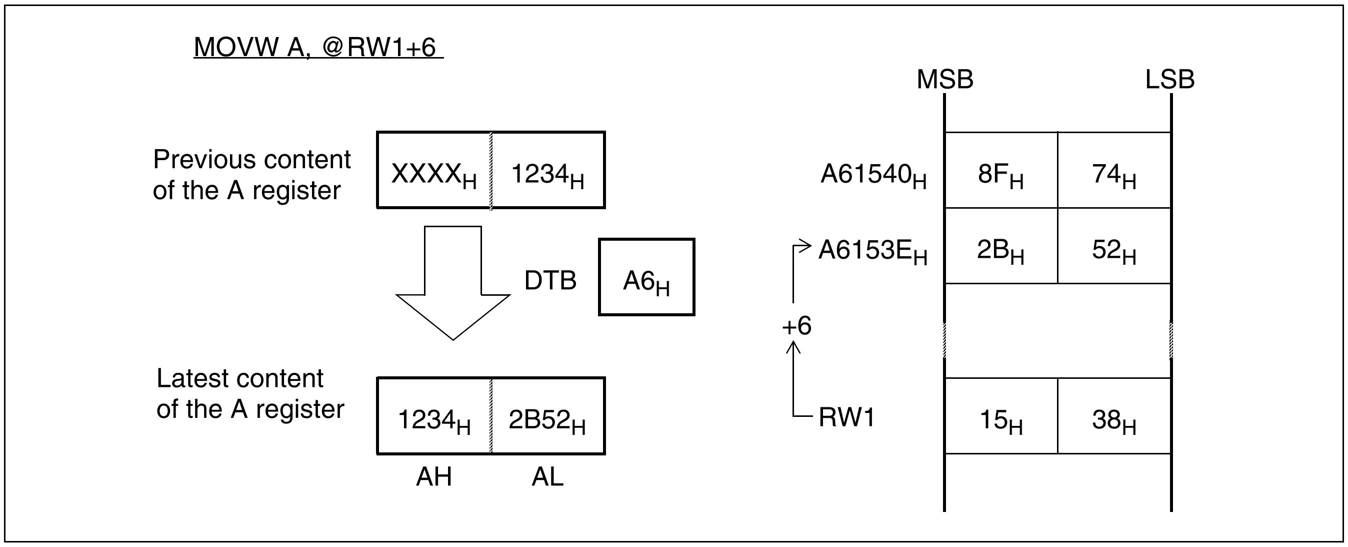


Figure 3.2-2 Example of Transferring Data between the AL and AH Registers by Means of the Data Preservation Function



3.3 User Stack Pointer (USP) and System Stack Pointer (SSP)

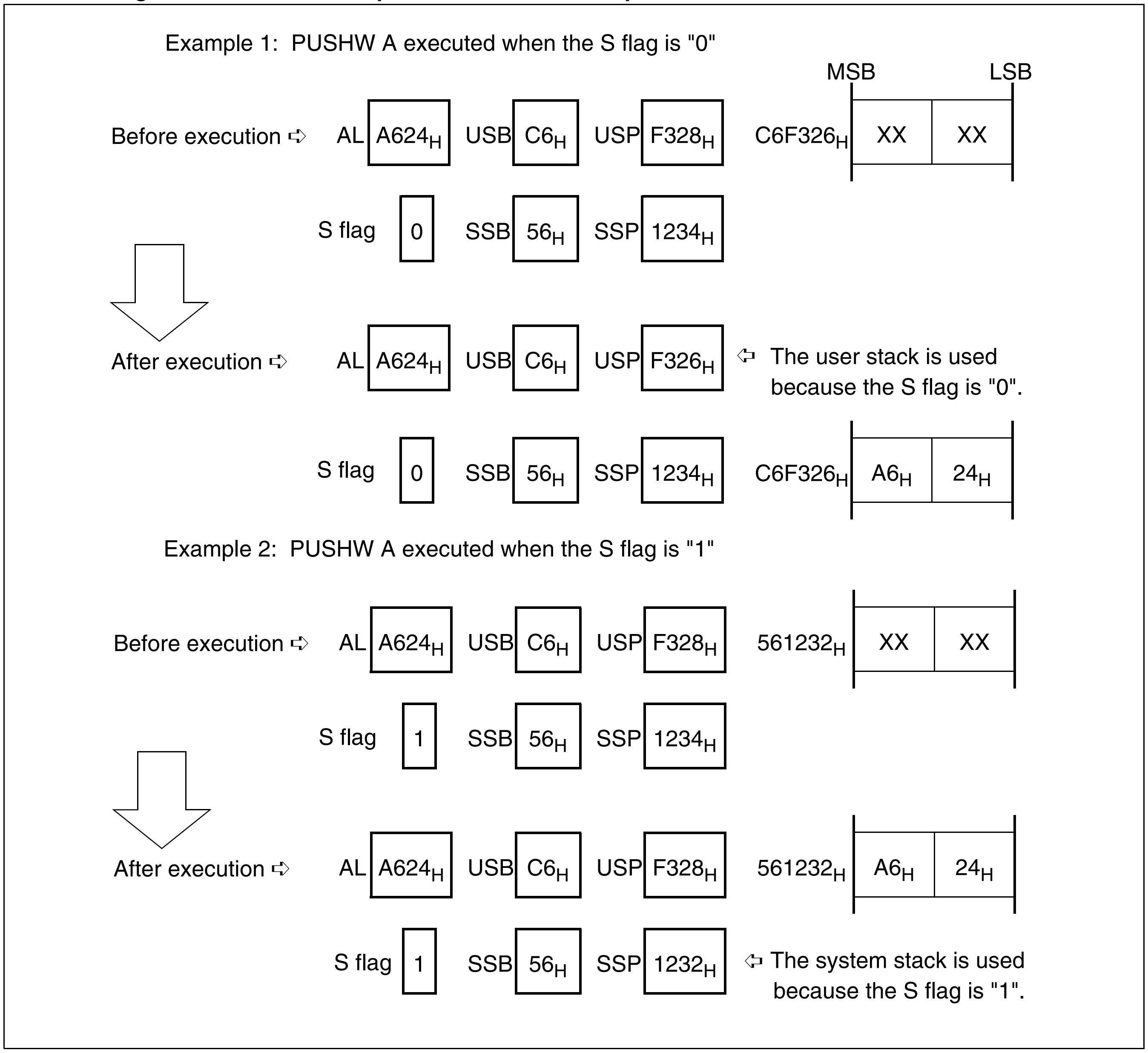
Both the user stack pointer (USP) and system stack pointer (SSP) are 16-bit registers. They are used to indicate a data save address or return address when a push, pop instruction, or subroutine is executed.
Basically, a value to be set in a stack pointer must be an even address.

■ User Stack Pointer (USP) and System Stack Pointer (SSP)

Both the user stack pointer (USP) and system stack pointer (SSP) are a 16-bit register. They are used to indicate a data save address or return address when a push, pop instruction, or subroutine is executed. The USP and SSP registers are used by stack manipulation instructions in the same manner. If the S flag in the condition code register (CCR) in the processor status (PS) register is "0", the USP register is active. If the S flag is "1", the SSP register is active (see Figure 3.3-1). Because the S flag becomes "1" when an interrupt is accepted, the SSP register is used to indicate a memory area to save register contents at an interrupt. The SSP register is used by an interrupt routine for stack manipulation, while the USP register is used by non-interrupt handling routines for stack manipulation. If it is unnecessary to divide the stack space, only the SSP register should be used.

For the SSP register, the upper 8 bits of an address used for stack manipulation are indicated by the system stack bank register (SSB). For the USP register, they are indicated by the user stack bank register (USB).

Figure 3.3-1 Relationships between Stack Manipulation Instruction and Stack Pointer



3.4 Processor Status (PS)

The processor status (PS) register consists of bits for controlling the CPU and those for indicating the status of the CPU. The PS register is divided into the following three registers.

- **Interrupt level mask register (ILM)**
- **Register bank pointer (RP)**
- **Condition code register (CCR)**

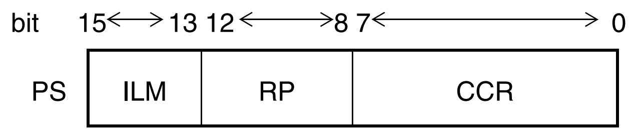
■ Processor Status (PS)

The processor status (PS) register consists of bits for controlling the CPU and those for indicating the status of the CPU.

- Interrupt level mask register (ILM): Indicates the level of an interrupt to be accepted.
- Register bank pointer (RP): Indicates the start address of a register bank.
- Condition code register (CCR): Consists of various flags that are set or reset during instruction execution or at an interrupt occurrence.

Figure 3.4-1 shows the structure of the processor status (PS) register.

Figure 3.4-1 Processor Status (PS) Register Structure



3.4.1 Interrupt Level Mask Register (ILM)

The following shows a configuration diagram of the interrupt level mask register (ILM).

ILM	**ILM2**	**ILM1**	**ILM0**
(Initial value)→	0	0	0

■ Interrupt Level Mask Register (ILM)

The interrupt level mask register (ILM) consists of 3 bits. It indicates the levels of interrupts acceptable to the CPU. If an interrupt request whose level is higher than the level indicated using these 3 bits, the interrupt is generated. Interrupt level 0 is the highest, and interrupt level 7 is the lowest (see Table 3.4-1). In other words, for an interrupt to be accepted, its interrupt level value must be smaller than the value held in the ILM register. When an interrupt is accepted, its interrupt level is set in the ILM register, thus prohibiting interrupts on lower levels from being accepted. Because the ILM register is initialized to all "0"s at a reset, the highest interrupt level is specified in the ILM register. It is possible to transfer 8-bit immediate data to the ILM register, but only the lower 3 bits of the data can be used.

Table 3.4-1 Interrupt Levels Indicated in the Interrupt Level Mask Register (ILM)

ILM2	ILM1	ILM0	Level value	Levels of acceptable interrupts
0	0	0	0	Interrupt disabled
0	0	1	1	0
0	1	0	2	1 and below
0	1	1	3	2 and below
1	0	0	4	3 and below
1	0	1	5	4 and below
1	1	0	6	5 and below
1	1	1	7	6 and below

3.4.2 Register Bank Pointer (RP)

The following shows a configuration diagram of the register bank pointer (RP).

RP	**B4**	**B3**	**B2**	**B1**	**B0**
(Initial value)$\rightarrow$	0	0	0	0	0

■ Register Bank Pointer (RP)

The register bank pointer (RP) indicates the address of an internal RAM area where the general-purpose registers of the F^2MC-16LX are. The start memory address of a register bank currently in use is represented using the following conversion expression: [000180_H + RP $\times$ 10_H]. The RP register consists of 5 bits. It can hold any value between "00_H" and "$1F_H$". So the start memory address of the register bank can be set in the range between 000180_H and $00037F_H$. Therefore, the register bank can be allocated at memory locations in the range between 000180_H and $00037F_H$. If the internal RAM area used as an external area, however, it cannot be used as general-purpose registers even if the register bank is within that range. It is possible to transfer 8-bit immediate data to the RP register, but only the lower 5 bits of the data can be used.

3.4.3 Condition Code Register (CCR)

The following shows a configuration diagram of the condition code register (CCR).

bit	7	6	5	4	3	2	1	0
CCR	–	I	S	T	N	Z	V	C
(Initial value)→		0	1	–	–	–	–	–

-: Undefined

■ Condition Code Register (CCR)

- I (Interrupt enable flag): If the I flag is "1", all non-software interrupts are acceptable. If the flag is "0", they are disabled. The flag is cleared by a reset.
- S (Stack flag): If the S flag is "0", the user stack pointer (USP) is active as a pointer for stack manipulation. If the flag is "1", the system stack pointer (SSP) is active. The flag is set at a reset and when an interrupt is accepted.
- T (Sticky bit flag): If at least one bit read from the C flag is "1" when a logical shift right instruction or an arithmetic shift right instruction is executed, this flag becomes "1". Otherwise, the flag becomes "0". The flag becomes "0" also if the amount of shifting is "0".
- N (Negative flag): If the most significant bit of an arithmetic or logical operation result is "1", this flag is set. If it is "0", this flag is cleared.
- Z (Zero flag): If the result of an arithmetic or logical operation is all "0"s, this flag is set. Otherwise, it is cleared.
- V (Overflow flag): This flag is set if a digit overflow occurs in a signed value generated as the result of an arithmetic or logical operation. Otherwise, the flag is cleared.
- C (Carry flag): This flag is set if an arithmetic or logical operation causes a carry from or a borrow to the most-significant bit. Otherwise, the flag is cleared.

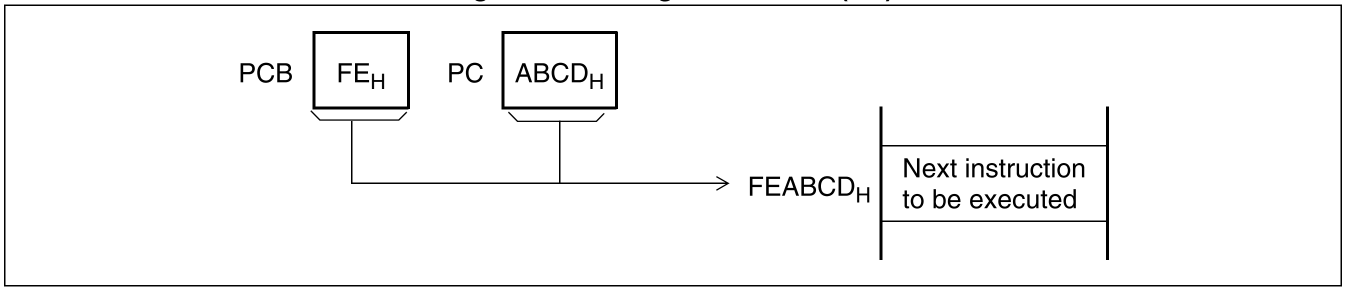
3.5 Program Counter (PC)

The program counter (PC) consists of 16 bits. It indicates the upper 16 bits of a memory address where the next instruction to be executed by the CPU.

■ Program Counter (PC)

The program counter (PC) consists of 16 bits. It indicates the lower 16 bits of a memory address where the next instruction to be executed by the CPU is (see Figure 3.5-1). The upper 8 bits of the memory address are indicated in the program bank register (PCB). The content of the PC register is updated, when a conditional branch or subroutine call instruction is executed, upon an interrupt occurrence, or at a reset. The PC register is used also as a base pointer for reading an operand.

Figure 3.5-1 Program Counter (PC)



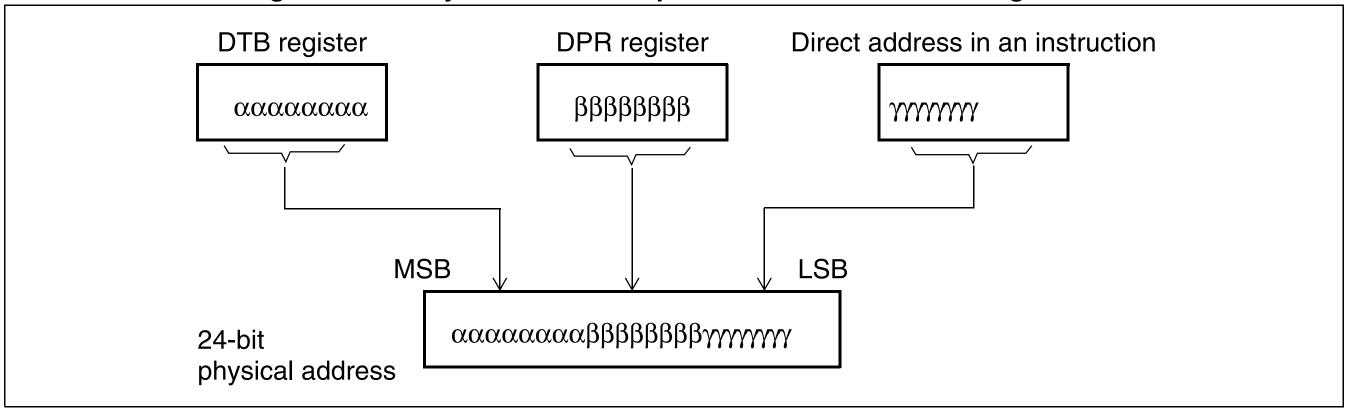
3.6 Direct Page Register (DPR)

The direct page register (DPR) specifies bits 8 to 15 (addr8 to addr15) of an operand address for an instruction in direct addressing mode.

■ Direct Page Register (DPR)

The direct page register (DPR) specifies bits 8 to 15 (addr8 to addr15) of an operand address for an instruction in direct addressing mode, as shown in Figure 3.6-1 . The DPR register is 8 bits long. It is initialized to "01_H" at a reset. It can be read- and write-accessed by an instruction.

Figure 3.6-1 Physical Address Specified in Direct Addressing Mode



3.7 Bank Registers

The following 5 bank registers are available in the F^2MC-16LX.

- **Program bank register (PCB)**
- **Data bank register (DTB)**
- **User stack bank register (USB)**
- **System stack bank register (SSB)**
- **Additional data bank register (ADB)**

These registers indicate memory banks allocated for the program, data, user stack, system stack, and additional spaces, respectively.

■ Bank Registers

All of these bank registers are 1 byte length. At a reset, the PCB register is initialized to "$0FF_H$", and the other registers, to "00_H". The PCB register can be read-accessed, but not write-accessed. The other bank registers can be both read- and write-accessed. The content of the PCB register is updated, when a JMPP, CALLP, RETP, or RETI instruction for a branch anywhere in the 16-Mbyte space is executed or an interrupt occurs. See "CHAPTER 2 MEMORY SPACE" for descriptions about the operation of each register.

CHAPTER 4 GENERAL-PURPOSE REGISTERS

The registers of the F^{2}MC-16LX can be grouped into two major categories: dedicated registers in the CPU and general-purpose registers allocated in memory.

This chapter describes the F^{2}MC-16LX general-purpose registers. These registers are allocated in a RAM in address space of the CPU. Similarly to the dedicated registers, the general-purpose registers can be accessed without specifying their address. However, the user can specify the purpose for which they are used in the same manner as for ordinary memory.

4.1 Register Banks in RAM

4.2 Calling General-purpose Registers in RAM

4.1 Register Banks in RAM

Each register bank consists of 8 words (16 bytes). They can be used as general-purpose registers (byte registers R0 to R7, word registers RW0 to RW7, and long word registers RL0 to RL3) for performing various types of operations and specifying pointers. RL0 to RL3 can be used also as a linear pointer to gain direct access to all spaces in memory.

■ Register Banks in RAM

Table 4.1-1 lists the function of each register, and Table 4.1-2 shows relationships between the registers.

Table 4.1-1 Functions of Each Register

Register name	Function
R0 to R7	Used to hold an operand in various types of instructions. Note: R0 is also used as a barrel shift counter and a counter of normarize instruction.
RW0 to RW7	Used to hold a pointer. Used to hold an operand in various types of instructions. Note: RW0 is used also as a string instruction counter.
RL0 to RL3	Used to hold a long pointer. Used to hold an operand in various types of instructions.

Table 4.1-2 Relationship between Registers

	RW0	RL0
	RW1	
	RW2	RL1
	RW3	
R0	RW4	RL2
R1		
R2	RW5	
R3		
R4	RW6	RL3
R5		
R6	RW7	
R7		

4.2 Calling General-purpose Registers in RAM

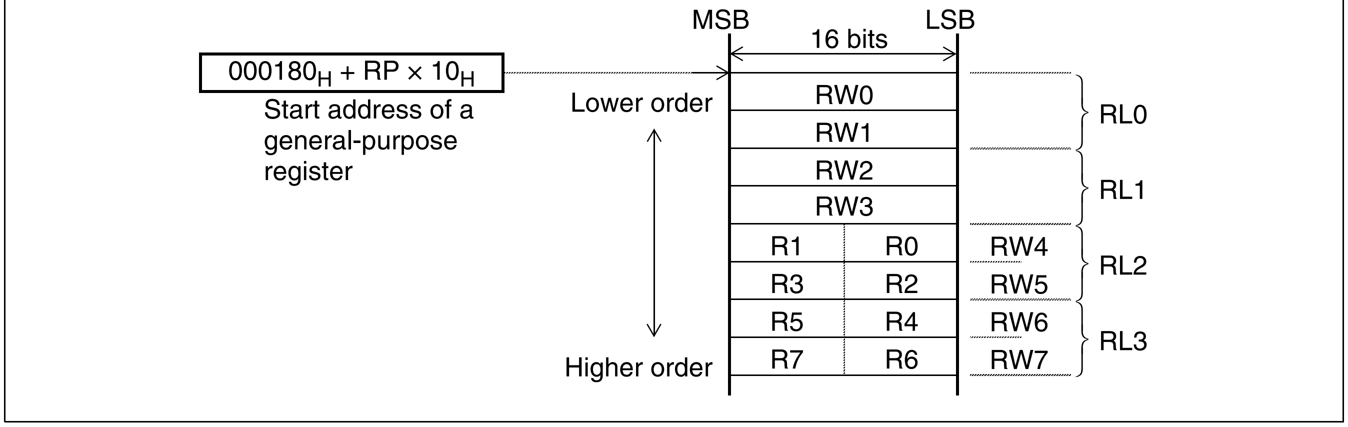
For general-purpose registers of the F^2MC-16LX, the register bank pointer (RP) is used to specify where in internal RAM between 000180_H and $00037F_H$ the register bank currently in use is allocated.

■ Calling General-purpose Registers in RAM

The general-purpose registers of the F^2MC-16LX are allocated in internal RAM between 000180_H and $00037F_H$ (in maximum configuration). The register bank pointer (RP) is used to indicate where in internal RAM between 000180_H and $00037F_H$ the register bank currently in use is allocated. Each bank contains the following 3 different registers. These registers are not independent of one another. Instead, they have the relationships shown in Figure 4.2-1 .

- R0 to R7: 8-bit general-purpose registers
- RW0 to RW7: 16-bit general-purpose registers
- RL0 to RL3: 32-bit general-purpose registers

Figure 4.2-1 General-purpose Registers



The relationships among the high- and low-order bytes in word registers (RW4 to RW7) and byte registers (R0 to R7) are represented using the following expression:

RW (i + 4) = R (i × 2 + 1) × 256 + R (i × 2) [where i = 0 to 3]

The relationships among the high- and low-order bytes in long registers (RL0 to RL3) and word registers (RW0 to RW7) are represented using the following expression:

RL (i) = RW (i × 2 + 1) × 65536 + RW (i × 2) [where i = 0 to 3]

For example, if the data in R1 and the data in R0 are arranged as high- and low-order bytes, respectively, the resulting data equals the data (2 bytes) in RW4.

CHAPTER 5
PREFIX CODES

The operation of an instruction can be modified by prefixing it with prefix code. The following 3 types of prefix codes are available.

- **Bank select prefix**
- **Common register bank prefix**
- **Flag change inhibit prefix code**

This chapter describes these prefixes.

5.1 Bank Select Prefix

5.2 Common Register Bank Prefix (CMR)

5.3 Flag Change Inhibit Prefix Code (NCC)

5.4 Constraints Related to the Prefix Codes

5.1 Bank Select Prefix

Placing a bank select prefix before an instruction enables selecting the memory space accessed by the instruction regardless of what the current addressing mode is.

■ Bank Select Prefix

The memory space of data to be accessed is determined according to the addressing mode. Placing a bank select prefix before an instruction enables to select the memory space accessed by the instruction regardless of what the current addressing mode is. Table 5.1-1 lists the bank select prefixes and the memory space selected according to each bank select prefix.

Table 5.1-1 Bank Select Prefixes

Bank select prefix	Memory space to be selected
PCB	Program counter space
DTB	Data space
ADB	Additional space
SPB	System or user stack space depending on the state of the stack flag

Be careful when using the following instructions.

● Transfer instructions (I/O access)

MOV A,io	MOV io, A	MOVX A,io	MOVW A,io
MOVW io,A	MOV io,#imm8	MOVW io,#imm16	

These instructions access the I/O space regardless of whether there is a prefix before them.

● Branch instruction

RETI

The system stack bank (SSB) is used regardless of whether there is a prefix before the branch instruction.

● Bit manipulation instructions (I/O access)

MOVB A,io:bp	MOVB io:bp,A	SETB io:bp
CLRB io:bp	BBC io:bp,rel	BBS io:bp,rel
WBTC	WBTS	

The I/O space is accessed regardless of whether there is a prefix before those instructions.

● String manipulation instructions

MOVS MOVSW SCEQ SCWEQ FILS FILSW

A bank register specified in the operand is used regardless of whether there is a prefix before these instructions.

● Other types of control instructions (stack manipulation)

PUSHW POPW

The system stack bank (SSB) or user stack bank (USB) is used depending on the state of the S flag, regardless of whether there is a prefix before these instructions.

POPW PS

In the following cases, the prefix of an instruction affects not only that instruction but also an instruction that follows it.

● Other types of control instructions (flag change)

AND CCR,#imm8 OR CCR,#imm8

The operations of these instructions are performed normally. The prefix of each of these instructions affects not only the instructions but also an instruction that follows them.

● Another type of control instruction (interrupt control)

MOV ILM,#imm8

The operation of the instruction is performed normally. The prefix of the instruction affects not only that instruction but also an instruction that follows it.

5.2 Common Register Bank Prefix (CMR)

Placing a common register bank prefix (CMR) before an instruction accessing a register bank enables to change that the instruction is to access only the registers in a common bank (register bank selected when RP = 0) allocated between 000180_H and $00018F_H$, regardless of what the current value of the register bank pointer (RP) is.

■ Common Register Bank Prefix (CMR)

To make data exchange among tasks easier, it is necessary to use a method that can access a certain specified register bank relatively easily no matter what value the RP register holds. To meet this requirement, the F^2MC-16LX has a register bank that can be used by all tasks in common. It is called a common bank. The common bank is allocated in memory between address 000180_H and $00018F_H$. It is selected when the RP register contains a value of "0".

Placing the common register bank prefix (CMR) before an instruction accessing a register bank enables to change that the instruction is to access only the registers in a common bank (register bank selected when RP = 0) allocated between 000180_H and $00018F_H$, regardless of what the current value of the register bank pointer (RP) is.

Be careful when using the following instructions.

● String instructions

MOVS NOVSW SCEQ FILS FILSW

If an interrupt is requested during execution of a string manipulation instruction attached with a prefix code, the prefix becomes ineffective for the string manipulation instruction after a return is made from the interrupt handling routine, possibly resulting in a malfunction. Do not place the CMR prefix before these string manipulation instructions.

● Other types of control instructions (flag change)

AND CCR,#imm8 OR CCR,#imm8 POPW PS

The operations of these instructions are performed normally. The prefix of each of these instructions affects not only the instructions but also an instruction that follows them.

● MOV ILM,#imm8

The operation of the instruction is performed normally. The prefix of the instruction affects not only that instruction but also an instruction that follows it.

5.3 Flag Change Inhibit Prefix Code (NCC)

Placing the flag change inhibit prefix code (NCC) before an instruction inhibits flags from changing during execution of the instruction.

■ Flag Change Inhibit Prefix Code (NCC)

The flag change inhibit prefix code (NCC) is used to suppress undesired changes to flags. Placing the NCC prefix before an instruction inhibits flags from changing during execution of the instruction.

Be careful when using the following instructions.

● Branch instructions

INT #vct8	INT9	INT addr16
INTP addr24	RETI	

These instructions change the flags in the condition code register (CCR) regardless of whether there is a prefix before them.

● String instructions

MOVE MOVSW SCEQ SCWEQ FILS FISW

If an interrupt is requested during execution of a string manipulation instruction attached with a prefix code, the prefix becomes ineffective for the string manipulation instruction after a return is made from the interrupt handling routine, possibly resulting in a malfunction. Do not place the NCC prefix before these string manipulation instructions.

● Another type of control instruction (task switching)

JCTX @A

This instruction changes the flags in the CCR register regardless of whether there is a prefix before it.

● Other types of control instructions (flag change)

AND CCR,#imm8 OR CCR,#imm8 POPW PS

These instructions change the flags in the CCR register regardless of whether there is a prefix before them. The prefix of each of these instructions affects not only the instructions but also an instruction that follows them.

● Another type of control instruction (interrupt control)

MOV ILM,#imm8

The operation of the instruction is performed normally. The prefix of the instruction affects not only that instruction but also an instruction that follows it.

5.4 Constraints Related to the Prefix Codes

If a prefix code is placed before an instruction where interrupt and hold requests are inhibited, the effect of the prefix code lasts until an instruction where neither an interrupt nor hold request is inhibited appears for the first time, as shown in Figure 5.4-2 .
If a prefix is followed by conflicting prefix codes, the last one is valid.

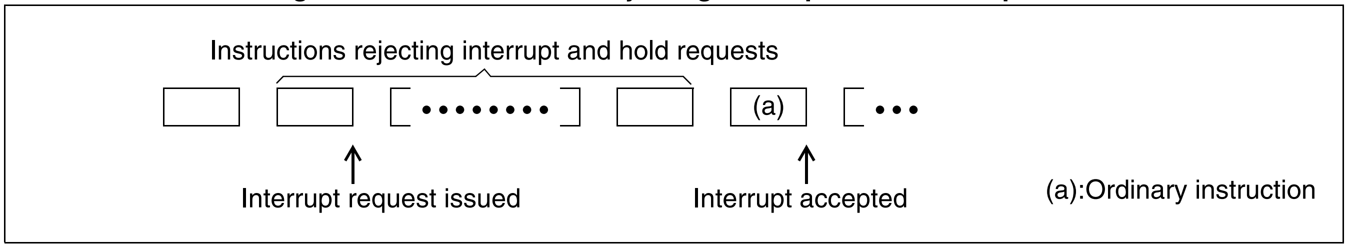
■ Relationships between Instructions Rejecting Interrupt Requests and Prefix Codes

The following 10 instructions/prefix codes reject interrupt and hold requests.

- MOV ILM,#imm8
- OR CCR,#imm8
- PCB
- NCC
- SPB
- AND CCR,#imm8
- POPW PS
- ADB
- DTB
- CMR

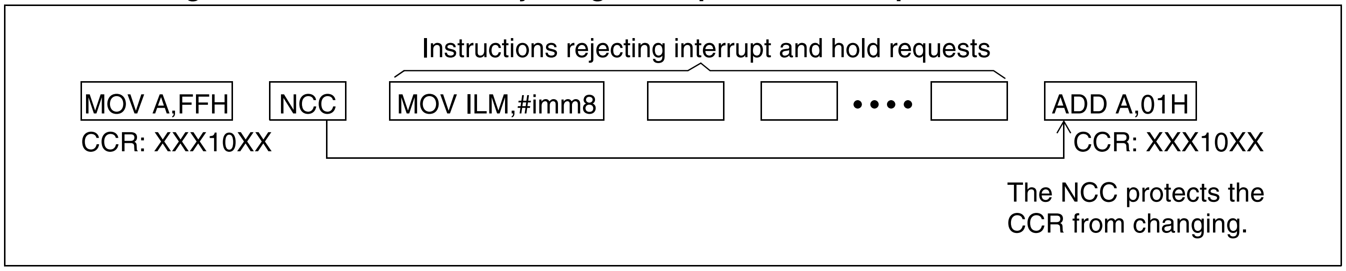
If an interrupt or hold request is issued during execution of any of the above instructions, the request is accepted only after any instruction not listed above appears for the first time after that instruction and is executed, as shown in Figure 5.4-1 .

Figure 5.4-1 Instructions Rejecting Interrupt and Hold Requests



If a prefix code is placed before an instruction rejecting interrupt and hold requests, its effect lasts until an instruction other than instructions rejecting interrupt and hold requests appears for the first time after the prefix code and is executed, as shown in Figure 5.4-2 .

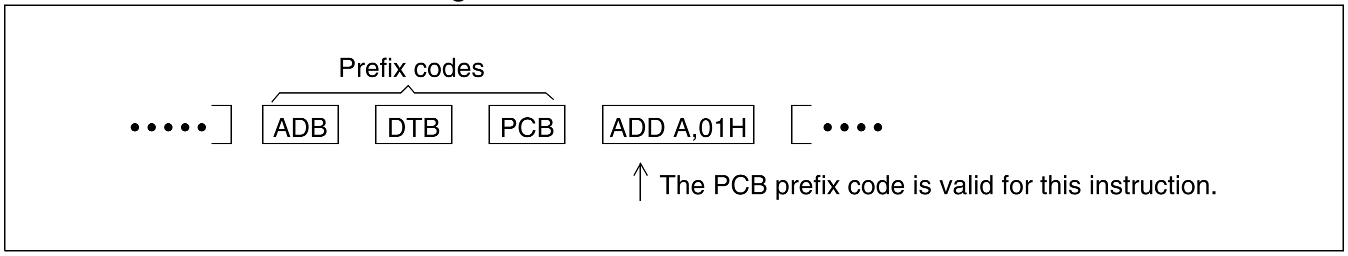
Figure 5.4-2 Instructions Rejecting Interrupt and Hold Requests and Prefix Code


Instructions rejecting interrupt and hold requests
MOV A,FFH
NCC
MOV ILM,#imm8
ADD A,01H
CCR: XXX10XX
CCR: XXX10XX
The NCC protects the CCR from changing.

■ If Two or More Prefix Codes Appear in Succession

If a prefix is followed by conflicting prefix codes, the last one is valid (see Figure 5.4-3).

Figure 5.4-3 Consecutive Prefix Codes



The term "conflicting prefix codes" indicates PCB, ADB, DTB, and SPB in the above figure.

CHAPTER 6

INTERRUPT HANDLING

This chapter describes the interrupt function and operation of F^{2}MC-16LX.

6.1 Interrupt Handling
6.2 Hardware Interrupt Operation Flow
6.3 Interrupt Handling Flowchart and Saving the Contents of Registers
6.4 Interrupt Vectors
6.5 Extended Intelligent I/O Service
6.6 Interrupt Control Register (ICR)
6.7 Meanings of the Bits of Interrupt Control Register (ICR)
6.8 Extended Intelligent I/O Service Descriptor (ISD)
6.9 Registers of Extended Intelligent I/O Service Descriptor
6.10 Exception Processing

6.1 Interrupt Handling

In F^2MC-16LX series, interrupt handling or extended intelligent I/O service is activated by the interrupt request from an internal resource. For interrupt handling, the processing appropriate to the interrupt request is performed by the interrupt handling program. For extended intelligent I/O service, the data transfer between the requesting internal resource and the memory is automatically performed. In addition, a function is provided to stop the execution of the extended intelligent I/O service by the request from the internal resource (such as built-in peripheral circuit).

■ Interrupt Handling

To permit an internal resource to make a hardware interrupt request to the F^2MC-16LX CPU, an interrupt request flag and an interrupt enable flag are required for that resource. The interrupt request flag is set by the occurrence of an event specific to the internal resource. When the interrupt request flag indicates the request being made and the interrupt enable flag is set to the enabled state, a hardware interrupt request is issued from the internal resource.

In the case of the internal resource that requires the activation of the extended intelligent I/O service accompanied by the occurrence of a hardware interrupt request, an extended intelligent I/O service enable (ISE) flag is provided in the interrupt control register (ICR) in the interrupt controller associated with that resource.

The occurrence of an interrupt request with the ISE flag set to "1" activates the extended intelligent I/O service. If only normal hardware interrupt requests are intended, set the ISE flag to "0".

For interrupt requests by the execution of the INT instruction, which are software interrupts, no interrupt request and enable flags are applied. Whenever the INT instruction is executed, an interrupt request occurs.

Any interrupt level of hardware interrupt request can be assigned to a given group regarding interrupt request. Interrupt levels are specified by the interrupt level setting bits (IL0, IL1, and IL2) in the ICR register in the interrupt controller. It is possible to specify eight interrupt level settings 0 to 7. Definition of the interrupt levels is such that "0" is the highest and "6" is the lowest.

From a group set to interrupt level 7, no interrupt requests can be made. Hardware interrupt requests are maskable (enabled/disabled) by the I flag in the condition code register (CCR) of the processor status (PS) and the ILM register (ILM0, ILM1, and ILM2).

When an unmasked interrupt request occurs, the CPU takes the following actions:

(1) Saves the data (12 bytes) held by the following registers into the memory area indicated by the system stack bank register (SSB) and the system stack pointer (SSP).

- Processor status (PS)
- Program counter (PC)
- Program bank register (PCB)
- Data bank register (DTB)
- Additional data bank register (ADB)
- Direct page register (DPR)
- Accumulator (A)

(2) Reads the interrupt vector in 3 bytes to PC and PCB.

(3) Updates the ILM register in the PS to the level setting value of the accepted interrupt request and sets the S flag in the CCR register.

(4) Initiates the instruction execution, starting with the address indicated by the interrupt vector.

For the INT instruction, the ILM register is not updated and the I flag in the CCR register is cleared. Subsequent interrupt requests are put to the pending state.

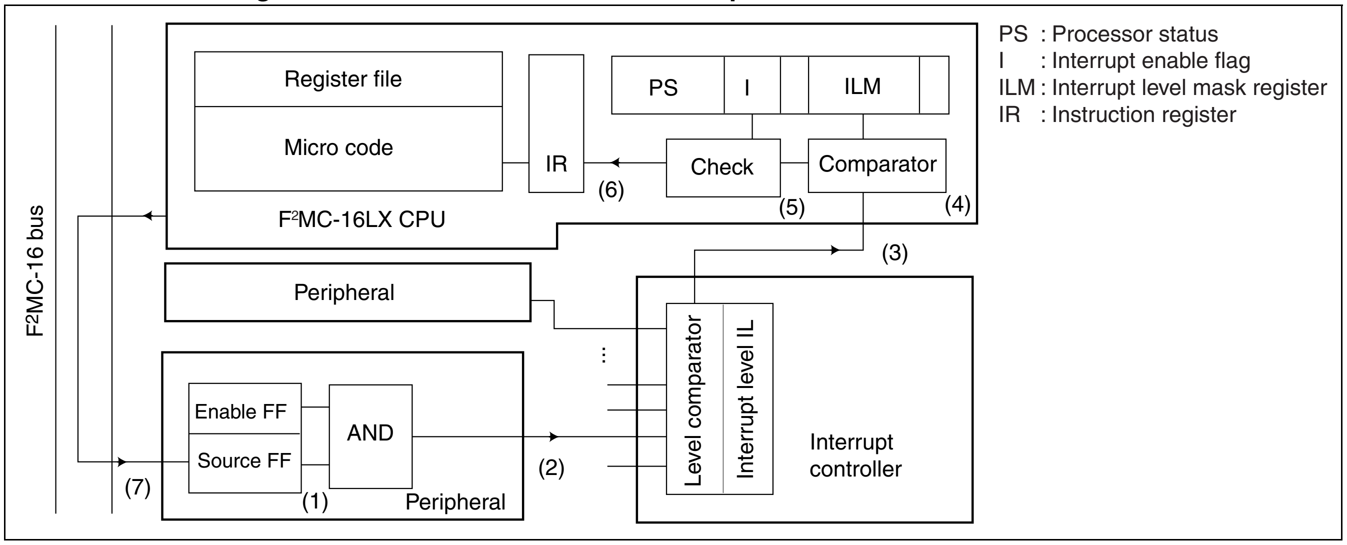
As a special case, hardware interrupt requests cannot be accepted during writing into an I/O area. This is intended to avoid the CPU malfunction, which might otherwise be caused by the occurrence of an interrupt request while the related data in the interrupt control registers for the resources are being rewritten.

6.2 Hardware Interrupt Operation Flow

Figure 6.2-1 shows the operation flow from the occurrence of a hardware interrupt request until the interrupt request has been cleared and removed from within the interrupt handling program.

■ Hardware Interrupt Operation Flow

Figure 6.2-1 From the Hardware Interrupt Occurrence to its Clearance



(1) An interrupt source occurs within the peripheral.

(2) If the interrupt enable bit within the peripheral is referred and it indicates the interrupt enabled state, an interrupt request is issued from the peripheral to the interrupt controller.

(3) The interrupt controller that has received that interrupt request determines the priority between the requests made at the same time and transfers the interrupt level corresponding to the appropriate interrupt to the CPU.

(4) The CPU compares the interrupt level requested by the interrupt controller with the IL bit held in the processor status register.

(5) Only if the result of this comparison is that the interrupt level priority is higher than the current interrupt handling level, the CPU checks the content of the I flag in the same processor status register.

(6) Only if the result of the check in (5) is that the I flag is set in the interrupt enabled state, the CPU sets the content of the IL bit to the requested level. Upon the completion of the ongoing instruction execution, the CPU passes the control to the interrupt handling routine to initiate the handling of that interrupt.

(7) When the software within the user's interrupt handling routine clears the occurred interrupt cause as mentioned in (1), this interrupt request process is terminated.

6.3 Interrupt Handling Flowchart and Saving the Contents of Registers

Figure 6.3-1 shows the interrupt handling flowchart and Figure 6.3-2 shows how the contents of the registers are saved with interrupt handling.

■ Interrupt Handling Flowchart

Figure 6.3-1 Interrupt Handling Flowchart

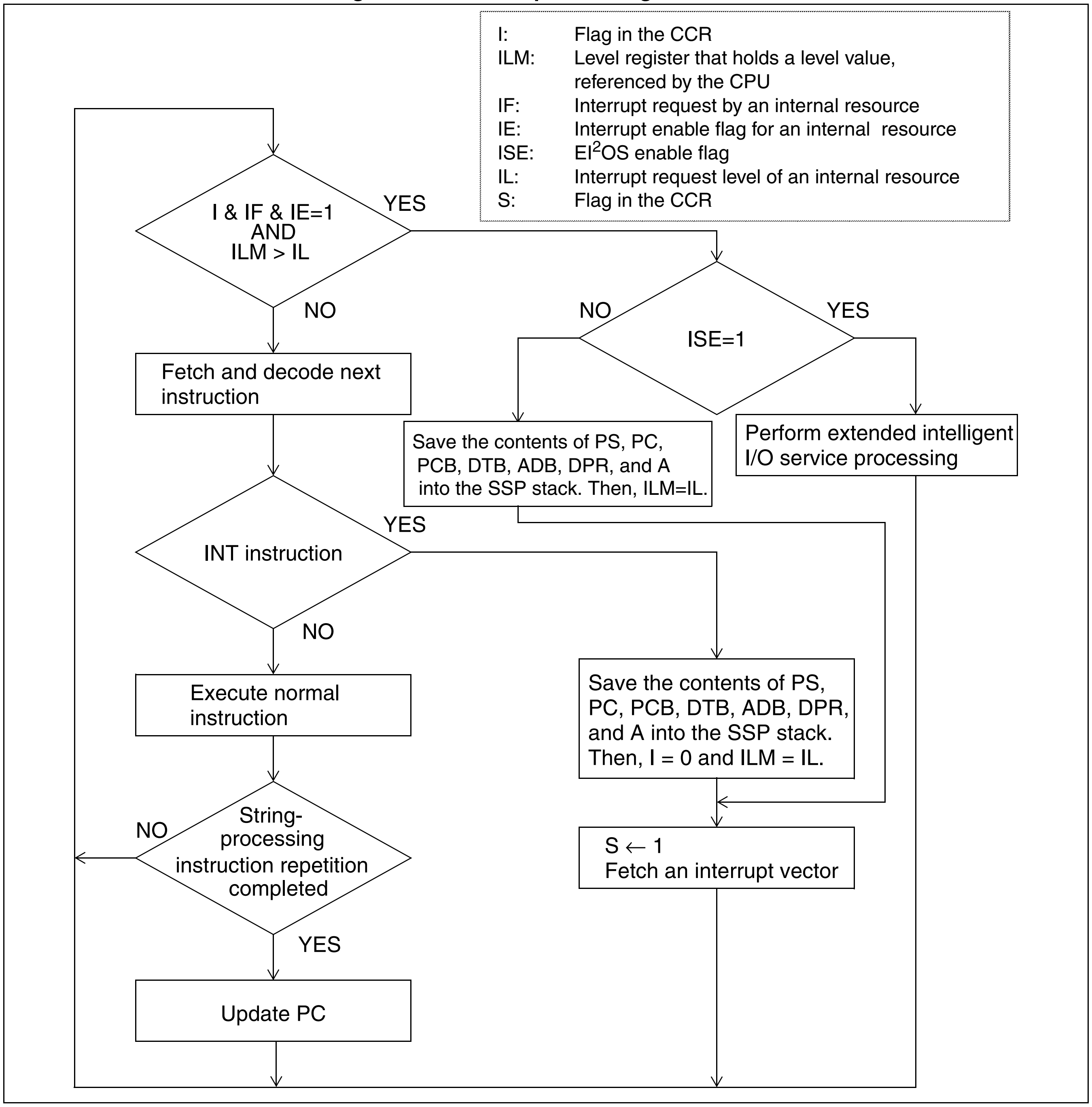
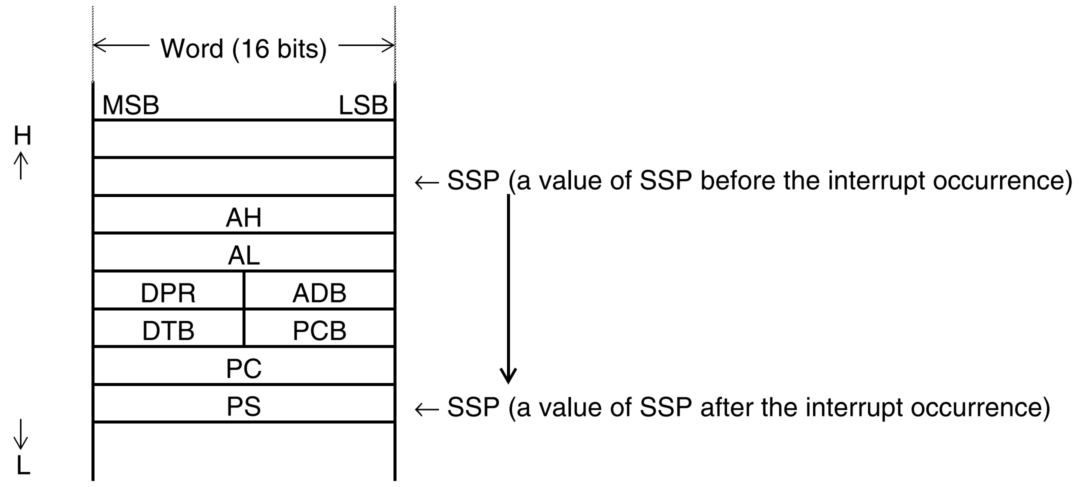

I: Flag in the CCR
ILM: Level register that holds a level value, referenced by the CPU
IF: Interrupt request by an internal resource
IE: Interrupt enable flag for an internal resource
ISE: EI^{2}OS enable flag
IL: Interrupt request level of an internal resource
S: Flag in the CCR
I & IF & IE=1 AND ILM > IL
YES
NO
Fetch and decode next instruction
ISE=1
NO
YES
Save the contents of PS, PC, PCB, DTB, ADB, DPR, and A into the SSP stack. Then, ILM=IL.
Perform extended intelligent I/O service processing
INT instruction
YES
NO
Execute normal instruction
Save the contents of PS, PC, PCB, DTB, ADB, DPR, and A into the SSP stack. Then, I = 0 and ILM = IL.
String-processing instruction repetition completed
NO
YES
S ← 1
Fetch an interrupt vector
Update PC

Figure 6.3-2 How the Contents of the Registers are Saved with Interrupt Handling



Word (16 bits)
MSB
LSB
H
L
AH
AL
DPR
ADB
DTB
PCB
PC
PS
← SSP (a value of SSP before the interrupt occurrence)
← SSP (a value of SSP after the interrupt occurrence)

6.4 Interrupt Vectors

Interrupt vectors are stored at addresses FFFC00$_H$ to FFFFFF$_H$ as shown in Table 6.4-1 Interrupt vectors share the same area for both hardware interrupt and software interrupt.

■ Interrupt Vectors

Table 6.4-1 Interrupt Vectors

Interrupt request	Vector address L	Vector address H	Vector address bank	Mode register
INT0 *1	FFFFFC$_H$	FFFFFD$_H$	FFFFFE$_H$	Not used
INT1 *1	FFFFF8$_H$	FFFFF9$_H$	FFFFFA$_H$	Not used
• • •	• • •	• • •	• • •	• • •
INT7 *1	FFFFE0$_H$	FFFFE1$_H$	FFFFE2$_H$	Not used
INT8 *2	FFFFDC$_H$	FFFFDD$_H$	FFFFDE$_H$	FFFFDF$_H$
INT9	FFFFD8$_H$	FFFFD9$_H$	FFFFDA$_H$	Not used
INT10 *3	FFFFD4$_H$	FFFFD5$_H$	FFFFD6$_H$	Not used
INT11	FFFFD0$_H$	FFFFD1$_H$	FFFFD2$_H$	Not used
• • •	• • •	• • •	• • •	• • •
INT 254	FFFC04$_H$	FFFC05$_H$	FFFC06$_H$	Not used
INT 254	FFFC00$_H$	FFFC01$_H$	FFFC02$_H$	Not used

*1: Because the vector area for the CALLV instruction is also used as the vector area for INT #vct8 (#0 to #7) when the PCB is "FF$_H$", care should be taken in using a vector for the CALLV instruction.
*2: It becomes a reset vector.
*3: It becomes a vector for exception processing.

6.5 Extended Intelligent I/O Service

The extended intelligent I/O service (EI^{2}OS) is a function for automatic data transfer between I/O and the memory. It enables the data transfer from/to I/O on a direct memory access (DMA) basis, though this was performed by the interrupt handling program before.

■ Overview of Extended Intelligent I/O Service

The extended intelligent I/O service is one type of hardware interrupt. This service achieves automatic data transfer between I/O and the memory, enabling the data transfer from/to I/O on a DMA basis, though this was formerly performed by the interrupt handling program. As compared with the method applied before as part of interrupt handling, the following advantages are added:

- Because the part of the program coded for data transfer is no longer needed, the program size is reduced.
- It is unnecessary to save the contents of registers because the internal registers are not used for data transfer, thus enhancing the transfer rate.
- Because the data transfer can be stopped according to the I/O status, unnecessary data transfer is eliminated.
- Buffer addresses can be selected without the need of increments and update.
- I/O register addresses can be selected without the need of increments and update.

When the extended intelligent I/O service is terminated, it sets the end condition before the automatic branch to the interrupt handling routine. This allows the user to know what the end condition was.

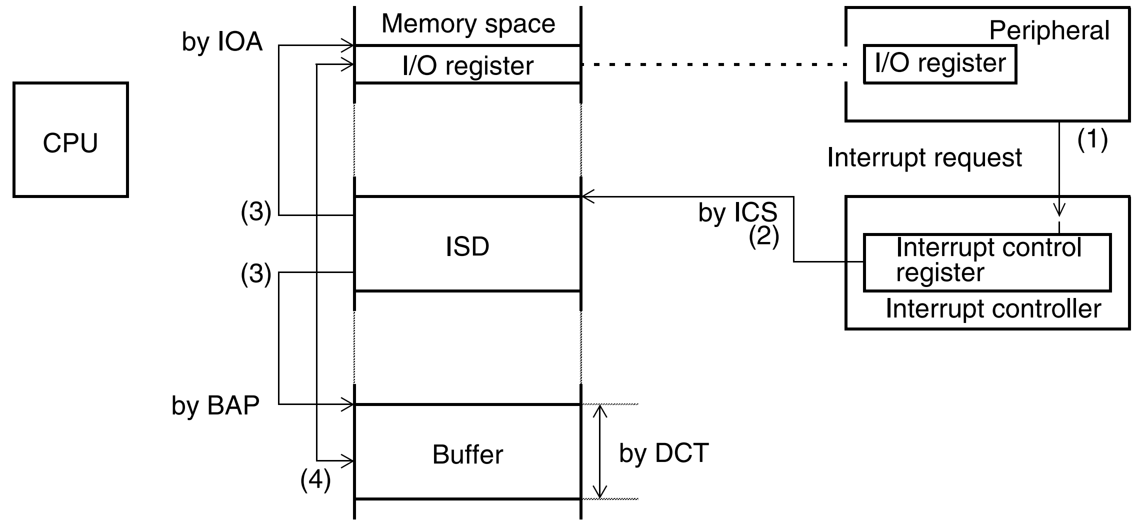
■ Structure of Extended Intelligent I/O Service

There are four functional entities below, which are related to the extended intelligent I/O service:

- Internal resource: Interrupt enable bit and interrupt request bit: Controls an interrupt request from a resource.
- Interrupt controller: ICR: Assigns an interrupt level to each interrupt request, determines the priority between the interrupts requested at the same time, and selects the operation of EI^{2}OS.
- CPU: I, ILM: Compares the requested interrupt level with the current level and verifies the interrupt enabled state.
- RAM: Descriptor: Describes the transfer information of EI^{2}OS.

Figure 6.5-1 shows the overview of extended intelligent I/O service.

Figure 6.5-1 Overview of Extended Intelligent I/O Service



(1) I/O requests data transfer.
(2) Interrupt controller selects the descriptor.
(3) Reads the transfer origin and destination from the descriptor.
(4) Data transfer between the I/O and the memory is performed.

Note:

Area that can be specified by the I/O address pointer (IOA) is 000000_H to $00FFFF_H$.

Area that can be specified by the buffer address pointer (BAP) is 000000_H to $00FFFF_H$.

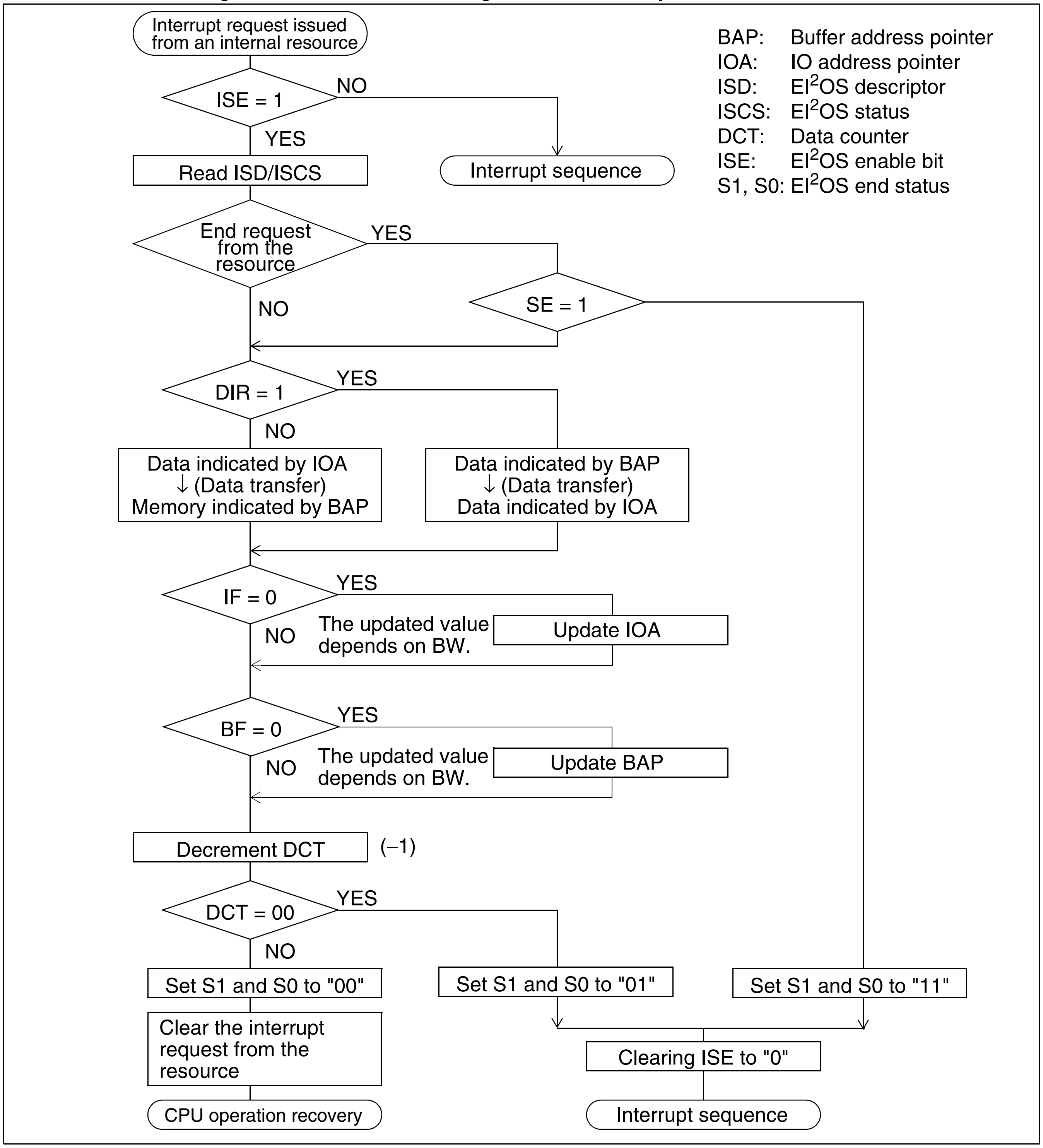
The maximum transfer count that can be specified by the data counter (DCT) is 65536.

6.5.1 Flowchart of Extended Iintelligent I/O Service Operation

Figure 6.5-2 shows the flowchart of extended intelligent I/O service operation.

■ Flowchart of Extended Intelligent I/O Service Operation

Figure 6.5-2 Extended Intelligent I/O Service Operation Flowchart

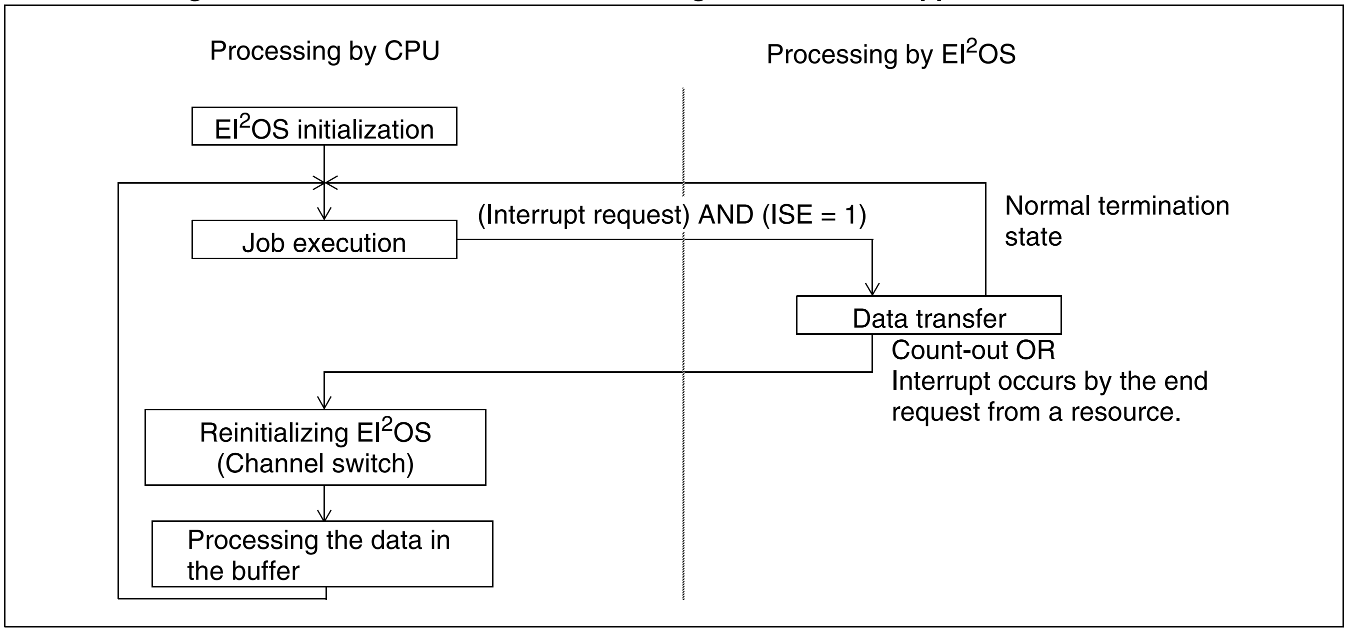


6.5.2 Flowchart of Extended Intelligent I/O Service Application Procedure

Figure 6.5-3 shows the flowchart of extended intelligent I/O service (EI^{2}OS) application procedure.

■ Flowchart of Extended Intelligent I/O Service Application Procedure

Figure 6.5-3 Flowchart of Extended Intelligent I/O Service Application Procedure



Processing by CPU
Processing by EI^{2}OS
EI^{2}OS initialization
Job execution
(Interrupt request) AND (ISE = 1)
Normal termination state
Data transfer
Count-out OR Interrupt occurs by the end request from a resource.
Reinitializing EI^{2}OS (Channel switch)
Processing the data in the buffer

6.6 Interrupt Control Register (ICR)

There are interrupt control registers (ICRs) in the interrupt controller. The number of ICRs is equivalent to the number of all I/Os (internal resource I/Os) that have the interrupt function.

■ Functions of Interrupt Control Registers (ICR0 to ICR15)

Each interrupt control register (ICR) has the following three functions:

- Sets the interrupt level of the associated internal resource.
- Selects either normal interrupt or extended intelligent I/O service to be executed for the interrupt request from the associated internal resource.
- Selects the channel for extended intelligent I/O service.

Access to this register by read-modify-write instructions should not be performed, because it may cause faulty operation.

■ Interrupt Control Register (ICR) Bit Configuration

Figure 6.6-1 shows the configuration of the bits of the interrupt control register (ICR).

Figure 6.6-1 Interrupt Control Register (ICR)

bit	15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0	
	ICS3	ICS2	ICS1/S1	ICS0/S0	ISE	IL2	IL1	IL0	00000111_B when the interrupt control register (ICR) is reset.
	W	W	*	*	R/W	R/W	R/W	R/W	

* : "1" is read by read operation.

Notes:

- ICS3 to ICS0 are effective when the extended intelligent I/O service is activated. If the extended intelligent I/O service is activated, set the ISE bit to "1". If not, set this bit to "0". Unless the extended intelligent I/O service is activated, the settings of ICS3 to ICS0 may be omitted.
- Only write is enabled for ISC1 and ICS0. Only read is enabled for S1 and S0.

6.7 Meanings of the Bits of Interrupt Control Register (ICR)

The meanings of the bits of the interrupt control register (ICR) are as follows:

- **Extended intelligent I/O service channel selection bits (ICS0 to ICS3):**
  Any combination of these bits specifies a channel for extended intelligent I/O service.
- **Extended intelligent I/O service end status (S0, S1):**
  The combinations of S0 and S1 bits indicate the end conditions of the extended intelligent I/O service.
- **Extended intelligent I/O service enable bit (ISE):**
  This bit activates the extended intelligent I/O service.
- **Interrupt level setting bits (IL0 to IL2):**
  Any combination of these bits sets an interrupt level.

■ Extended Intelligent I/O Service Channel Selection Bits (bit15 to bit12 or bit7 to bit4: ICS0 to ICS3)

These bits are used for write only and any combination of these bits specifies a channel for extended intelligent I/O service. A value set by these bits determines the address in the memory of the extended intelligent I/O service descriptor which is detailed later in this manual. All ICSs are initialized by reset. Table 6.7-1 lists the correspondence between ICSs bits, the channel numbers, and descriptor addresses.

Table 6.7-1 Correspondence between ICS Bits, Channel Numbers, and Descriptor Addresses

ICS3	ICS2	ICS1	ICS0	Selected channel	Descriptor address
0	0	0	0	0	000100_H
0	0	0	1	1	000108_H
0	0	1	0	2	000110_H
0	0	1	1	3	000118_H
0	1	0	0	4	000120_H
0	1	0	1	5	000128_H
0	1	1	0	6	000130_H
0	1	1	1	7	000138_H
1	0	0	0	8	000140_H
1	0	0	1	9	000148_H
1	0	1	0	10	000150_H
1	0	1	1	11	000158_H
1	1	0	0	12	000160_H
1	1	0	1	13	000168_H
1	1	1	0	14	000170_H
1	1	1	1	15	000178_H

■ Extended Intelligent I/O Service End Status (bit13, bit 12 or bit5, bit4: S0, S1)

These bits are used for read only. By checking a value set by these bits at the end of the extended intelligent I/O service, you can know what the end condition was. After reset, any value becomes "00". Table 6.7-2 shows the relationship between the S0 and S1 bit settings and the end conditions.

Table 6.7-2 Extended Intelligent I/O Service End Status Bits (S0 and S1) and End Conditions

S1	S0	End condition
0	0	Reserved
0	1	End by count out
1	0	Reserved
1	1	End by the request from an internal resource

■ Extended Intelligent I/O Service Enable Bit (bit11 or bit3: ISE)

This bit is read and write enabled. If an interrupt request occurs with this bit set to "1", the extended intelligent I/O service is activated. If an interrupt request occurs with this bit set to "0", the interrupt sequence is activated. Furthermore, when any end condition for the extended intelligent I/O service is met (that is, S1 and S0 bits are other than "00"), the ISE bit is cleared. If the associated internal resource is not provided with extended intelligent I/O service, the ISE bit must be set to "0" by software. The ISE bit is initialized to "0" by reset.

■ Interrupt Level Setting Bits (bit10 to bit8 or bit2 to bit0: IL2 to IL0)

These bits are read and write enabled and any combination of these bits specifies an interrupt level of the associated internal resource. The setting is initialized to level 7 (no interrupt) by reset. Table 6.7-3 shows the relationship between the interrupt level setting bits and the interrupt levels.

Table 6.7-3 Interrupt Level Setting Bits and Associated Interrupt Levels

IL2	IL1	IL0	Level value
0	0	0	0 (Highest priority)
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6 (Lowest priority)
1	1	1	7 (No interrupt)

6.8 Extended Intelligent I/O Service Descriptor (ISD)

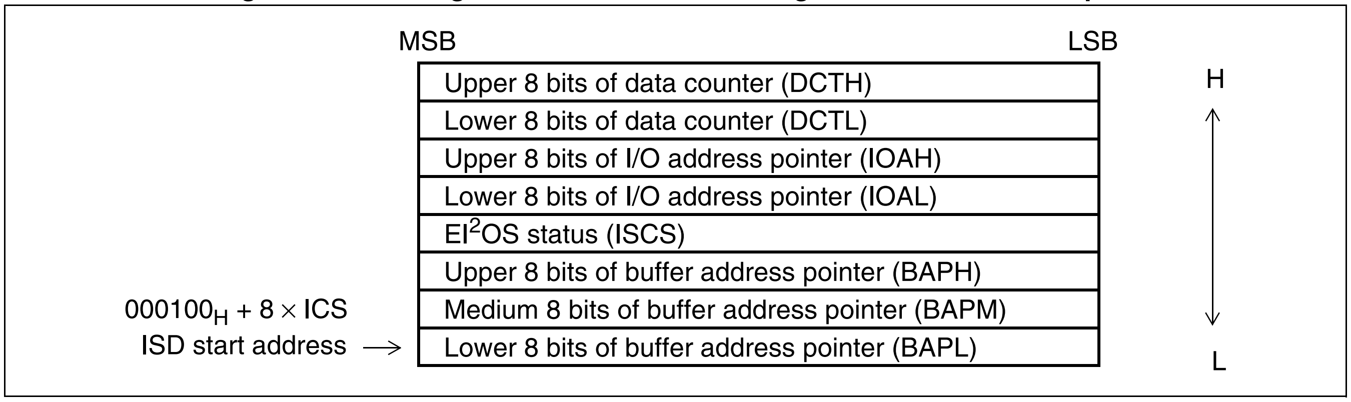
The extended intelligent I/O service descriptor (ISD) is allocated to the area of 000100_H through $00017F_H$ in the internal RAM. It consists of the followings:

- **Various types of control data for data transfer**
- **Status data**
- **Buffer address pointer**

■ Extended Intelligent I/O Service Descriptor (ISD)

Figure 6.8-1 shows the configuration of the extended intelligent I/O service descriptor (ISD).

Figure 6.8-1 Configuration of Extended Intelligent I/O Service Descriptor


MSB
LSB
Upper 8 bits of data counter (DCTH)
Lower 8 bits of data counter (DCTL)
Upper 8 bits of I/O address pointer (IOAH)
Lower 8 bits of I/O address pointer (IOAL)
EI²OS status (ISCS)
Upper 8 bits of buffer address pointer (BAPH)
Medium 8 bits of buffer address pointer (BAPM)
Lower 8 bits of buffer address pointer (BAPL)
000100H + 8 × ICS
ISD start address
H
L

6.9 Registers of Extended Intelligent I/O Service Descriptor

The extended intelligent I/O service descriptor (ISD) consists of the following registers:

- **Buffer address pointer (BAP)**
- **Extended intelligent I/O service status register (ISCS)**
- **I/O register address pointer (IOA)**
- **Data counter (DCT)**

Note that these registers are undefined when reset.

■ Buffer Address Pointer (BAP)

The buffer address pointer (BAP) is a 24-bit register that holds an address to be used in the next transfer by extended intelligent I/O service. An independent buffer address pointer (BAP) exists for each extended intelligent I/O service channel. Thus, data transfer on each extended intelligent I/O service channel is possible between an arbitrary address among 16 Mbytes and I/O.

Note:

If the BF bit in the extended intelligent I/O service status register (ISCS) indicates "update enabled", only the lower 16 bits of BAP (BAPL) will change, but the upper 8 bits (BAPH) will not change.

■ Extended Intelligent I/O Service Status Register (ISCS)

The extended intelligent I/O service status register (ISCS) is a register of 8-bit length. It indicates whether the value is updated or fixed and incremental or decremental update is enabled regarding the buffer address pointer and the I/O register address pointer. In addition, it indicates the data format (byte/word) for transfer and the transfer direction. Figure 6.9-1 shows the configuration of the extended intelligent I/O service status register (ISCS).

Figure 6.9-1 Configuration of Extended Intelligent I/O Service Status Register (ISCS)

bit	7	6	5	4	3	2	1	0	
	Reserved	Reserved	Reserved	IF	BW	BF	DIR	SE	: ISCS (undefined when reset)

Note: ISCS bit7 to bit5 must be coded with "0".

The contents of the bits of the ISCS register are as follows:

● bit4 (IF): Specifies whether the I/O register address pointer is updated or fixed.

- 0: The I/O register address pointer is updated after the data transfer.
- 1: The I/O register address pointer is not updated after the data transfer.

Note: Only increment is enabled.

● bit3 (BW): Specifies the data length for transfer.

- 0: Byte
- 1: Word

● bit2 (BF): Indicates whether the buffer address pointer is updated or fixed.

- 0: The buffer address pointer is updated after the data transfer.
- 1: The buffer address pointer is not updated after the data transfer.

Note: If updated, only the lower 16 bits of the buffer address pointer will change. Only increment is enabled.

● bit1 (DIR): Specifies the data transfer direction.

- 0: I/O $\rightarrow$ Buffer
- 1: Buffer $\rightarrow$ I/O

● bit0 (SE): Controls the termination of the extended intelligent I/O service by the request from an internal resource.

- 0: Does not terminate the extended intelligent I/O service by the request from an internal resource.
- 1: Terminates the extended intelligent I/O service by the request from an internal resource.

■ I/O Register Address Pointer (IOA)

The I/O register address pointer (IOA) is a register of 16-bit length. It indicates the lower digits of the address (A15 to A0) of the I/O register that transfers data between itself and the buffer. All upper positions of the address (A23 to A16) are coded with "0", and an arbitrary I/O address from 000000_H to $00FFFF_H$ can be specified in the upper positions.

Figure 6.9-2 shows the configuration of the I/O register address pointer (IOA).

Figure 6.9-2 Configuration of I/O Register Address Pointer (IOA)

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00	: IOA (undefined when reset)

■ Data Counter (DCT)

The data counter (DCT) is a register of 16-bit length and holds the data count for transfer. Before each data is transferred, this counter is decremented by one. When this counter value becomes "0", the extended intelligent I/O service is terminated.

Figure 6.9-3 shows the configuration of the data counter.

Figure 6.9-3 Configuration of Data Counter (DCT)

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	B15	B14	B13	B12	B11	B10	B09	B08	B07	B06	B05	B04	B03	B02	B01	B00	: DCT (undefined when reset)

6.10 Exception Processing

Exception processing is basically the same as interrupts. Upon the detection of an exceptional event on a boundary between instructions, exception processing is performed apart from normal execution. Generally, exception processing occurs as a result of an unexpected action. Thus, it is recommended to use the exception processing feature only for debugging purposes or reactivating the software for recovery in case of emergency.

■ Exception Occurrence because of the Execution of an Undefined Instruction

F^2MC-16LX handles all codes that have not been defined in the instruction map as undefined instructions.

If an undefined instruction is executed, F^2MC-16LX performs a processing similar to "INT10" which is a software interrupt instruction. That is, the execution branches to a routine indicated by the interrupt number 10 vector, after the contents of the following eight components are saved into the system stack:

- Lower bits of accumulator (AL)
- Lower bits of accumulator (AH)
- Direct page register (DPR)
- Data bank register (DTB)
- Additional data bank register (ADB)
- Program bank register (PCB)
- Program counter (PC)
- Processor status (PS)

Then, F^2MC-16LX clears the interrupt enable flag (I flag) and sets the stack flag (S flag). The value of PC saved into the stack is that address of the location where the undefined instruction is stored. For 2-byte or longer instruction codes, it is that address of the location where the code identified as being undefined is stored. It is possible to make recovery by the RETI instruction, but the same exception recurs, so there is no point in making such recovery.

CHAPTER 7 ADDRESSING

This chapter describes addressing for the F^2MC-16LX instructions.
Addressing specifies the data to be used and an address.
In F^2MC-16LX, effective addressing or an used instruction code determines the address format (absolute address or relative address). When the address format is determined by the instruction code itself, an address must be specified in compliance with the used instruction code.
Some instructions enable several types of addressing to be specified.

7.1 Effective Address Field

7.2 Direct Addressing

7.3 Indirect Addressing

7.1 Effective Address Field

Table 7.1-1 lists the address formats that may be specified in the effective address field.

■ Effective Address Field

Table 7.1-1 Effective Address Field

Code	Coding			Address format	Default bank
00	R0	RW0	RL0	Register direct Each column corresponds to the address coding in the byte, word, and long word types in the order left to right.	None
01	R1	RW1	(RL0)		
02	R2	RW2	RL1		
03	R3	RW3	(RL1)		
04	R4	RW4	RL2		
05	R5	RW5	(RL2)		
06	R6	RW6	RL3		
07	R7	RW7	(RL3)		
08	@RW0			Register indirect	DTB
09	@RW1				DTB
0A	@RW2				ADB
0B	@RW3				SPB
0C	@RW0 +			Register indirect with post-increment	DTB
0D	@RW1 +				DTB
0E	@RW2 +				ADB
0F	@RW3 +				SPB
10	@RW0 + disp8			Register indirect with 8-bit displacement	DTB
11	@RW1 + disp8				DTB
12	@RW2 + disp8				ADB
13	@RW3 + disp8				SPB
14	@RW4 + disp8			Register indirect with 8-bit displacement	DTB
15	@RW5 + disp8				DTB
16	@RW6 + disp8				ADB
17	@RW7 + disp8				SPB
18	@RW0 + disp16			Register indirect with 16-bit displacement	DTB
19	@RW1 + disp16				DTB
1A	@RW2 + disp16				ADB
1B	@RW3 + disp16				SPB
1C	@RW0 + RW7			Register indirect with index	DTB
1D	@RW1 + RW7			Register indirect with index	DTB
1E	@PC + disp16			PC indirect with 16-bit displacement	PCB
1F	addr16			Direct address	DTB

7.2 Direct Addressing

In direct addressing, a value, register, and address must be directly specified for the operands.

■ Direct Addressing

● Immediate data (#imm)

Directly specify an operand value. There are four types of immediate data according to data length as below:

- #imm4
- #imm8
- #imm16
- #imm32

● Register direct

Directly specify a register for the operand. Registers that can be specified are as below:

• General-purpose registers	(Byte):	R0, R1, R2, R3, R4, R5, R6, R7
	(Word):	RW0, RW1, RW2, RW3, RW4, RW5, RW6, RW7
	(Long word):	RL0, RL1, RL2, RL3
• Dedicated registers	(Accumulator):	A, AL
	(Pointer):	SP *
	(Bank):	PCB, DTB, USB, SSB, ADB
	(Page):	DPR
	(Control):	PS, CCR, RP, ILM

*: For SP, either user stack pointer (USP) or system stack pointer (SSP) is selected for use, according to the value of the S flag in the condition code register (CCR). For branch instructions, program counter (PC) is not described in the operand of the instruction, but it is automatically specified.

● Direct branch address (addr16)

Directly specify an address to which the execution will branch by means of displacement. The address length with displacement is 16 bits and the address indicates the destination of the branch in the logical space. This addressing is applied to an unconditional branch instruction and a subroutine call instruction. bits 16 to 23 of the address are given by the program bank register (PCB).

● Physical direct branch address (addr24)

Directly specify a physical address to which the execution will branch by means of displacement. The data length with displacement is 24 bits. This addressing is applied to an unconditional branch instruction, a subroutine call instruction, and a software interrupt instruction.

- **I/O direct (io)**

  Directly specify a memory address in the operand by means of 8-bit displacement. Independently of the respective values for data bank register (DTB) and direct page register (DPR), the I/O space with physical addresses 000000_H to $0000FF_H$ is accessible. It is invalid to describe the bank select prefix to specify a bank before an instruction using this addressing.

- **Abbreviated direct address (dir)**

  Specify lower eight bits of a memory address in the operand. Bits 8 to 15 of the address are given by the direct page register (DPR). Bits 16 to 23 of the address are given by the data bank register (DTB).

- **Direct address (addr16)**

  Specify lower 16 bits of a memory address in the operand. Bits 16 to 23 of the address are given by the data bank register (DTB).

- **I/O direct bit address (io:bp)**

  Directly specify a bit within the range of physical addresses 000000_H to $0000FF_H$. Bit position is represented by :bp. The higher number is the most significant bit and the lower number is the least significant bit.

- **Abbreviated direct bit address (dir:bp)**

  Directly specify lower eight bits of a memory address in the operand. Bits 8 to 15 of the address are given by the direct page register (DPR). Bits 16 to 23 of the address are given by the data bank register (DTB). Bit position is represented by :bp. The higher number is the most significant bit and the lower number is the least significant bit.

- **Direct bit address (addr16:bp)**

  Directly specify an arbitrary bit within 64 Kbytes. Bits 16 to 23 of the address are given by the data bank register (DTB). Bit position is represented by :bp. The higher number is the most significant bit and the lower number is the least significant bit.

- **Vector address (#vct)**

  The address to which the execution will branch is determined by the content of the vector that is specified herein. The vector number data length may be either four bits or eight bits. This addressing is applied to a subroutine call instruction and a software interrupt instruction.

7.3 Indirect Addressing

In indirect addressing, the data indicated by the operand you coded indirectly gives an address.

■ Indirect Addressing

● Register indirect (@RWj j = 0 to 3)

The register indirect addressing is used to access a memory location whose address is specified by the content of general-purpose register RWj. Bits 16 to 23 of the address are given by the data bank register (DTB) if RW0 and RW1 are used, by the SPB if RW3 is used, and by the additional data bank register (ADB) if RW2 is used.

● Register indirect with post-increment (@RWj+ j = 0 to 3)

This addressing is also used to access a memory location whose address is specified by the content of general-purpose register RWj. After the execution of the operand operation, RWj is incremented by the operand data length (1 for byte, 2 for word, and 4 for long word). Bits 16 to 23 of the address are given by the data bank register (DTB) if RW0 and RW1 are used, by the SPB if RW3 is used, and by the additional data bank register (ADB) if RW2 is used.

If the value resulting from post-increment indicates the address of the increment-specified register itself, the value of this register is incremented when referred subsequently. Then, if a data write instruction is issued to the register, the priority is given to the data write instruction, so that the register value, which would otherwise be incremented, becomes the written data.

● Register indirect with displacement (@RWi+disp8 i = 0 to 7, @RWj+disp16 j = 0 to 3)

This addressing is used to access a memory location whose address is specified by the displacement added to the content of general-purpose register RWj. Displacement may be either byte or word and is added as a signed value. Bits 16 to 23 of the address are given by the data bank register (DTB) if RW0, RW1, RW4, and RW5 are used. Bits 16 to 23 are given by the SPB if RW3 and RW7 and by the additional data bank register (ADB) if RW2 and RW6 are used.

● Long register indirect with displacement (@RLi+disp8 i = 0 to 3)

This addressing is used to access a memory location whose address is specified by the lower 24 bits that result from the displacement added to the content of general-purpose register RLi. Displacement is eight bits and added as a signed value.

● Program counter indirect with displacement (@PC+disp16)

This addressing is used to access a memory location whose address is specified by (address of instruction + 4 + disp16). Displacement is a word length. Bits 16 to 23 of the address are given by the program bank register (PCB).

Note that respective operand addresses of the instructions listed next are not regarded as being (next instruction address + disp16):

- DBNZ eam, rel
- DWBNQ eam, rel
- CBNE eam, #imm8, rel
- CWBNE eam, #imml16, rel
- MOV eam, #imm8
- MOVM eam, #imm16

● Register indirect with base index (@RW0+RW7, @RW1+RW7)

This addressing is used to access a memory location whose address is specified by a value obtained by adding the content of RW0 or RW1 to the content of general-purpose register RW7. Bits 16 to 23 of the address are given by the data bank register (DTB).

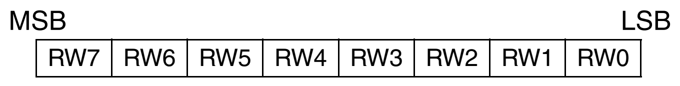
● Program counter relative branch address (rel)

The address to which the execution will branch is determined by a value obtained by adding the 8-bit displacement to the value of the program counter (PC). If the result of the addition exceeds 16 bits, the bank register is not incremented or decremented and the part of excess is ignored. Consequently, the address falls within the closed bank of 64 Kbytes. This addressing is applied to an unconditional or conditional branch instruction. Bits 16 to 23 of the address are given by the program bank register (PCB).

● Register List (rlst)

This addressing specifies a register subjected to push/pop for the stack (see Figure 7.3-1).

Figure 7.3-1 Configuration of Register List



When the bit is "1", the associated register is selected. When the bit is "0", the associated register is not selected.

● Accumulator indirect (@A)

This addressing is used to access a memory location whose address is specified by the 16-bit content of the lower bytes of the accumulator (AL). Bits 16 to 23 of the address are given by the data bank register (DTB).

● Accumulator indirect branch address (@A)

The address to which the execution will branch is determined by the 16-bit content for the lower bytes of the accumulator (AL). This address indicates the destination of the branch within the bank space. Bits 16 to 23 of the address are given by the program bank register (PCB). In the case of the jump context (JCTX) instruction, however, bits 16 to 23 of the address are given by the data bank register (DTB). This addressing is applied to an unconditional branch instruction.

● Indirectly specified branch address (@ear)

The word data with the address specified by ear corresponds to the address to which the execution will branch.

● Indirectly specified branch address (@eam)

The word data with the address specified by eam corresponds to the address to which the execution will branch.

CHAPTER 8
INSTRUCTION OVERVIEW

This chapter provides explanation for the items described in "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS" and what the symbols used therein stand for.

8.1 Instruction Overview
8.2 Symbols (Abbreviations) Used in Detailed Execution Instructions
8.3 Effective Address Field
8.4 Execution Cycles

8.1 Instruction Overview

In "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS" the following items are described for each instruction.

- **Assembler format**
- **Operation**
- **CCR**
- **Byte count**
- **Execution cycles**
- **Correction value**
- **Example**

■ Instruction Overview

In "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS" the following items are described for each instruction.

● Assembler format

The format for coding each instruction into an assembler source program is presented.

• Upper case letters and symbols:	Write them as they are into a source program.
• Lower case letters:	Rewrite them into a source program.
• Number after a lower case letter:	Indicates a bit width in the instruction.

● Operation

The operation for registers and data by instruction execution is presented.

● CCR

The status of each flag (I, S, T, N, Z, V and C) of the condition code register (CCR) is presented.

- *: Denotes that the flag changes with the instruction execution.
- –: Denotes that the flag does not change.
- S: Denotes that the flag is set with the instruction execution.
- R: Denotes that the flag is reset with the instruction execution.

● Byte count

The byte count of the instruction (machine language) after assembled is presented.

● Execution cycles

The number of instruction execution cycles is presented.

For the meaning of the letter symbol used in the table, which is presented for description of execution cycles, see Table 8.4-1 .

● Correction value

A correction value used for calculating the number of instruction execution cycles is presented. For the meanings of the letter symbols ((b), (c), and (d)) used in the table, which is presented for description of correction values, see Table 8.4-2 . The number of instruction execution cycles is determined by the sum of a value given in the column of execution cycles and a value given in the column of correction value.

● Example

An example of each instruction is presented.

All numeric values of the data given in any example are hexadecimal numbers. Any numeric value of the data given in the operand represents a hexadecimal number with suffix (H).

8.2 Symbols (Abbreviations) Used in Detailed Execution Instructions

Table 8.2-1 lists the symbols used in detailed execution instruction.

■ Symbols (abbreviations) Used in Detailed Execution Instructions

Table 8.2-1 Symbols (abbreviations) Used in Detailed Execution Instructions (1 / 2)

Coding	Meaning
A	32-bit accumulator The length of used bits varies depending on the instruction. Byte: Lower 8 bits of AL Word: 16 bits of AL Long word: 32 bits of AL and AH
AH AL	Upper 16 bits of A Lower 16 bits of A
SP	Stack pointer (USP or SSP)
PC	Program counter
PCB	Program bank register
DTB	Data bank register
ADB	Additional data bank register
SSB	System stack bank register
USB	User stack bank register
DPR	Direct page register
brg1	DTB, ADB, SSB, USB, DPR, PCB
brg2	DTB, ADB, SSB, USB, DPR
Ri	R0, R1, R2, R3, R4, R5, R6, R7
Rj	R0, R1, R2, R3
RWi	RW0, RW1, RW2, RW3, RW4, RW5, RW6, RW7
RWj	RW0, RW1, RW2, RW3
RLi	RL0, RL1, RL2, RL3
dir	Abbreviated direct addressing
addr16 addr24 ad24 0-15 ad24 16-23	Direct addressing Physical direct addressing Bits 0 to 15 of addr24 Bits 16 to 23 of addr24
io	I/O area (000000_H to $0000FF_H$)
imm4 imm8 imm16 imm32 ext (imm8)	4-bit immediate data 8-bit immediate data 16-bit immediate data 32-bit immediate data 16-bit data resulting from the signed extension of 8-bit immediate data

Table 8.2-1 Symbols (abbreviations) Used in Detailed Execution Instructions (2 / 2)

Coding	Meaning
disp8 disp16	8-bit displacement 16-bit displacement
bp	Bit offset value
vct4 vct8	Vector number (0 to 15) Vector number (0 to 255)
( )b	Bit address
re1	Specifies a PC relative branch.
ear eam	Effective addressing (codes 00 to 07) Effective addressing (codes 08 to 1F)
r1st	Register list

8.3 Effective Address Field

Table 8.3-1 lists the address formats that may be specified in the effective address field.

■ Effective Address Field

Table 8.3-1 Effective Address Field

Code	Coding			Address format	Byte count of address extension *
00 01 02 03 04 05 06 07	R0 R1 R2 R3 R4 R5 R6 R7	RW0 RW1 RW2 RW3 RW4 RW5 RW6 RW7	RL0 (RL0) RL1 (RL1) RL2 (RL2) RL3 (RL3)	Register direct ea corresponds to the address coding in the byte, word, and long word types in the order left to right.	-
08 09 0A 0B	@RW0 @RW1 @RW2 @RW3			Register indirect	0
0C 0D 0E 0F	@RW0 + @RW1 + @RW2 + @RW3 +			Register indirect with post-increment	0
10 11 12 13 14 15 16 17	@RW0 + disp8 @RW1 + disp8 @RW2 + disp8 @RW3 + disp8 @RW4 + disp8 @RW5 + disp8 @RW6 + disp8 @RW7 + disp8			Register indirect with 8-bit displacement	1
18 19 1A 1B	@RW0 + disp16 @RW1 + disp16 @RW2 + disp16 @RW3 + disp16			Register indirect with 16-bit displacement	2
1C 1D 1E 1F	@RW0 + RW7 @RW1 + RW7 @PC + disp16 addr16			Register indirect with index Register indirect with index PC indirect with 16-bit displacement Direct address	0 0 2 2

*: The byte count of address extension corresponds to "#" (byte count) shown in the instruction list and "+" shown in the byte count field of each detailed instruction.

8.4 Execution Cycles

The number of cycles required for the execution of an instruction (execution cycles) is obtained by adding a "correction value", which is determined according to the condition, to the number of "cycles" specific to each instruction. However, actual instruction execution cycles may include the cycles required for reading the program in addition to the sum of "cycles" and a "correction value".

■ Execution Cycles

The number of cycles required for the execution of an instruction is obtained by adding up the number of "cycles" specific to each instruction, a "correction value", which is determined according to the condition, and "cycles" required for program fetch.

When fetching a program stored in a memory connected to a 16-bit bus, such as built-in ROM, program fetch is performed each time the instruction under execution passes over a 2-byte (word) boundary. If data access interference occurs, it results in an increasing number of execution cycles.

When fetching a program stored in a memory connected to an 8-bit bus, which is an external data bus, program fetch is performed per byte in the instruction under execution. If data access interference occurs, it results in an increasing number of execution cycles.

During CPU intermittent operation, the access to a general-purpose register, built-in ROM, built-in RAM, built-in I/O or external bus causes the CPU clock to make a halt for a certain time. This CPU halt time is equivalent to the number of cycles specified by the CG1/CG0 bit of the low power consumption mode control register. Therefore, the number of cycles required for the execution of an instruction during the CPU intermittent operation should be calculated differently. That is, add "a correction value" determined by "the number of times for access" × "cycles" for the CPU halt time to normal execution cycles.

■ Calculating Execution Cycles

Table 8.4-1 , Table 8.4-2 , and Table 8.4-3 provide the referenced information which may help you to calculate instruction execution cycles.

Table 8.4-1 Execution Cycles Specific to Each Addressing Method of an Effective Address

Code	Operand	(a) *	Number of times of register access specific to each addressing method
		Execution cycles specific to each addressing method	
00 to 07	Ri RWi RLi	Presented in the instruction list.	Presented in the instruction list.
08 to 0B	@RWj	2	1
0C to 0F	@RWj +	4	2
10 to 17	@RWi + disp8	2	1
18 to 1B	@RWj + disp16	2	1
1C	@RW0 + RW7	4	2
1D	@RW1 + RW7	4	2
1E	@PC + disp16	2	0
1F	addr16	1	0

*: (a) is used in "~" (cycles) and "B" (correction value) in "APPENDIX B F^2MC-16LX Instruction Lists (351 Instructions)" as well as "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS".

Table 8.4-2 Correction Values for Cycles Used for Calculating Actual Execution Cycles

Operand	(b) * byte		(c) * word		(d) * long	
	The number of cycles	The number of times of access	The number of cycles	The number of times of access	The number of cycles	The number of times of access
Internal register	+0	1	+0	1	+0	2
Internal memory even address	+0	1	+0	1	+0	2
Internal memory odd address	+0	1	+2	2	+4	4
External data bus 16-bit even address	+1	1	+1	1	+2	2
External data bus 16-bit odd address	+1	1	+4	2	+8	4
External data bus 8-bit	+1	1	+4	2	+8	4

*: (b), (c), and (d) are used in "~" (cycles) and "B" (correction value) in "APPENDIX B F^2MC-16LX Instruction Lists (351 Instructions)" as well as "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS".

Note: For the application to external buses, the wait cycles for ready input and automatic ready must be added.

Table 8.4-3 Correction Values for Cycles Used for Calculating Program Fetch Cycles

Instruction	Byte boundary	Word boundary
Internal memory	–	+2
External data bus 16-bit	–	+3
External data bus 8-bit	+3	–

Notes:
- For the application to external buses, the wait cycles for ready input and automatic ready must be added.
- Actually, all program fetches do not always cause the delay for the execution of an instruction. Thus, these correction values should be used to calculate the required execution cycles in the worst case.

CHAPTER 9
DETAILED EXECUTION INSTRUCTIONS

This chapter explains each of the execution instructions used by the assembler, in reference format. The execution instructions are presented in alphabetical order.

9.1 Detailed Execution Instructions

9.1 Detailed Execution Instructions

This section explains each of the execution instructions used by the assembler, in reference format. The execution instructions are presented in alphabetical order.

■ Reading Detailed Execution Instructions

For an explanation of each of the items and symbols (abbreviations) used in the explanation of each execution instruction, see "CHAPTER 8 INSTRUCTION OVERVIEW".

For an explanation of the alphabetical characters (a), (b), (c), and (d) used in an explanation (table) of correction values and numbers of cycles, see Table 8.4-1 and Table 8.4-2 .

9.1.1 ADD (Add Byte Data of Destination and Source to Destination)

Add the byte data specified by the second operand to the byte data specified by the first operand and store the result in the first operand. If the first operand is the accumulator (A), "0" are transferred to bits 8 to 15 of A.

■ ADD (Add Byte Data of Destination and Source to Destination)

● Assembler format:

ADD A,#imm8	ADD A,dir
ADD A,ear	ADD A,eam
ADD ear,A	ADD eam,A

● Operation:

(First operand) ← (First operand)+(Second operand) (Byte addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

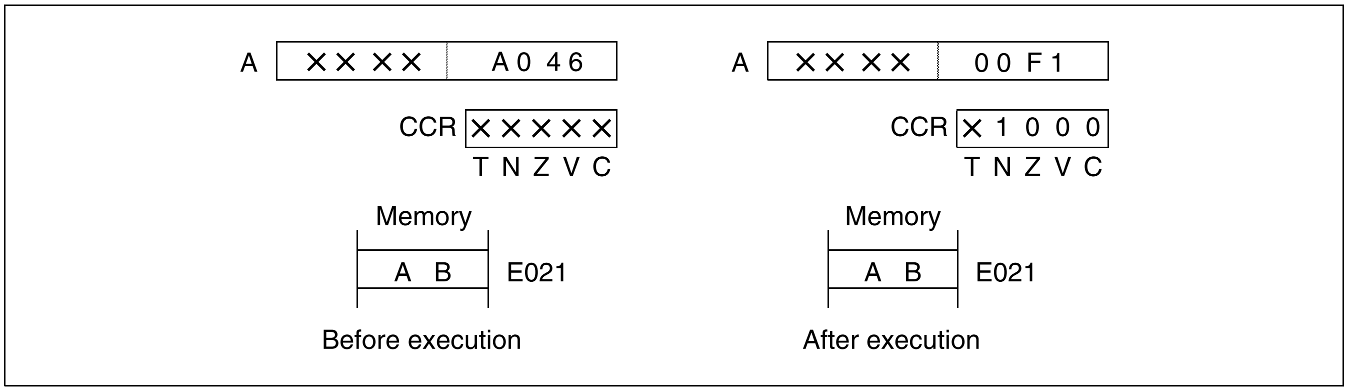
First operand	A	A	A	A	ear	eam
Second operand	#im8	dir	ear	eam	A	A
Number of bytes	2	2	2	2+	2	2+
Number of cycles	2	3	3	4+(a)	3	5+(a)
Correction value	0	(b)	0	(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ADD A,0E021H

In this example, the data (AB_H) at address $E021_H$ is added to the least significant byte data (46_H) of A.



9.1.2 ADDC (Add Byte Data of AL and AH with Carry to AL)

Add the low-order byte data of AL, low-order byte data of AH, and carry bit (C) together and restore the result in AL. "0" are transferred to the high-order byte of AL.

■ ADDC (Add Byte Data of AL and AH with Carry to AL)

- Assembler format:

ADDC A

- Operation:

(AL) ← (AH)+(AL)+(C)   (Byte addition with a carry)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

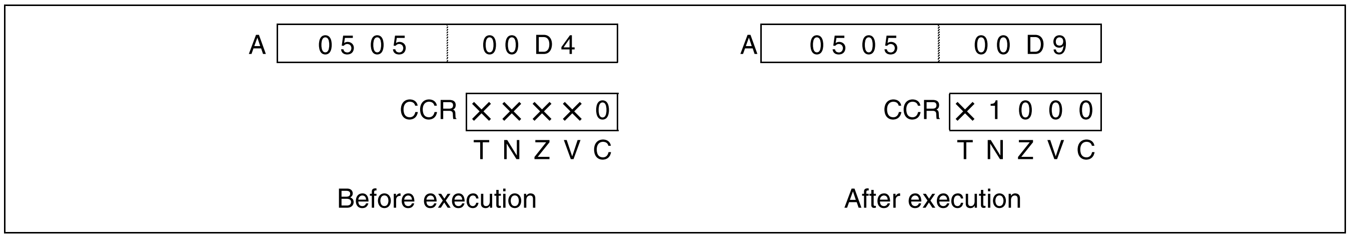
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

- Example:

ADDC A

In this example, the low-order byte data (05_H) of AH, low-order byte data ($D4_H$) of AL, and carry bit (0) are added together.



9.1.3 ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator)

Add the least significant byte data of the accumulator (A), byte data at the effective address, and carry bit (C) together and restore the result in the least significant byte of A. "0" are transferred to bits 8 to 15 of A.

■ ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator)

● Assembler format:

```
ADDC    A, ear
ADDC    A, eam
```

●Operation:

(A) ← (A)+(ea)+(C)   (Byte addition with a carry)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

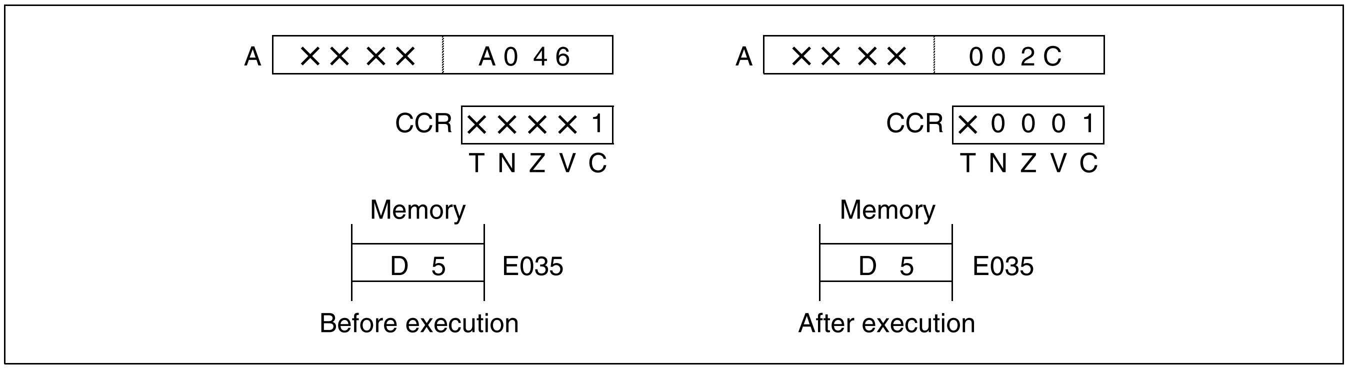
First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	4+(a)
Number of accesses	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ADDC A, 0E035H

In this example, the least significant byte data (46_H) of A, data ($D5_H$) at address $E035_H$, and carry bit (1) are added together.



9.1.4 ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator)

Add the low-order word data (AL) of the accumulator (A), word data specified by the second operand, and carry bit (C) together and restore the result in the low-order word of A.

■ ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator)

- Assembler format:

```
ADDCW     A, ear
ADDCW     A, eam
```

- Operation:

(A) ← (A)+(ea)+(C)   (Word addition with a carry)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

- Number of bytes, Number of cycles, and Correction value:

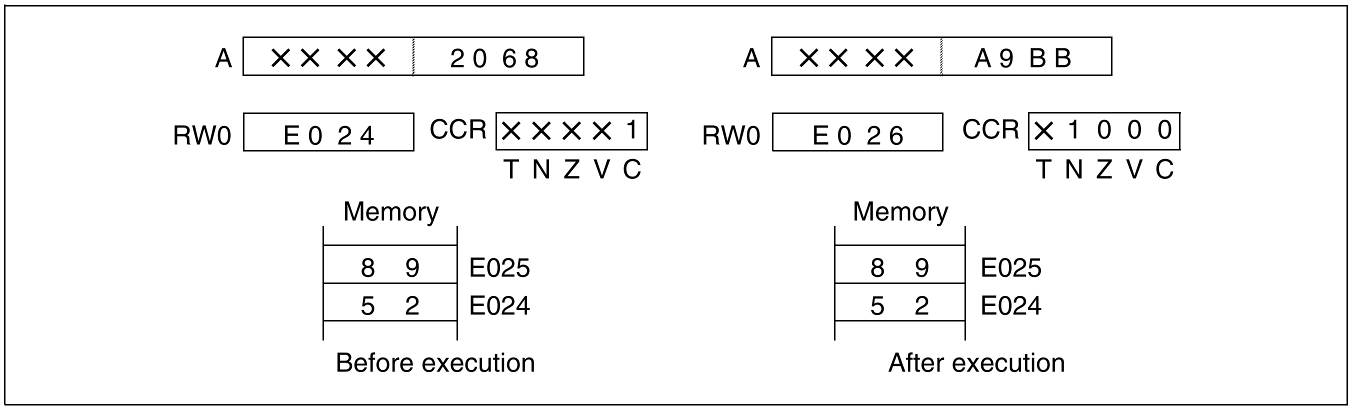
First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	4+(a)
Correction value	0	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ADDCW A,@RW0+

In this example, the low-order word data (2068_H) of A, address data (8952_H) specified by RW0, and carry bit (1) are added together.



9.1.5 ADDDC (Add Decimal Data of AL and AH with Carry to AL)

Add the low-order byte data of AL, low-order byte data of AH, and carry bit (C) together in decimal and restore the result in the low-order byte of AL. "0" are transferred to the high-order byte of AL.

■ ADDDC (Add Decimal Data of AL and AH with Carry to AL)

- Assembler format:

ADDDC A

- Operation:

(AL) ← (AH)+(AL)+(C) (Decimal addition with a carry)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Undefined
C:	Set when a carry has occurred as a result of the decimal operation, cleared otherwise.

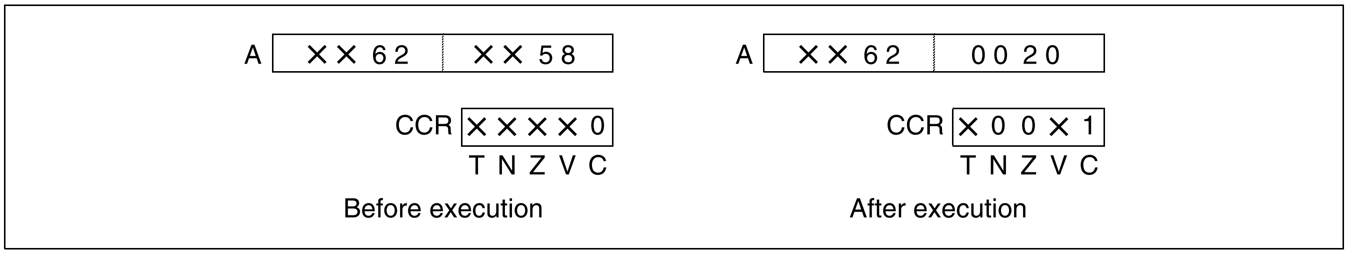
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3
Correction value:	0

- Example:

ADDDC A

In this example, the low-order byte data (62_H) of AL, low-order byte data (58_H) of AH, and carry bit (C) are added together in decimal operation.



9.1.6 ADDL (Add Long Word Data of Destination and Source to Destination)

Add the long word data specified by the second operand to the long word data specified by the first operand and restore the result in the first operand.

■ ADDL (Add Long Word Data of Destination and Source to Destination)

- Assembler format:

ADDL A,#imm32

ADDL A,ear   ADDL A,eam

- Operation:

(First operand) ← (First operand)+(Second operand)   (Long word addition)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

- Number of bytes, Number of cycles, and Correction value:

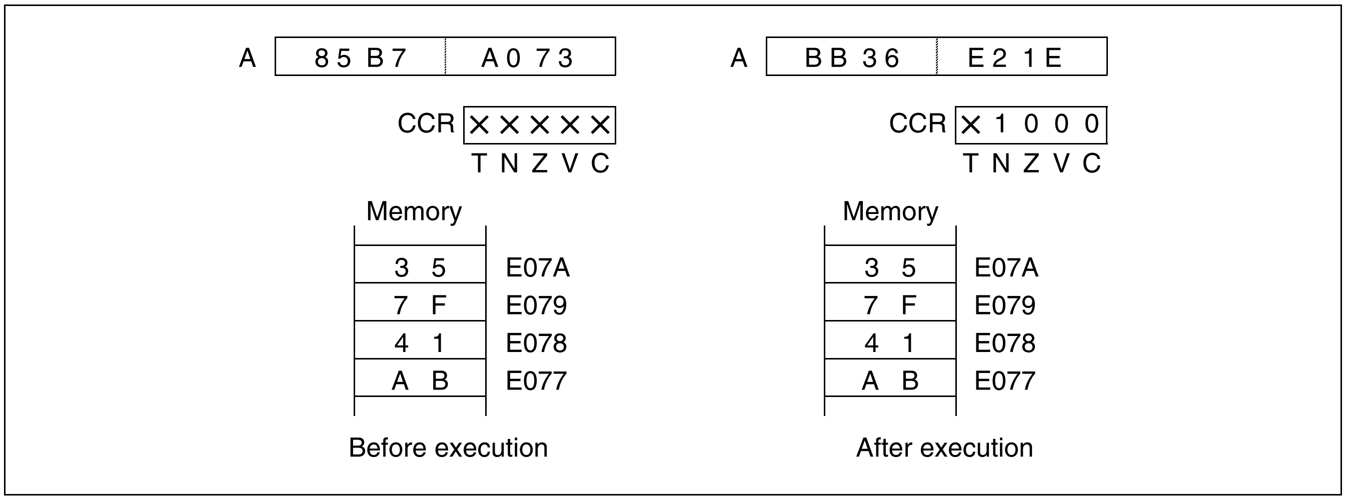
First operand	A	A	A
Second operand	#i32	ear	eam
Number of bytes	5	2	2+
Number of cycles	4	6	7+(a)
Correction value	0	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ADDL A,0E077H

In this example, the data (357F41AB$_H$) at addresses E077$_H$ to E07A$_H$ is added to the data (85B7A073$_H$) of A.



9.1.7 ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer)

Add 16-bit immediate data or the value resulting from sign-extending 8-bit immediate data to the word data pointed to by SP (stack pointer) and restore the result in SP. If the addition result exceeds 16 bits, an underflow occurs.
CCR does not indicate whether an underflow has occurred.

■ ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer)

● Assembler format:

(1) ADDSP #imm8
(2) ADDSP #imm16

● Operation:

(1) (SP) ← (SP)+Sign-extended #imm8 (Word addition)
(2) (SP) ← (SP)+#imm16 (Word addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

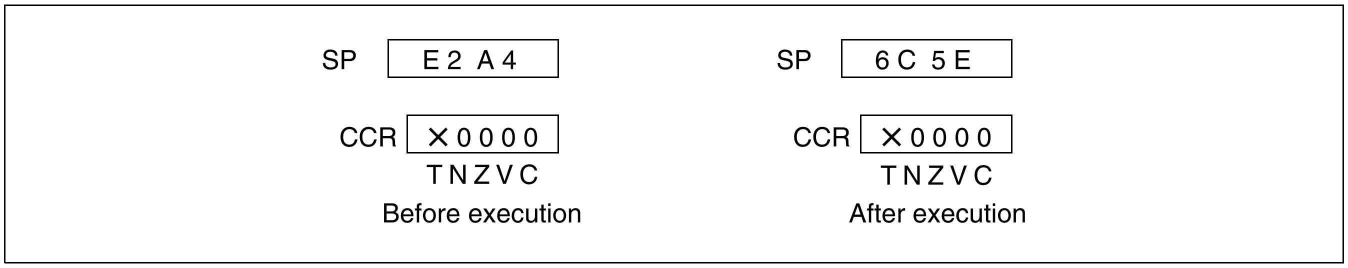
● Number of bytes, Number of cycles, and Correction value:

Operand	#im8	#i16
Number of bytes	2	3
Number of cycles	3	3
Correction value	0	0

● Example:

ADDSP #89BAH

In this example, $89BA_H$ is added. The addition result exceeds 16 bits, causing an underflow.



9.1.8 ADDW (Add Word Data of AL and AH to AL)

Add the word data of AH and that of AL together and restore the result to AL.

■ ADDW (Add Word Data of AL and AH to AL)

● Assembler format:

ADDW A

● Operation:

(AL) ← (AH)+(AL)   (Word addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

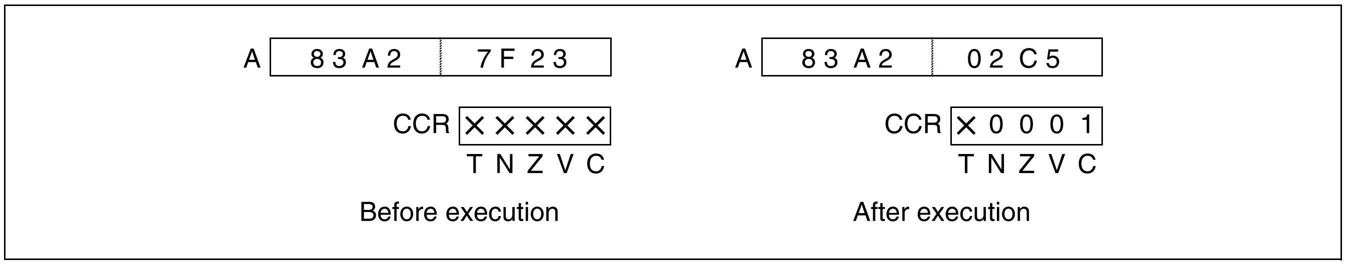
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

ADDW A

In this example, a carry occurs, causing the carry flag to be set.



9.1.9 ADDW (Add Word Data of Destination and Source to Destination)

Add the word data specified by the second operand to the word data specified by the first operand and restore the result in the first operand.

■ ADDW (Add Word Data of Destination and Source to Destination)

● Assembler format:

ADDW A,#imm16

ADDW A,ear ADDW A,eam

ADDW ear,A ADDW eam,A

● Operation:

(First operand) ← (First operand)+(Second operand) (Word addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a carry has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

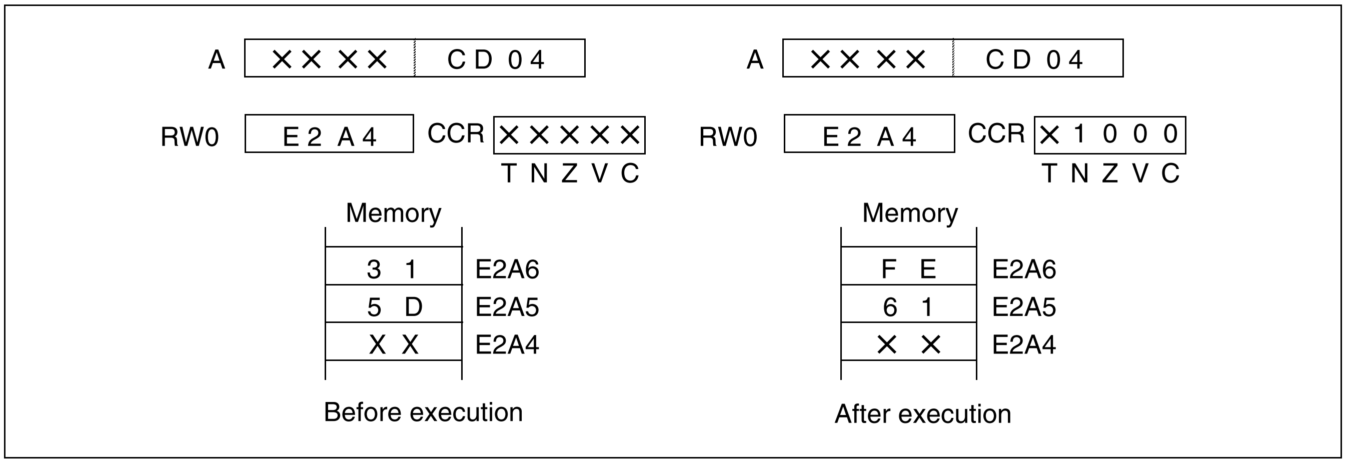
First operand	A	A	A	ear	eam
Second operand	#i16	ear	eam	A	A
Number of bytes	3	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ADDW @RW0+1,A

In this example, the low-order word data ($CD04_H$) of the accumulator is added to the address value ($315D_H$) specified by @RW0+1.



9.1.10 AND (And Byte Data of Destination and Source to Destination)

Take the logical AND operation of the byte data specified by the first operand and the byte data specified by the second operand and restore the result in the first operand.

■ AND (And Byte Data of Destination and Source to Destination)

● Assembler format:

AND A,#imm8

AND A,ear AND A,eam

AND ear,A AND eam,A

● Operation:

(First operand) ← (First operand) and (Second operand) (Byte logical AND)

The logical AND operation of the byte data specified by the first operand and the byte data specified by the second operand is taken on a bit-by-bit basis and the result is restored in the first operand.

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

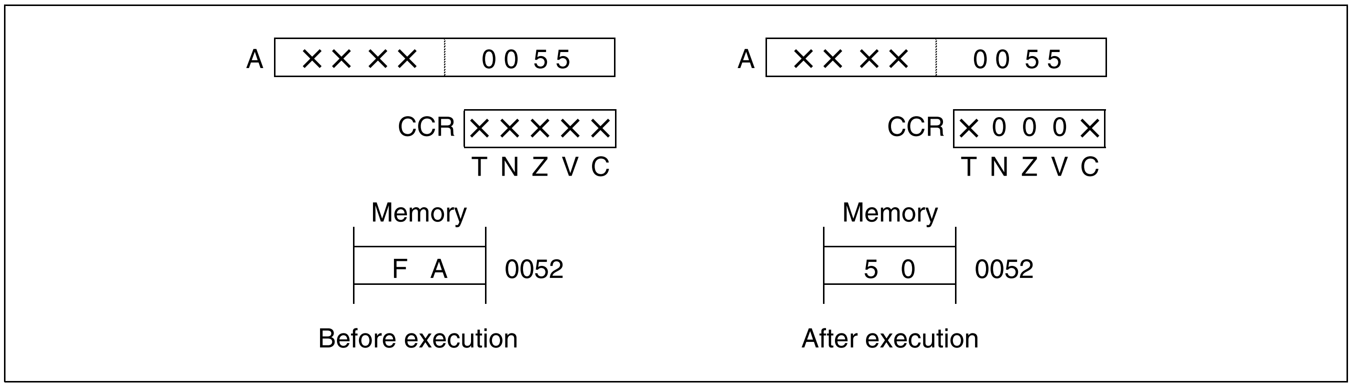
First operand	A	A	A	ear	eam
Second operand	#im8	ear	eam	A	A
Number of bytes	2	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

AND 0052H,A

In this example, the logical AND operation is taken of the address data (FA_H) at 0052_H and the least significant byte data (55_H) of the accumulator.



9.1.11 AND (And Byte Data of Immediate Data and Condition Code Register)

Take the logical AND operation of the byte data of the condition code register (CCR) and 8-bit immediate data and restore the result in CCR.
In the logical AND operation, the most significant bit of the byte data is not taken into consideration.

■ AND (And Byte Data of Immediate Data and Condition Code Register)

● Assembler format:

AND CCR,#imm8

● Operation:

(CCR) $\leftarrow$ (CCR) and #imm8 (Byte logical AND)

● CCR:

I	S	T	N	Z	V	C
*	*	*	*	*	*	*

I	:	Stores bit 6 of the operation result.
S	:	Stores bit 5 of the operation result.
T	:	Stores bit 4 of the operation result.
N	:	Stores bit 3 of the operation result.
Z	:	Stores bit 2 of the operation result.
V	:	Stores bit 1 of the operation result.
C	:	Stores bit 0 of the operation result.

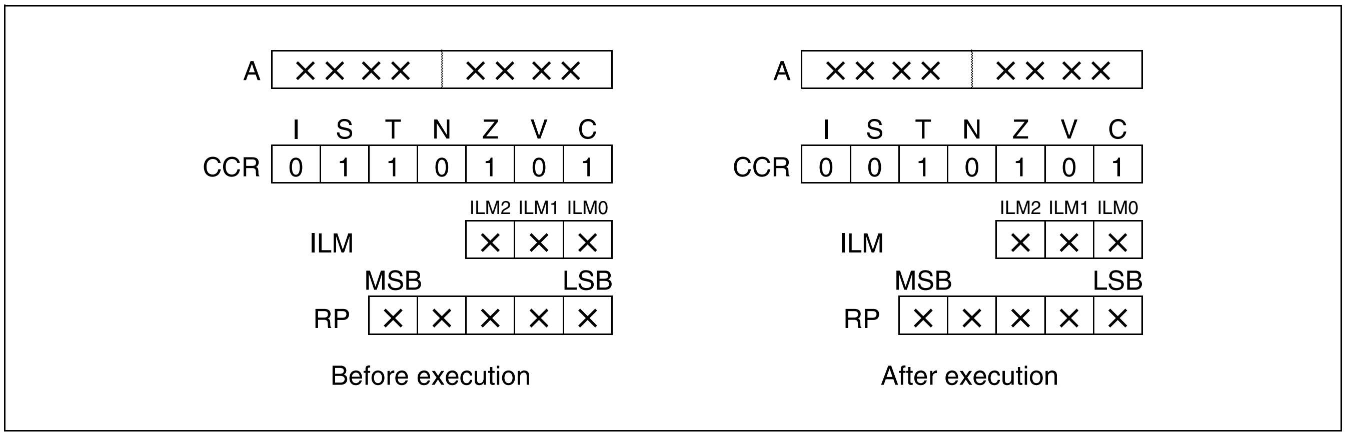
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3
Correction value:	0

● Example:

AND CCR,#57H

In this example, the logical AND operation is taken of the value (0110101_B) of the condition code register (CCR) and 57_H.



9.1.12 ANDL (And Long Word Data of Destination and Source to Destination)

Take the logical AND operation for the long word data of the accumulator (A) and that specified by the second operand in a bit-by-bit basis and restore the result in A.

■ ANDL (And Long Word Data of Destination and Source to Destination)

- Assembler format:

ANDL A,ear ANDL A,eam

- Operation:

(A) ← (A) and (Second operand) (Long word logical AND)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

- Number of bytes, Number of cycles, and Correction value:

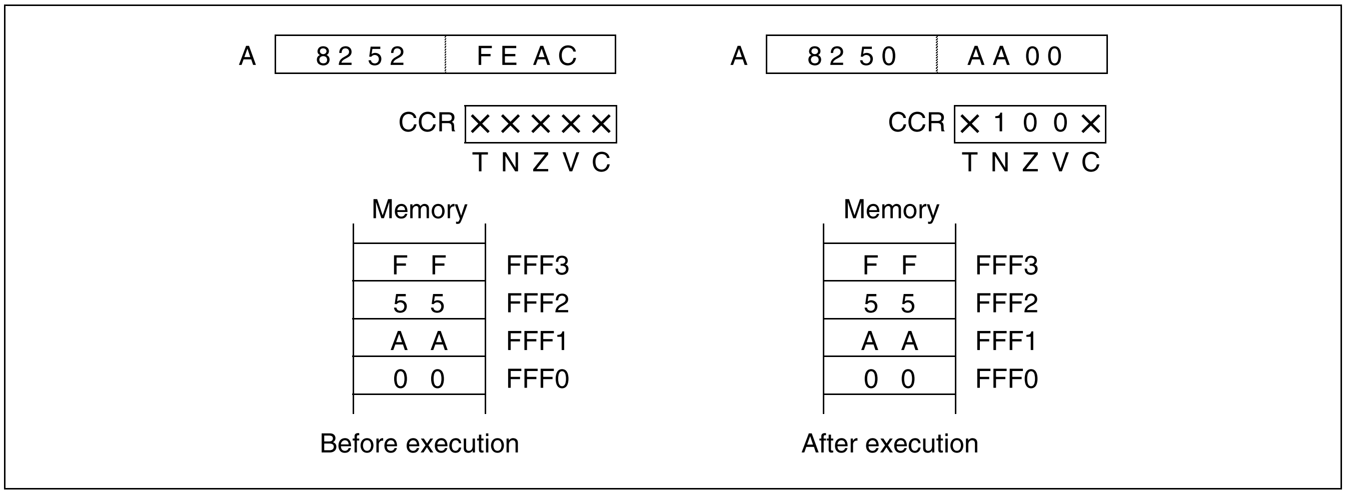
First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	6	7+(a)
Correction value	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ANDL A,0FFF0H

In this example, the logical AND operation is taken of the long word data ($8252FEAC_H$) of A and the data ($FF55AA00_H$) at $0FFF0_H$ to $0FFF3_H$, in a bit-by-bit basis.



9.1.13 ANDW (And Word Data of AH and AL to AL)

Take the logical AND operation of the word data of AH and that of AL and restore the result in AL.

■ ANDW (And Word Data of AH and AL to AL)

- Assembler format:

ANDW A

- Operation:

(AL) ← (AH) and (AL) (Word logical AND)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

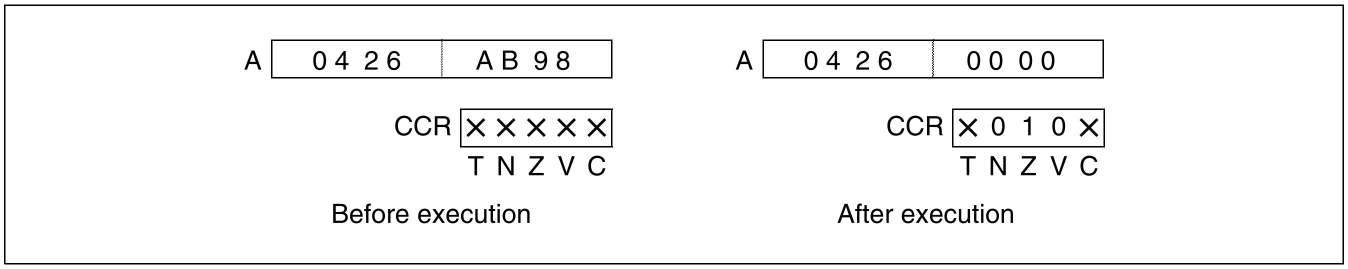
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

- Example:

ANDW A

In this example, the logical AND operation is taken of the word data (0426_H) of AH and word data ($AB98_H$) of AL.



9.1.14 ANDW (And Word Data of Destination and Source to Destination)

Take the logical AND operation of the word data specified by the first operand and the word data specified by the second operand and restore the reresult in the first operand.

■ ANDW (And Word Data of Destination and Source to Destination)

● Assembler format:

ANDW A,#imm16	
ANDW A,ear	ANDW A,eam
ANDW ear,A	ANDW eam,A

● Operation:

(First operand) ← (First operand) and (Second operand) (Word logical AND)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

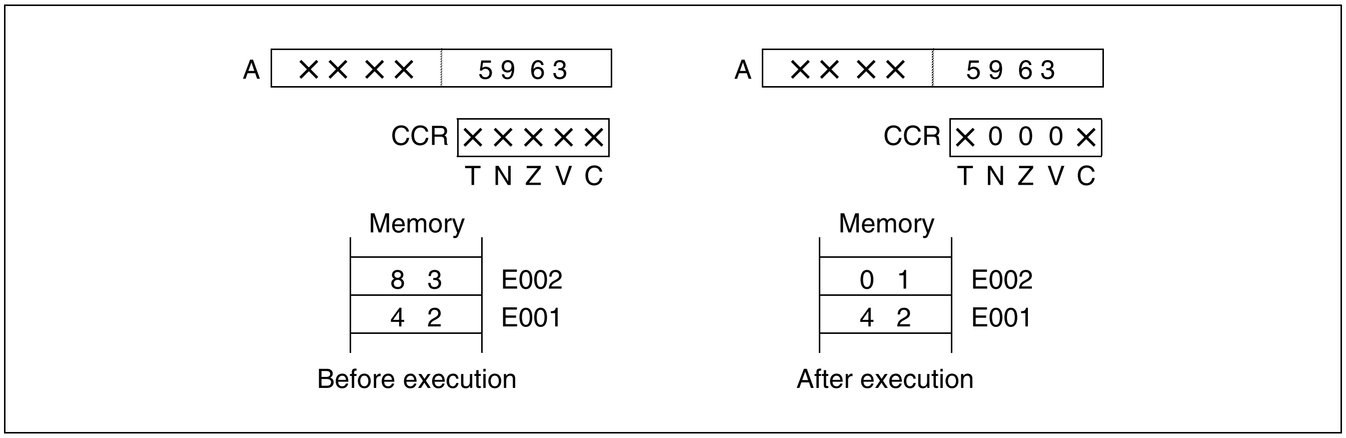
First operand	A	A	A	ear	eam
Second operand	#i16	ear	eam	A	A
Number of bytes	3	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ANDW 0E001H,A

In this example, the logical AND operation is taken of the word data (8342_H) at addresses $0E001_H$ and $0E002_H$ and the low-order word data (5963_H) of the accumulator.



9.1.15 ASR (Arithmetic Shift Byte Data of Accumulator to Right)

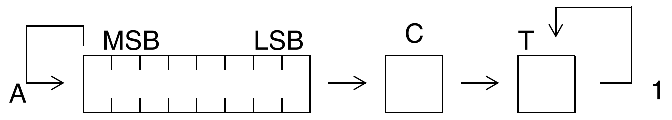
Shift the least significant byte data of the accumulator (A) arithmetically to the right by the number of bits specified by the second operand. The most significant bit of the least significant byte data for A is not changed. The bit last shifted out from the least significant bit is stored in the carry bit (C) of the condition code register (CCR).

■ ASR (Arithmetic Shift Byte Data of Accumulator to Right)

● Assembler format:

ASR A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

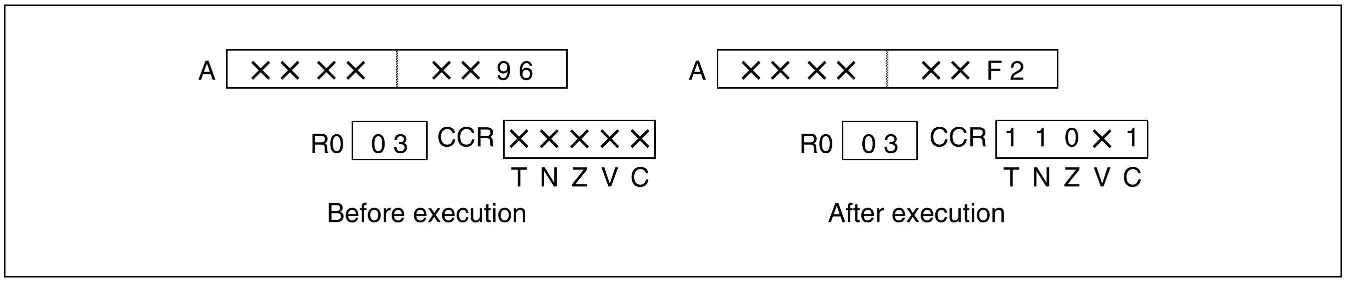
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 5 + (R0)
Correction value:	0

● Example:

ASR A,R0

In this example, the least significant byte data (96_H) of A is shifted arithmetically to the right by three bits.



9.1.16 ASRL (Arithmetic Shift Long Word Data of Accumulator to Right)

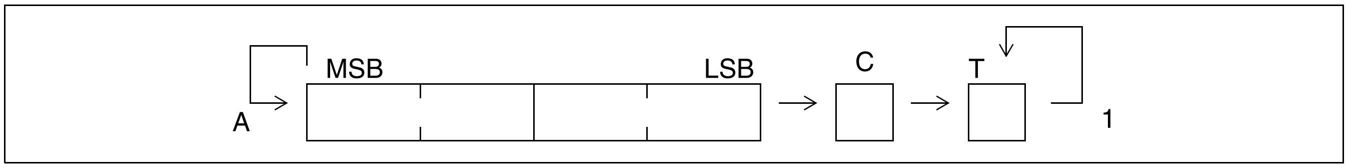
Shift the long word data of the accumulator (A) arithmetically to the right by the number of bits specified by the second operand. The most significant bit of A is not changed. The bit last shifted out from the least significant bit is stored in the carry bit (C) of the condition code register (CCR).

■ ASRL (Arithmetic Shift Long Word Data of Accumulator to Right)

● Assembler format:

ASRL A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

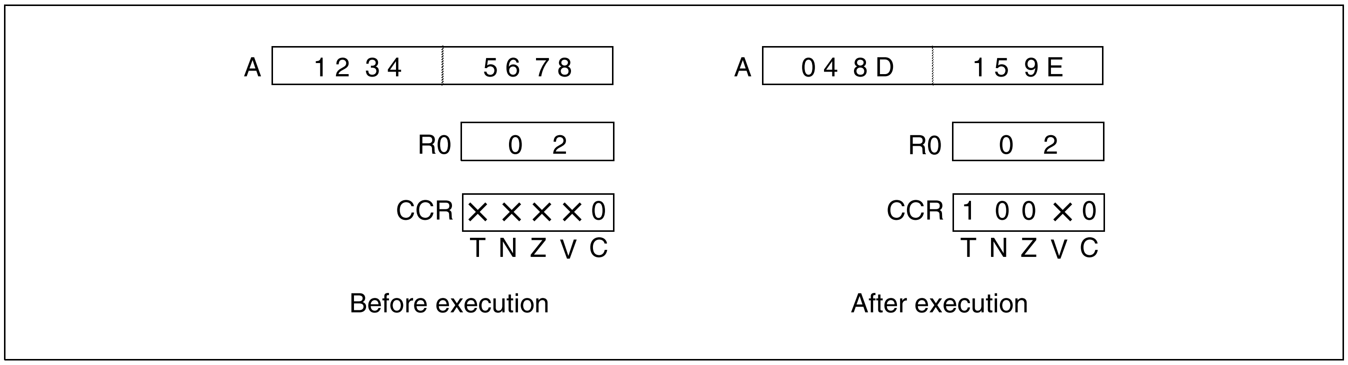
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 6 + (R0)
Correction value:	0

● Example:

ASRL A,R0

In this example, the long word data (12345678_H) of A is shifted arithmetically to the right by two bits.



9.1.17 ASRW (Arithmetic Shift Word Data of Accumulator to Right)

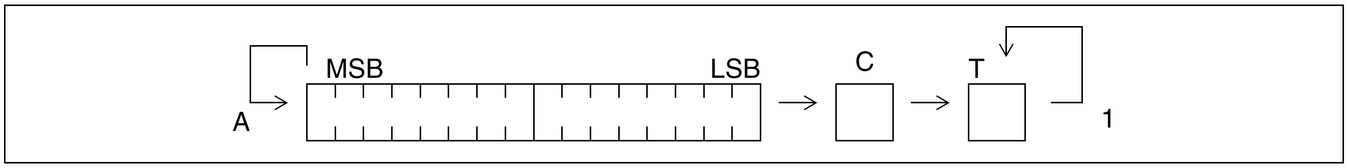
Shift the low-order word data of the accumulator (A) arithmetically to the right by one bit. The most significant bit of the low-order word data for A is not changed. The bit shifted out from the least significant bit is stored in the carry bit (C).

■ ASRW (Arithmetic Shift Word Data of Accumulator to Right)

● Assembler format:

ASRW A

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

I and S:	Unchanged
T:	Set when the old carry value is equal to "1" or the old T value is equal to "1", cleared otherwise.
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit shifted out from the LSB of A.

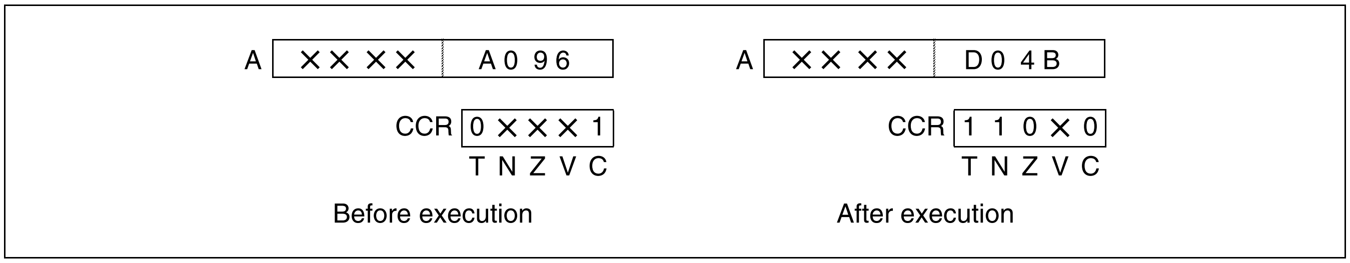
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

ASRW A

In this example, the low-order word data (A096$_H$) of A is shifted arithmetically to the right by one bit.



9.1.18 ASRW (Arithmetic Shift Word Data of Accumulator to Right)

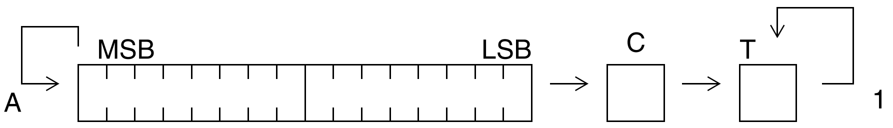
Shift the low-order word data of the accumulator (A) arithmetically to the right by the number of bits specified by the second operand. The most significant bit of the low-order word data for A is not changed. The bit last shifted out from the least significant bit is stored in the carry bit (C) of the condition code register (CCR).

■ ASRW (Arithmetic Shift Word Data of Accumulator to Right)

● Assembler format:

ASRW A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

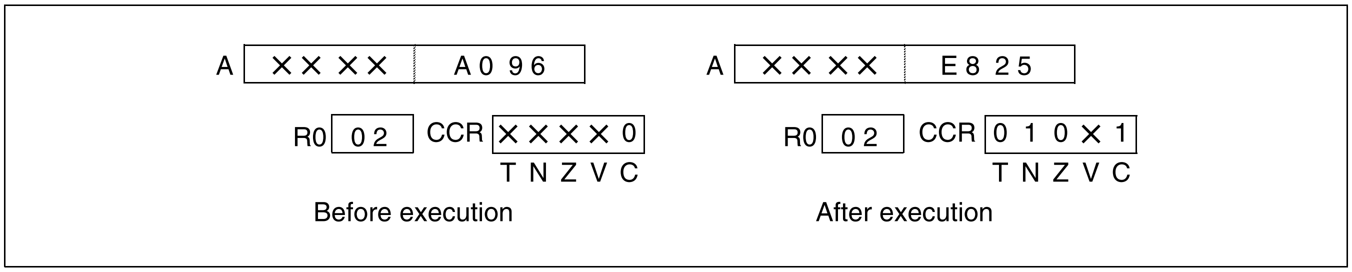
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of states:	6 when (R0) is equal to 0; otherwise, 5 + (R0)
Correction value:	0

● Example:

ASRW A,R0

In this example, the low-order word data ($A096_H$) of A is shifted arithmetically to the right by two bits.



9.1.19 BBcc (Branch if Bit Condition satisfied)

Cause a branch if the bit data specified by the first operand satisfies the condition. Control is transferred to the address resulting from word-adding the sign-extended data, specified by the second operand, to the address of the instruction following the BBcc instruction.

■ BBcc (Branch if Bit Condition satisfied)

● Assembler format:

BBC <First operand>,rel BBS <First operand>,rel

● Operation:

If the condition is satisfied:	(PC) ← (PC) + <Number of bytes> + rel	(Word addition)
If the condition is not satisfied:	(PC) ← (PC)+<Number of bytes>	(Word addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	*	–	–

I, S, T, and N:Unchanged

Z: Set when the bit data is "0"; cleared when "1".

V and C: Unchanged

● Number of bytes, Number of cycles, and Correction value:

BBcc	BBC			BBS		
Condition	Bit data = 0			Bit data = 1		
First operand	addr16:bp	dir:bp	io:bp	addr16:bp	dir:bp	io:bp
Number of bytes	5	4	4	5	4	4
Number of cycles *1	If a branch is taken: 8 If a branch is not taken: 7	If a branch is taken: 8 If a branch is not taken: 7	If a branch is taken: 7 If a branch is not taken: 6	If a branch is taken: 8 If a branch is not taken: 7	If a branch is taken: 8 If a branch is not taken: 7	If a branch is taken: 7 If a branch is not taken: 6
Correction value *2	(b)	(b)	(b)	(b)	(b)	(b)

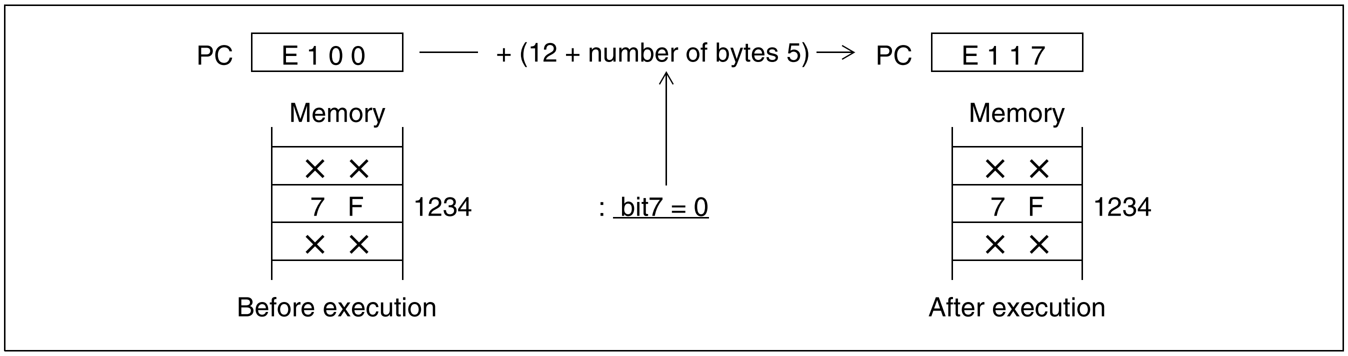
*1: “If a branch is taken” indicates the number of cycles assumed if a branch is taken. “If a branch is not taken” indicates the number of cycles assumed if a branch is not taken.

*2: For the explanation of (b) in the table, see Table 8.4-2 .

● Example:

BBC 1234H:7,12H

In this example, a branch is taken if bit 7 of the data at memory address 1234_H is equal to "0" (condition satisfied).



9.1.20 Bcc (Branch relative if Condition satisfied)

Each instruction causes a branch if the condition determined for that instruction is satisfied. Control is transferred to the address resulting from word-adding the sign-extended data, specified by the operand, to the address of the instruction following the BBcc instruction.

■ Bcc (Branch relative if Condition satisfied)

- Assembler format:

BZ/BEQ	rel	BNZ/BNE	rel
BC/BLO	rel	BNC/BHS	rel
BN	rel	BP	rel
BV	rel	BNV	rel
BT	rel	BNT	rel
BLT	rel	BGE	rel
BLE	rel	BGT	rel
BLS	rel	BHI	rel
BRA	rel		

- Operation:

If the condition is satisfied:	(PC) ← (PC)+2+rel	(Word addition)
If the condition is not satisfied:	(PC) ← (PC)+2	(Word addition)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes: 2

Number of cycles: 3 when branching is not performed, 4 otherwise.

Correction value: 0

Branch instruction and condition:

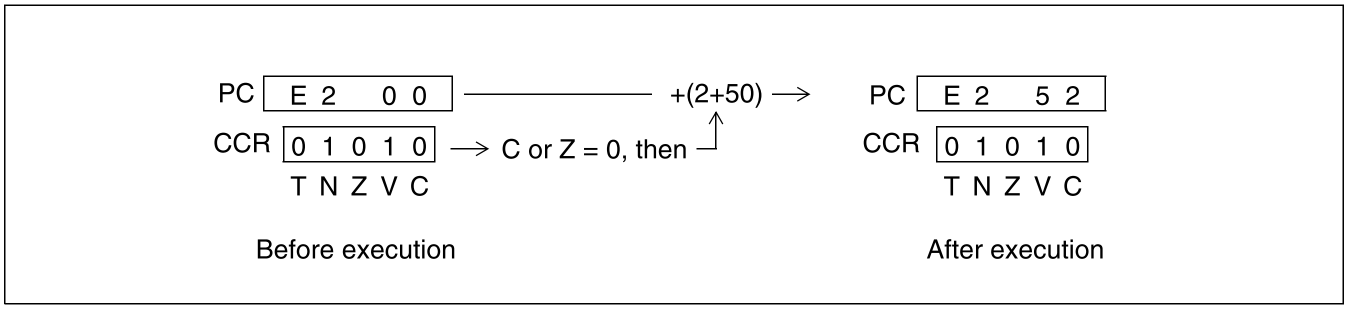
Bcc	BZ/ BEQ	BNZ/ BNE	BC/ BLO	BNC/ BHS	BN	BP	BV	BNV	BT	BNT	BRA
Condition	Z=1	Z=0	C=1	C=0	N=1	N=0	V=1	V=0	T=1	T=0	Always satisfied

Bcc	BLT	BGE	BLE	BGT	BLS	BHI
Condition	V xor N = 1	V xor N = 0	(V xor N) or Z = 1	(V xor N) or Z = 0	C or Z = 1	C or Z = 0

● Example:

BHI 50H

In this example, a branch is taken if either C or Z or both of the condition code register (CCR) are equal to "0" (condition satisfied).



9.1.21 CALL (Call Subroutine)

Cause a branch to the address specified by the operand. By executing the RET instruction in the subroutine to which control has been transferred, control returns to the instruction following the CALL instruction.

■ CALL (Call Subroutine)

● Assembler format:

CALL @ear CALL @eam
CALL addr16

● Operation:

(SP) ← (SP)–2 (Word subtraction), ((SP)) ← (PC)+<Number of bytes>
(PC) ← <Operand>

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

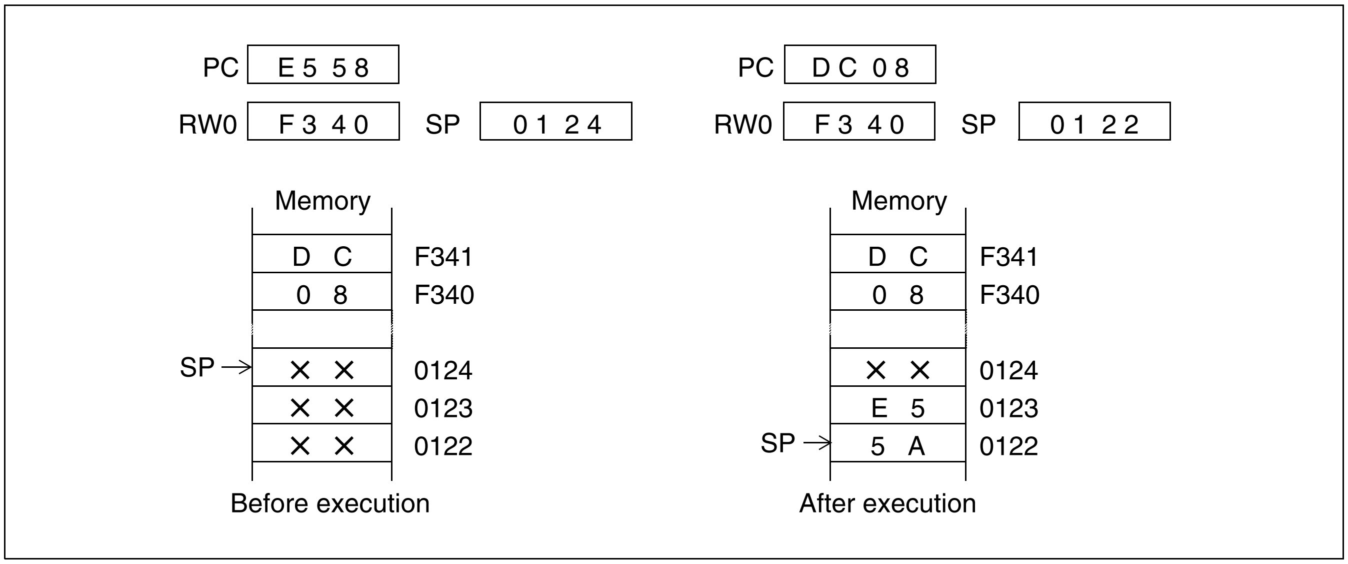
● Number of bytes, Number of cycles, and Correction value:

Operand	@ear	@eam	ad16
Number of bytes	2	2+	3
Number of cycles	6	7+(a)	6
Correction value	(c)	2×(c)	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CALL @RW0



9.1.22 CALLP (Call Physical Address)

Cause a branch to the physical address specified by the operand. The program bank register (PCB) stores the most significant byte of the data specified by the operand. By executing the RETP instruction in the subroutine to which control has been transferred, control returns to the instruction following the CALLP instruction.

■ CALLP (Call physical Address)

● Assembler format:

CALLP @ear CALLP @eam

CALLP addr24

● Operation:

(SP) ← (SP)–2 (Word subtraction), ((SP)) ← (PCB) (Zero extension)

(SP) ← (SP)–2 (Word subtraction), ((SP)) ← (PC)+<Number of bytes>

(PCB) ← Physical address to branch to (High-order byte)

(PC) ← Physical address to branch to (Low-order word)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

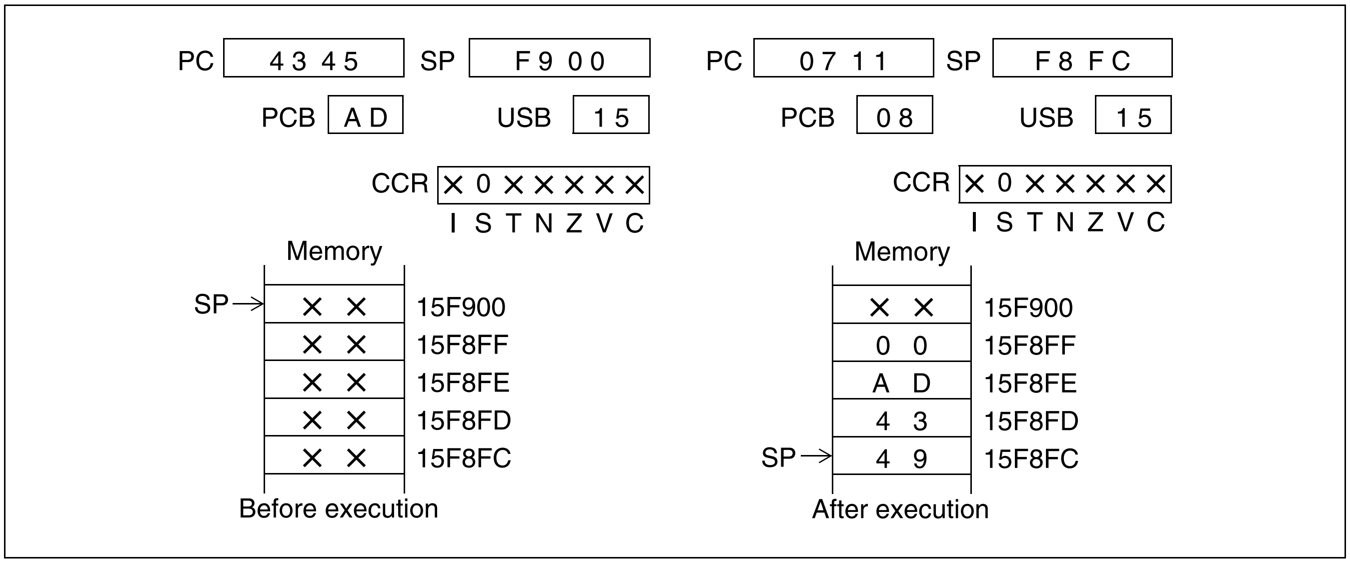
Operand	@ear	@eam	ad24
Number of bytes	2	2+	4
Number of cycles	10	11+(a)	10
Correction value	2×(c)	3×(c)+(b)	2×(c)

For the explanation of (a), (b), and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CALLP 080711H

In this example, the most significant byte (08_H) of the operand is set in the program bank register (PCB).



9.1.23 CALLV (Call Vectored Subroutine)

Cause a branch to the address pointed to by the interrupt vector specified by the operand. By executing the RET instruction in the subroutine to which control has been transferred, control returns to the instruction following the CALLV instruction. The RET instruction is the same as that used with the CALL instruction.

■ CALLV (Call Vectored Subroutine)

- Assembler format:

  CALLV #vct4

- Operation:

  (SP) ← (SP)–2 (Word subtraction)   ((SP)) ← (PC) + 1

  (PC) ← Vector address

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

  None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	7
Correction value:	2×(c)

  Note: For the explanation of (c), see Table 8.4-2 .

Note:

When the value of the program bank register (PCB) is equal to "FF_H", the vector area is also used as the vector area for INT #vct8 (#0 to #7). Caution must, therefore, be exercised when the area is used. (See Table 9.1-1 .)

● Example:

CALLV #15

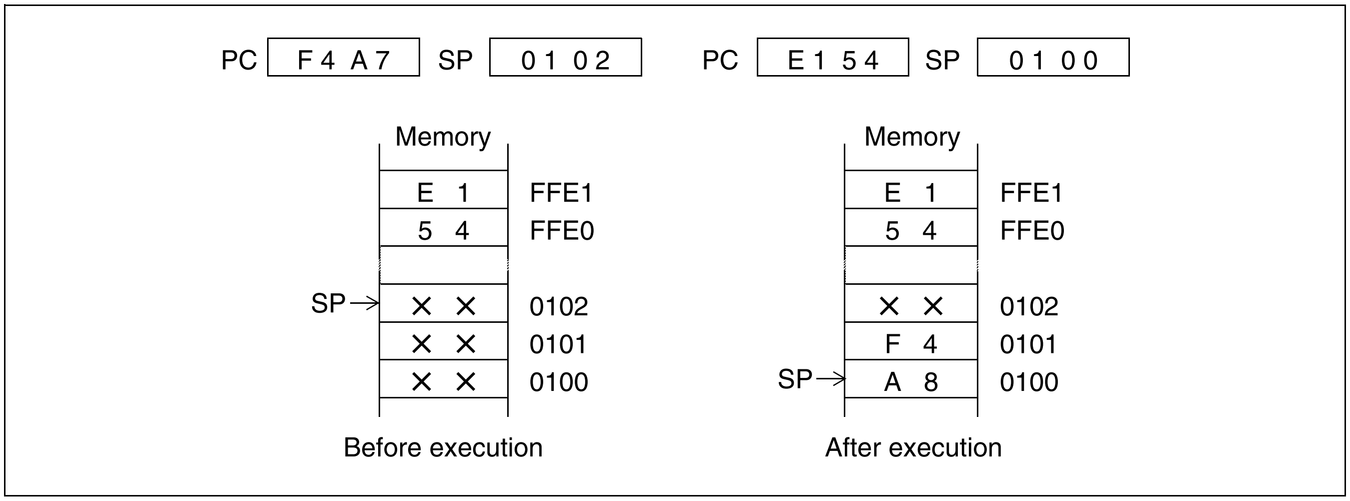


Table 9.1-1 CALLV Vector List

Instruction	Vector address L	Vector address H
CALLV #0	$XXFFFE_H$	$XXFFFF_H$
CALLV #1	$XXFFFC_H$	$XXFFFD_H$
CALLV #2	$XXFFFA_H$	$XXFFFB_H$
CALLV #3	$XXFFF8_H$	$XXFFF9_H$
CALLV #4	$XXFFF6_H$	$XXFFF7_H$
CALLV #5	$XXFFF4_H$	$XXFFF5_H$
CALLV #6	$XXFFF2_H$	$XXFFF3_H$
CALLV #7	$XXFFF0_H$	$XXFFF1_H$
CALLV #8	$XXFFEE_H$	$XXFFEF_H$
CALLV #9	$XXFFEC_H$	$XXFFED_H$
CALLV #10	$XXFFEA_H$	$XXFFEB_H$
CALLV #11	$XXFFE8_H$	$XXFFE9_H$
CALLV #12	$XXFFE6_H$	$XXFFE7_H$
CALLV #13	$XXFFE4_H$	$XXFFE5_H$
CALLV #14	$XXFFE2_H$	$XXFFE3_H$
CALLV #15	$XXFFE0_H$	$XXFFE1_H$

Note: XX is replaced by the value of the PCB register.

9.1.24 CBNE (Compare Byte Data and Branch if not equal)

Perform byte comparison on the first and second operands (8-bit immediate data) and cause a branch if the first and second operands are not equal. Control is transferred to the address equal to the address of the instruction following the CBNE instruction plus the word value resulting from sign-extending the third operand. A branch is not taken if the first and second operands are equal.
Note that, when the first operand is @PC + disp16, the operand address is equal to the "address of the location containing the machine instruction for the CBNE instruction + 4 + disp16", not the "address of the location containing the machine instruction for the instruction following the CBNE instruction 4 + disp16".

■ CBNE (Compare Byte Data and Branch if not equal)

● Assembler format:

CBNE A,#imm8,rel

CBNE ear,#imm8,rel

CBNE eam,#imm8,rel

● Operation:

(First operand)≠imm8 (Byte comparison) :(PC) ← (PC)+<Number of bytes>+rel

(First operand)=imm8 (Byte comparison) :(PC) ← (PC)+<Number of bytes>

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the compare operation result is "1", cleared otherwise.
Z:	Set when (First operand) = imm8, cleared otherwise.
V:	Set when an overflow has occurred as a result of the compare operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the compare operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

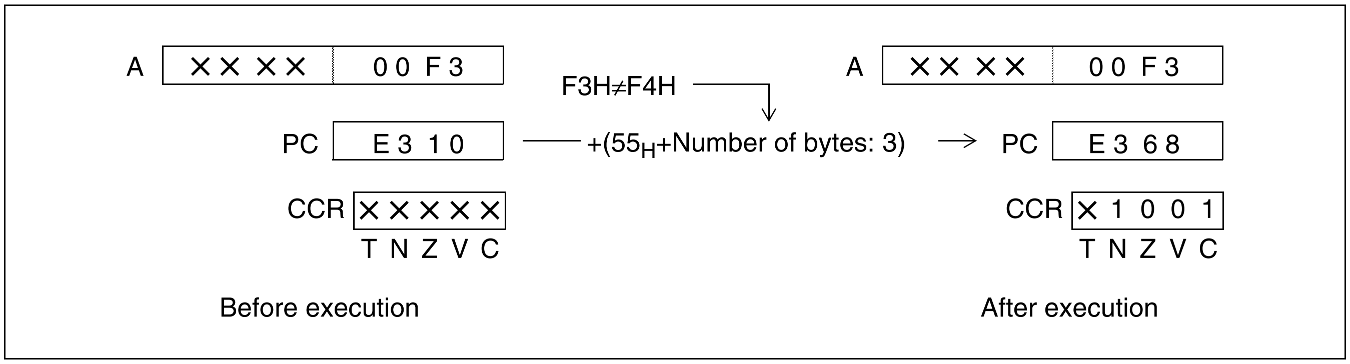
First operand	A	ear	eam *
Number of bytes	3	4	4+
Number of cycles	If a branch is taken: 5 If a branch is not taken: 4	If a branch is taken: 13 If a branch is not taken: 12	If a branch is taken: 7+(a) If a branch is not taken: 6+(a)
Correction value	0	0	(b)

* : @Rwj+ addressing cannot be used in eam. If such code is executed, +4 is added to the contents of Rwj.
For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

●Example:

CBNE A, #0F4H,55H

In this example, (First operand) ≠ Second operand (8-bit immediate data) is indicated.



9.1.25 CLRB (Clear Bit)

Clear the bit specified by bp to "0", in the memory location specified by the operand.

■ CLRB (Clear Bit)

- Assembler format:

  CLRB dir:bp

  CLRB io:bp

  CLRB addr16:bp

- Operation:

  (Operand) b ← 0 (Bit transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

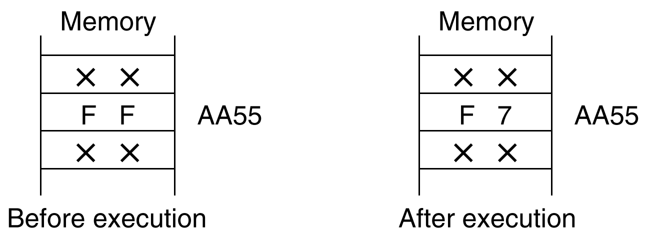
Operand	dir:bp	io:bp	ad16:bp
Number of bytes	3	3	4
Number of cycles	7	7	7
Correction value	2×(b)	2×(b)	2×(b)

For the explanation of (b) in the table, see Table 8.4-2 .

- Example:

  CLRB 0AA55H:3

  In this example, bit 3 of the data (FF_H) at address $AA55_H$ is cleared to "0".



9.1.26 CMP (Compare Byte Data of Destination and Source)

Compare the byte data specified by the first operand with that specified by the second operand and set the flag changes in the condition code register (CCR). The data specified by the first operand and that by the second operand are not changed.
If only the accumulator (A) is specified as an operand, AH and AL are compared.

■ CMP (Compare Byte Data of Destination and Source)

● Assembler format:

(1) CMP A,#imm8
CMP A,ear   CMP A,eam
(2) CMP A

● Operation:

(1)	(First operand)–(Second operand)	(Byte comparison)
(2)	(AH)–(AL)	(Byte comparison)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

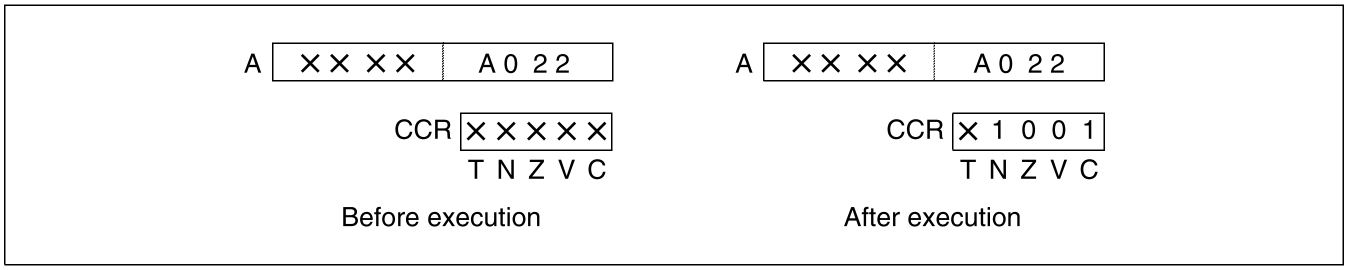
First operand	A	A	A	AH
Second operand	#im8	ear	eam	AL
Number of bytes	2	2	2+	1
Number of cycles	2	2	3+(a)	1
Correction value	0	0	(b)	0

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CMP A,#7FH

In this example, the least significant byte data (22_H) of A is compared with $7F_H$.



9.1.27 CMPL (Compare Long Word Data of Destination and Source)

Compare the long word data specified by the first operand with that specified by the second operand and set the result in the condition code register (CCR). The data specified by the first operand and that specified by the second are not changed.

■ CMPL (Compare Long Word Data of Destination and Source)

● Assembler format:

CMPL A,#imm32

CMPL A,ear CMPL A,eam

● Operation:

(First operand)–(Second operand) (Long word comparison)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

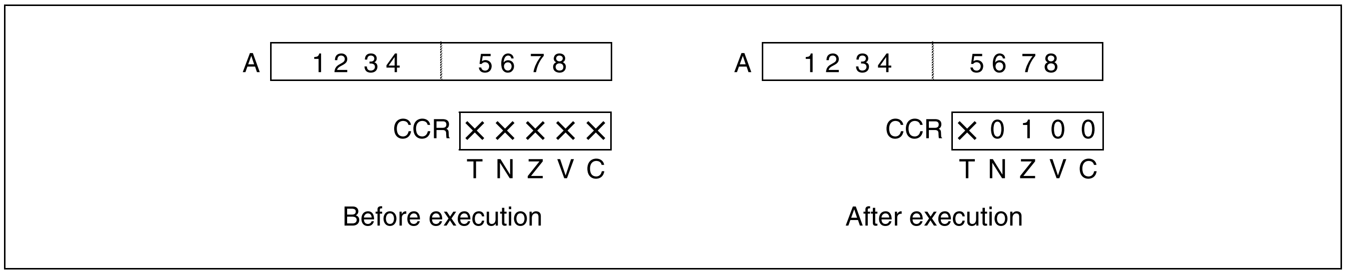
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A
Second operand	#i32	ear	eam
Number of bytes	5	2	2+
Number of cycles	3	6	7+(a)
Correction value	0	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CMPL A,#12345678H



9.1.28 CMPW (Compare Word Data of Destination and Source)

Compare the word data specified by the first operand with that specified by the second operand and set the result in the condition code register (CCR). The data specified by the first operand and that specified by the second operand are not changed.
If only A is specified as an operand, AH and AL are compared.

■ CMPW (Compare Word Data of Destination and Source)

● Assembler format:

(1) CMPW A,#imm16
CMPW A,ear CMPW A,eam
(2) CMPW A

● Operation:

(1)	(First operand)–(Second operand)	(Word comparison)
(2)	(AH)–(AL)	(Word comparison)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

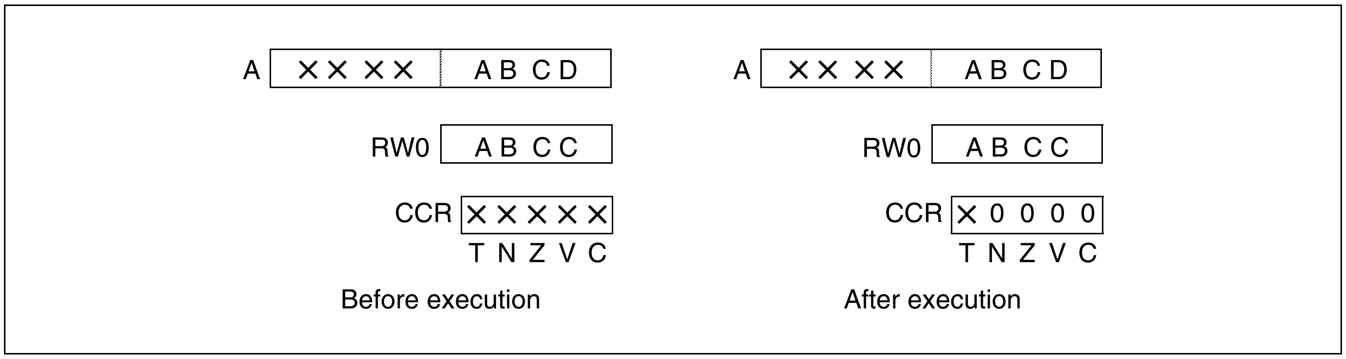
First operand	A	A	A	AH
Second operand	#i16	ear	eam	AL
Number of bytes	3	2	2+	1
Number of cycles	2	2	3+(a)	1
Correction value	0	0	(c)	0

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CMPW A,RW0

In this example, the low-order word data (ABCD$_H$) of A is compared with the data (ABCC$_H$) specified by RW0.



9.1.29 CWBNE (Compare Word Data and Branch if not Equal)

Perform word comparison on the first and second operands (16-bit immediate data) and cause a branch if the first and second operands are not equal. Control is transferred to the address equal to the address of the instruction following the CWBNE instruction plus the word data resulting from sign-extending the third operand. A branch is not taken if the first and second operands are equal.
Note that, when the first operand is @PC + disp16, the operand address is equal to the "address of the location containing the machine instruction for the CWBNE instruction + 4 + disp16", not the "address of the location containing the machine instruction for the instruction following the CWBNE instruction + disp16".

■ CWBNE (Compare Word Data and Branch if not Equal)

● Assembler format:

CWBNE A,#imm16,rel

CWBNE ear,#imm16,rel

CWBNE eam,#imm16,rel

● Operation:

(First operand)≠imm16 (Word comparison) : (PC) ← (PC)+<Number of bytes>+rel

(First operand)=imm16 (Word comparison) : (PC) ← (PC)+<Number of bytes>

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the comparison result is "1", cleared otherwise.
Z:	Set when (First operand) = #imm16, cleared otherwise.
V:	Set when an overflow has occurred as a result of the compare operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the compare operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

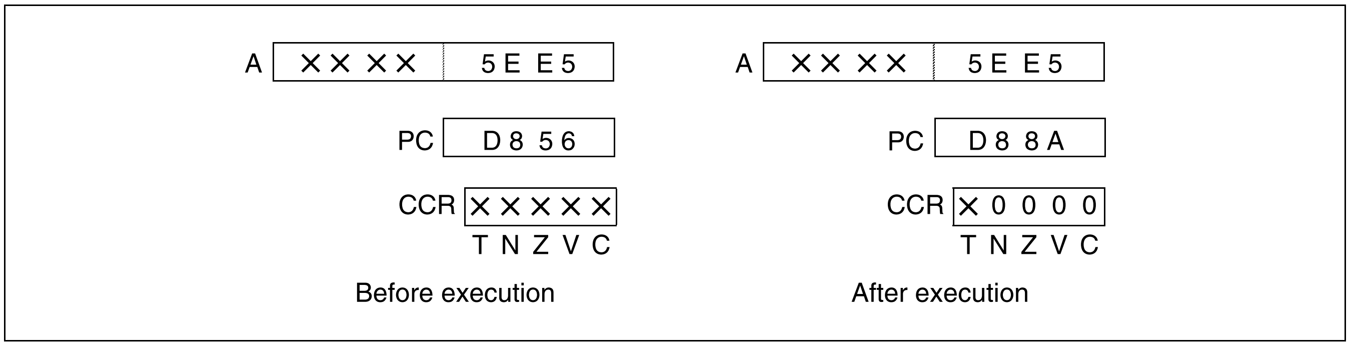
First operand	A	ear	eam *
Number of bytes	4	5	5+
Number of cycles	If a branch is taken: 5 If a branch is not taken: 4	If a branch is taken: 8 If a branch is not taken: 7	If a branch is taken: 7+(a) If a branch is not taken: 6+(a)
Correction value	0	0	(c)

*: @Rwj+ addressing cannot be used in eam. If such code is executed, +4 is added to the contents of Rwj.
For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

CWBNE A,#0E5E5H,30H

In this example, (First operand) ≠ imm16.



9.1.30 DBNZ (Decrement Byte Data and Branch if not "0")

Decrement the data specified by the first operand by one byte, and if the result is not equal to "0", a branch is generated. Control is transferred to the address equal to the address of the instruction following the DBNZ instruction plus the word data resulting from sign-extending the data specified by the second operand. If the decrement result is equal to "0", control is transferred to the next instruction.
Note that, when the first operand is @PC + disp16, the operand address is equal to the "address of the location containing the machine instruction for the DBNZ instruction + 4 + disp16", not the "address of the location containing the machine instruction for the instruction following the DBNZ instruction + disp16".

■ DBNZ (Decrement Byte Data and Branch if not "0")

● Assembler format:

DBNZ ear,rel DBNZ eam,rel

● Operation:

(ea) ← (ea)–1 (Byte subtraction)

if (ea) ≠ 0 : (PC) ← (PC)+<Number of bytes>+rel

if (ea) = 0 : (PC) ← (PC)+<Number of bytes>

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

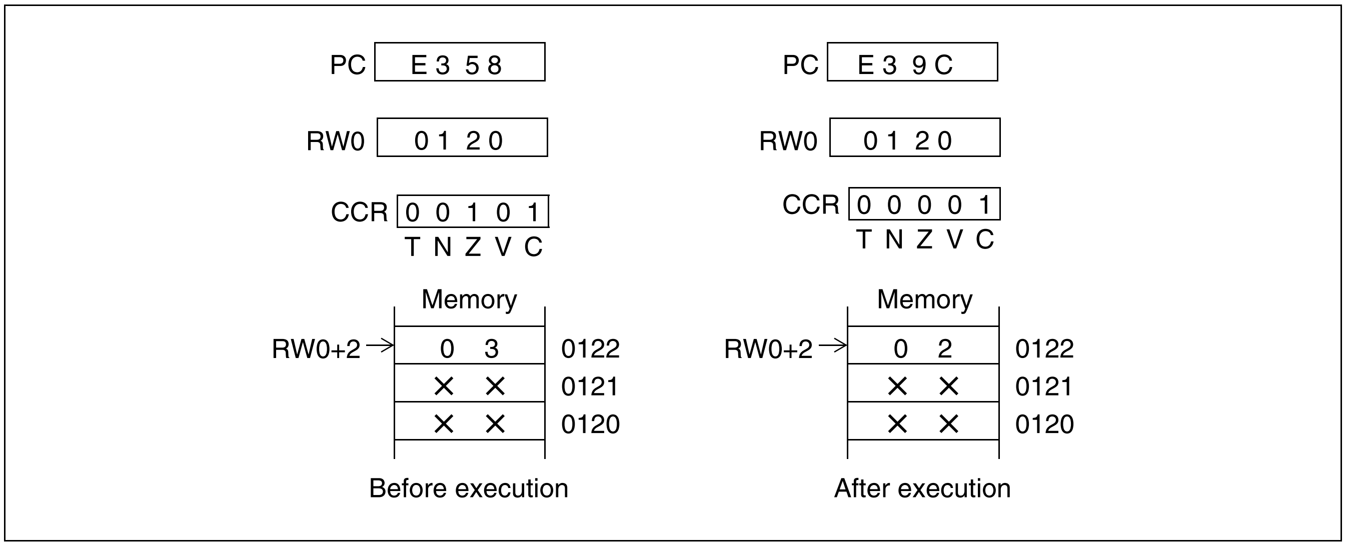
First operand	ear	eam
Number of bytes	3	3+
Number of cycles	If a branch is taken: 7 If a branch is not taken: 6	If a branch is taken: 8+(a) If a branch is not taken: 7+(a)
Correction value	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

DBNZ @RW0+2,40H

In this example, (First operand) - 1 ≠ 0 is shown.



9.1.31 DEC (Decrement Byte Data)

Decrement the byte data specified by the operand by one and store the result in the operand.

■ DEC (Decrement Byte Data)

● Assembler format:

DEC ear DEC eam

● Operation:

(ea) ← (ea)–1 (Byte subtraction)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

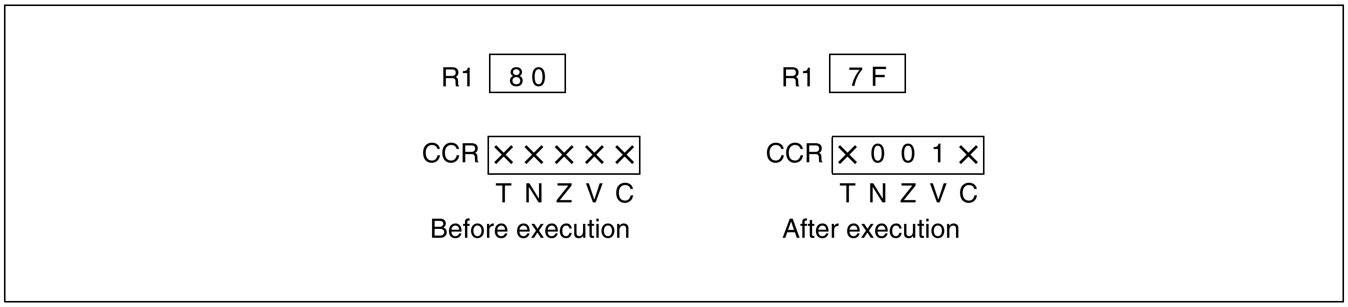
● Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	5+(a)
Correction value	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

DEC R1



9.1.32 DECL (Decrement Long Word Data)

Decrement the long word data specified by the operand by one and restore the result in the operand.

■ DECL (Decrement Long Word Data)

- Assembler format:

  DECL ear DECL eam

- Operation:

  (ea) ← (ea)–1 (Long word subtraction)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

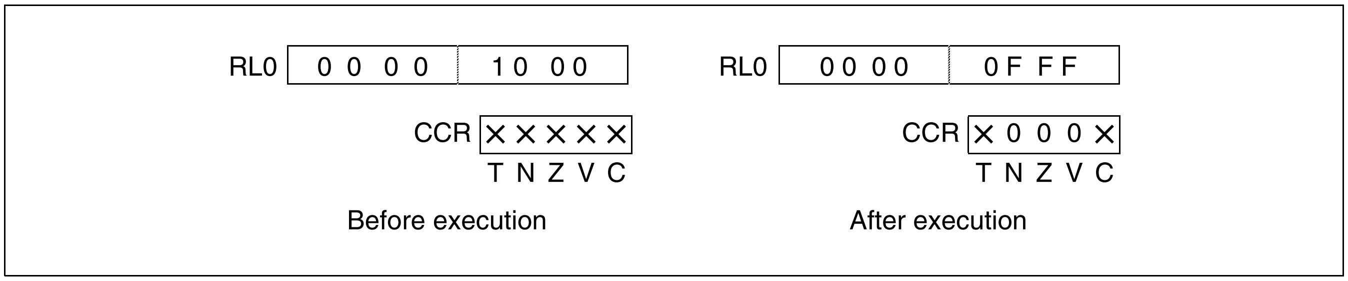
- Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	7	9+(a)
Correction value	0	2×(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

  DECL RL0



9.1.33 DECW (Decrement Word Data)

Decrement the word data specified by the operand by one and restore the result in the operand.

■ DECW (Decrement Word Data)

- Assembler format:

  DECW ear DECW eam

- Operation:

  (ea) ← (ea)–1 (Word subtraction)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

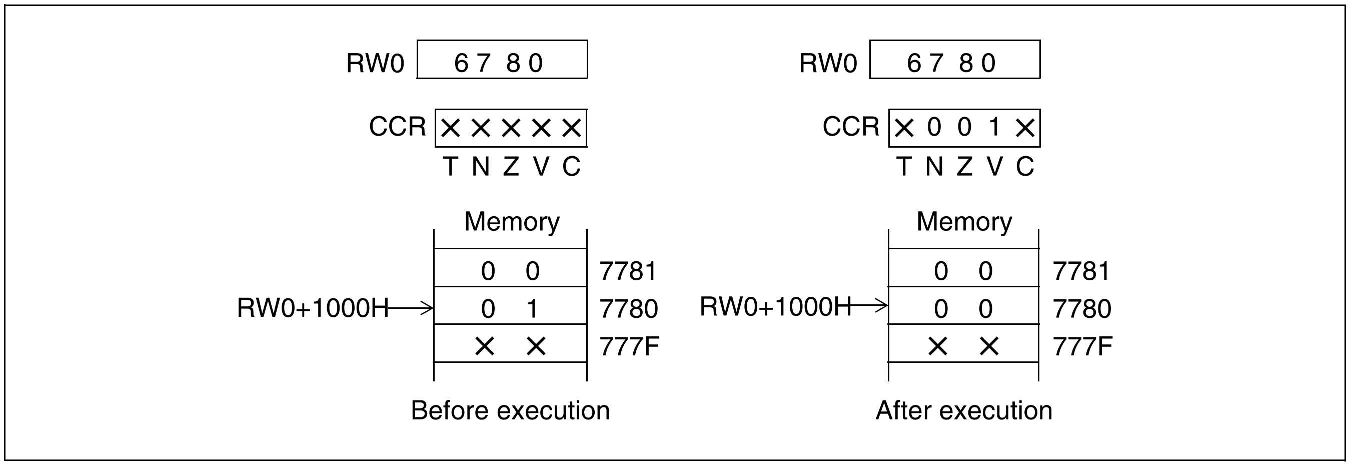
- Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	5+(a)
Correction value	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

DECW @RW0+1000H



9.1.34 DIV (Divide Word Data by Byte Data)

Divide the word data specified by the first operand by the byte data specified by the second operand and store the quotient (byte data) in the first operand and the remainder (byte data) in the second operand. The operation assumes that the values are signed ones.
If only A is specified by an operand, the word data of AH is divided by the byte data of AL and the quotient (byte data) is stored in AL and the remainder (byte data) in AH. The operation assumes that the values are signed ones.
If division by "0" occurs, the second operand or AL retains the value it had immediately before the instruction was executed. If an overflow occurs, the contents of AL are destroyed.

■ DIV (Divide Word Data by Byte Data)

● Assembler format:

(1) DIV A,ear DIV A,eam

(2) DIV A

● Operation:

(1) Word (A)/Byte (ea), Quotient → Byte (A), Remainder → Byte (ea)

(2) Word (AH)/Byte (AL), Quotient → Byte (AL), Remainder → Byte (AH)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	*	*

I, S, T, N, and Z:	Unchanged
V:	Set when an overflow has occurred as a result of the operation or the divisor is "0", cleared otherwise.
C:	Set when the divisor is "0", cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

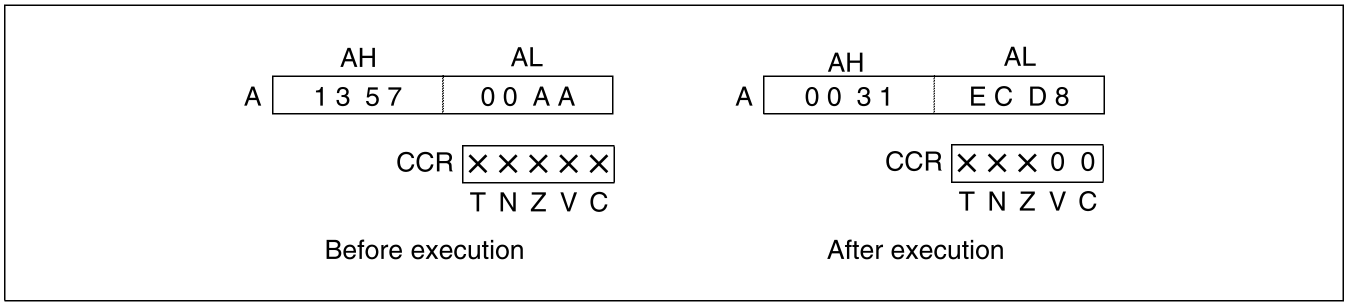
Second operand	DIV A	DIV A, ear	DIV A, eam
Number of bytes	2	2	2+
Number of cycles	Division by zero: 3 Overflow: 8 or 18 Normal termination: 18	Division by zero: 4 Overflow: 11 or 22 Normal termination: 23	Division by zero: 5+(a) Overflow: 12+(a) or 23+(a) Normal termination: 24+(a)
Correction value	0	0	*

*: (b) when division by zero or an overflow occurs; 2 × (b) when the instruction terminated normally.

For the explanation of (a) in the table and (b) in "*", see Table 8.4-1 and Table 8.4-2 .

● Example:

DIVA



9.1.35 DIVW (Divide Long Word Data by Word Data)

Divide the long word data specified by the first operand (A) by the word data specified by the second operand and store the quotient (word data) in A and the remainder (word data) in the second operand. The operation assumes that the values are signed ones. If division by "0" occurs, the second operand or AL retains the value it had immediately before the instruction was executed. If an overflow occurs, the contents of AL are destroyed.

■ DIVW (Divide Long Word Data by Word Data)

● Assembler format:

DIVW A,ear DIVW A,eam

● Operation:

Long word (A)/Word (ea), Quotient → Word (A), Remainder → Word (ea)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	*	*

I, S, T, N, and Z:	Unchanged
V:	Set when an overflow has occurred as a result of the operation or the divisor is "0", cleared otherwise.
C:	Set when the divisor is "0", cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

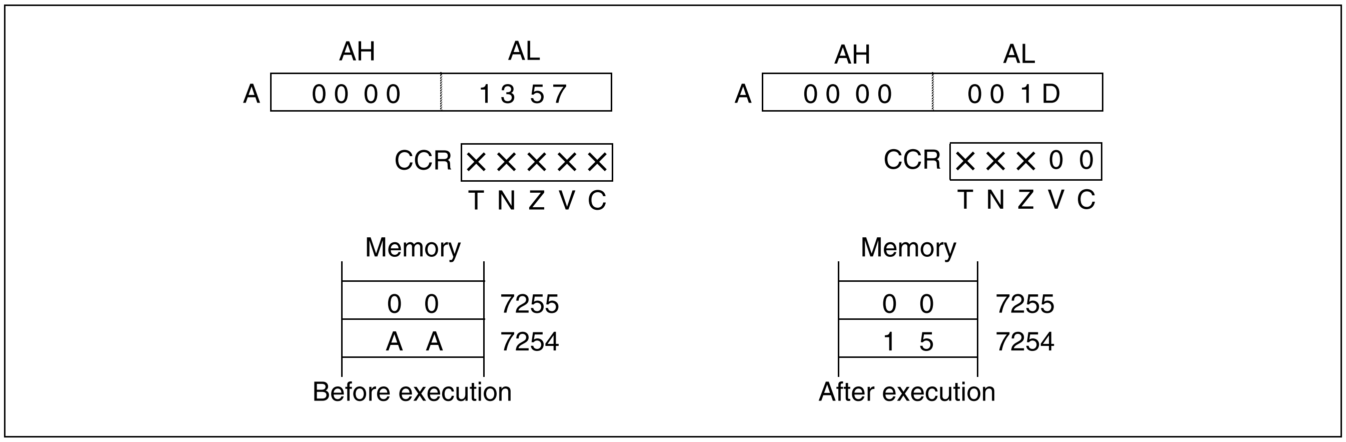
Second operand	DIVW A, ear		DIVW A, eam	
Sign of the dividend	Plus	Minus	Plus	Minus
Number of bytes	2	2	2+	2+
Number of cycles	Division by zero: 4 Overflow: 11 or 30 Normal termination: 31	Division by zero: 4 Overflow: 12 or 31 Normal termination: 32	Division by zero: 5+(a) Overflow: 12+(a) or 31+(a) Normal termination: 32+(a)	Division by zero: 5+(a) Overflow: 12+(a) or 32+(a) Normal termination: 33+(a)
Correction value	0	0	*	*

*: (c) when division by zero or an overflow occurs; 2 × (c) when the instruction terminated normally.

For the explanation of (a) in the table and (c) in "*", see Table 8.4-1 and Table 8.4-2 .

● Example:

DIVW A,7254H



9.1.36 DIVU (Divide unsigned Word Data by unsigned Byte Data)

Divide the word data specified by the first operand by the byte data specified by the second operand and store the quotient (byte data) in the first operand and the remainder (byte data) in the second operand. The operation assumes that the values are unsigned ones.
If only A is specified by an operand, the word data of AH is divided by the byte data of AL and the quotient (byte data) is stored in AL and the remainder (byte data) in AH. The operation assumes that the values are unsigned ones.
If an overflow or division by "0" occurs, the second operand or AL retains the value it had immediately before the instruction was executed.

■ DIVU (Divide unsigned Word Data by unsigned Byte Data)

● Assembler format:

(1) DIVU A,ear DIVU A,eam
(2) DIVU A

● Operation:

(1) Word (A)/Byte (ea), Quotient → Byte (A), Remainder → Byte (ea)
(2) Word (AH)/Byte (AL), Quotient → Byte (AL), Remainder → Byte (AH)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	*	*

I, S, T, N, and Z:	Unchanged
V:	Set when an overflow has occurred as a result of the operation or the divisor is "0", cleared otherwise.
C:	Set when the divisor is "0", cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

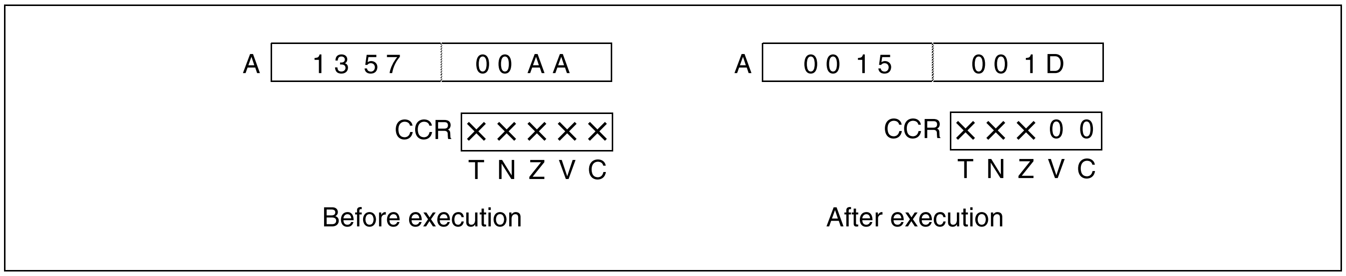
Assembler format	DIVU A	DIVU A, ear	DIVU A, eam
Number of bytes	1	2	2+
Number of cycles	Division by zero: 3 Overflow: 7 Normal termination: 15	Division by zero: 4 Overflow: 8 Normal termination: 16	Division by zero: 6+(a) Overflow: 9+(a) Normal termination: 19+(a)
Correction value	0	0	*

*: (b) when division by zero or an overflow occurs; 2 × (b) when the instruction terminated normally.

For the explanation of (a) in the table and (b) in "*", see Table 8.4-1 and Table 8.4-2 .

● Example:

DIVU A



9.1.37 DIVUW (Divide unsigned Long Word Data by unsigned Word Data)

Divide the long word data specified by the first operand (A) by the word data specified by the second operand and store the quotient (word data) in A and the remainder (word data) in the second operand. The operation assumes that the values are unsigned ones.
If an overflow or division by "0" occurs, the second operand or AL retains the value it had immediately before the instruction was executed.

■ DIVUW (Divide unsigned Long Word Data by unsigned Word Data)

● Assembler format:

DIVUW A,ear DIVUW A,eam

● Operation:

Long Word (A)/Word (ea), Quotient $\rightarrow$ Word (A), Remainder $\rightarrow$ Word (ea)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	*	*

I, S, T, N, and Z:	Unchanged
V:	Set when an overflow has occurred as a result of the operation or the divisor is "0", cleared otherwise.
C:	Set when the divisor is "0", cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

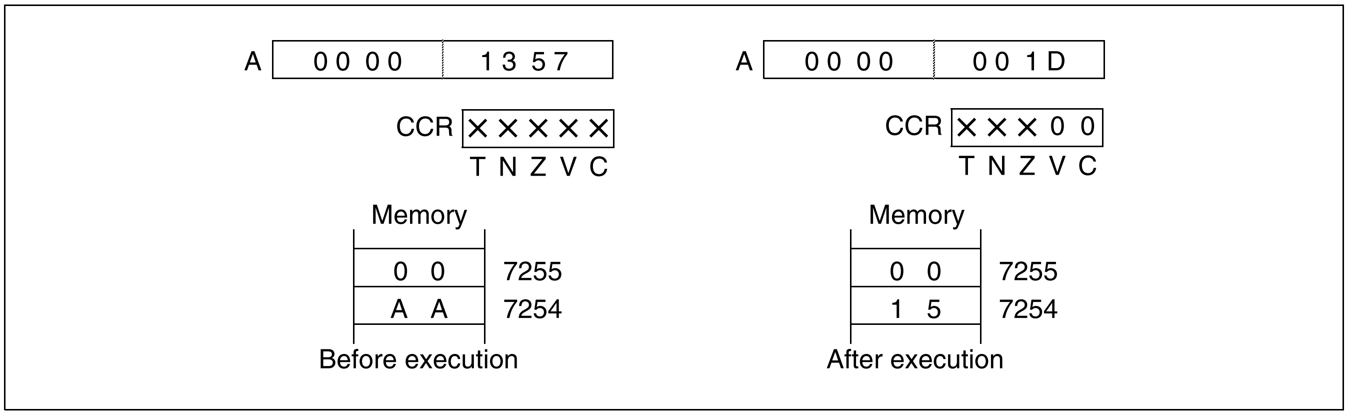
Assembler format	DIVU A, ear	DIVU A, eam
Number of bytes	2	2+
Number of cycles	Division by zero: 4 Overflow: 7 Normal termination: 22	Division by zero: 6+(a) Overflow: 8+(a) Normal termination: 26+(a)
Correction value	0	*

*: (c) when division by zero or an overflow occurs; 2 × (c) when the instruction terminated normally.

For the explanation of (a) in the table and (c) in "*", see Table 8.4-1 and Table 8.4-2 .

● Example:

DIVUW A,7254H



9.1.38 DWBNZ (Decrement Word Data and Branch if not Zero)

Decrement the data specified by the first operand by one word, and if the result is not equal to "0", cause a branch. Control is transferred to the address equal to the address of the instruction following the DWBNZ instruction plus the word data resulting from sign-extending the data specified by the second operand. If the decrement result is equal to "0", control is transferred to the instruction following the DWBNZ instruction. When the first operand is @PC + disp16, the operand address is equal to the "address of the location containing the machine instruction for the DWBNZ instruction + 4 + disp16", not the "address of the location containing the machine instruction for the instruction following the DWBNZ instruction + disp16".

■ DWBNZ (Decrement Word Data and Branch if not Zero)

● Assembler format:

DWBNZ ear,rel DWBNZ eam,rel

● Operation:

(First operand) ← (First operand)–1 (Word subtraction)

When (First operand)≠0, (PC) ← (PC)+<Number of bytes>+second operand

(PC) ← (PC)+<Number of bytes>

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

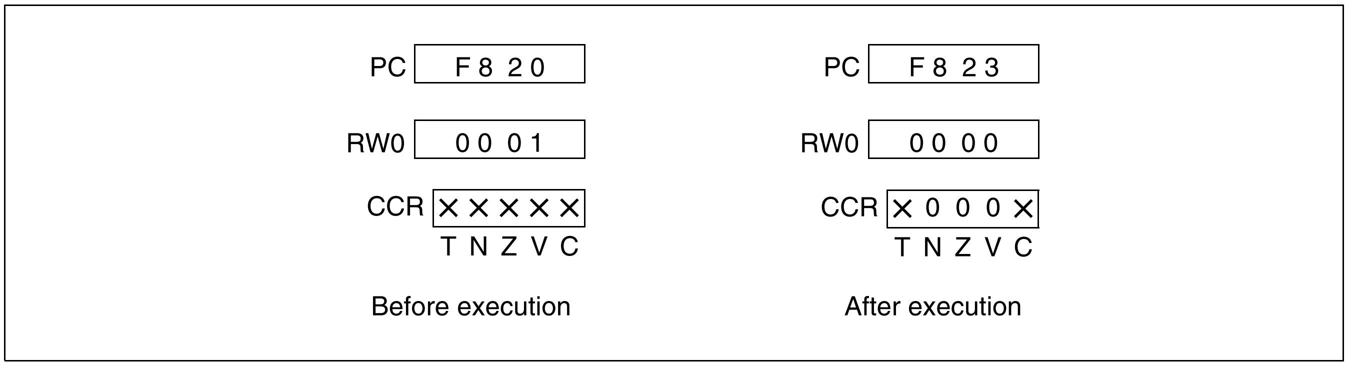
Second operand	ear	eam
Number of bytes	3	3+
Number of cycles	If a branch is taken: 7 If a branch is not taken: 6	If a branch is taken: 8+(a) If a branch is not taken: 7+(a)
Correction value	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

DWBNZ RW0,30H

In this example, (First operand) – 1 = 0.



9.1.39 EXT (Sign Extend from Byte Data to Word Data)

Extend the least significant byte data of A to word data as a signed binary number.

■ EXT (Sign Extend from Byte Data to Word Data)

- Assembler format:

EXT

- Operation:

When bit 7 of A=0, bits 8 to 15 of A ← 00_H

When bit 7 of A≠0, bits 8 to 15 of A ← FF_H

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the sign-extended data is "1", cleared otherwise.
Z:	Set when the sign-extended data is "0", cleared otherwise.
V and C:	Unchanged

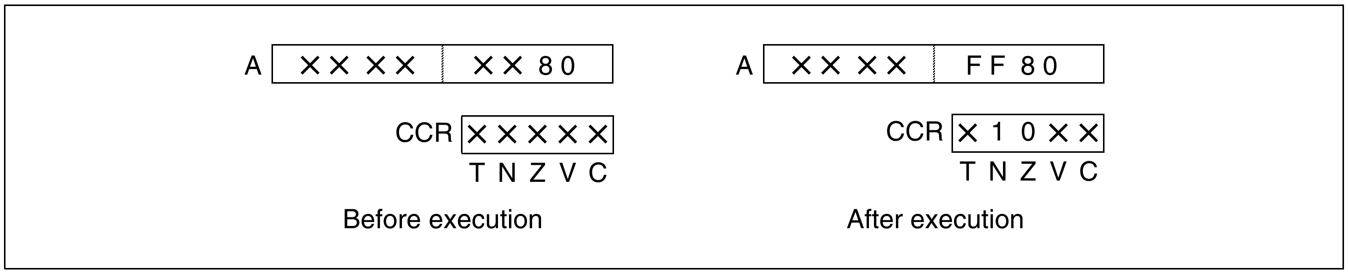
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	1
Correction value:	0

- Example:

EXT

In this example, the most significant bit of the least significant byte data ("80_H") for A is equal to "1", and "FF_H" is set in bits 8 to 15 of A to extend the byte data.



9.1.40 EXTW (Sign Extend from Word Data to Long Word Data)

Extend the low-order word data of A to long word data as a signed binary number.

■ EXTW (Sign Extend from Word Data to Long Word Data)

● Assembler format:

EXTW

● Operation:

When bit15 of A=0, bits 16 to 31 of A ← 0000_H

When bit15 of A≠0, bits 16 to 31 of A ← $FFFF_H$

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the sign-extended data is "1", cleared otherwise.
Z:	Set when the sign-extended data is "0", cleared otherwise.
V and C:	Unchanged

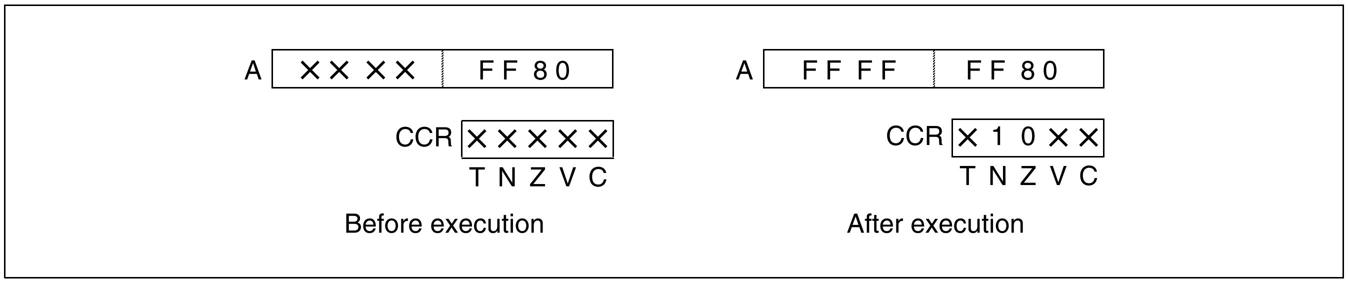
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

EXTW

In this example, the most significant bit of the low-order word data ("$FF80_H$") for A is equal to "1", and "$FFFF_H$" is set in bits 16 to 31 of A to extend the low-order word data.



9.1.41 FILS (Fill String Byte)

Transfer the contents of AL to the RW0-byte area that starts from the address whose high-order eight bits are specified by the bank register specified by <bank> and whose low-order 16 bits are specified by the contents of AH.
If RW0 is equal to "0", transfer is not performed. If an interrupt occurs during the execution of the instruction, the execution of the instruction is suspended. After the interrupt has been handled, the execution of the instruction is resumed.
Four types of registers PCB, DTB, ADB, and SPB can be specified by <bank>. If <bank> is omitted, DTB is assumed.

■ FILS (Fill String Byte)

● Assembler format:

FILS [I] [<bank>]

● Operation:

While RW0 ≠ 0, the following operation is repeated:
((AH)) ← (AL) (Byte transfer), (AH) ← (AH)+1,
(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

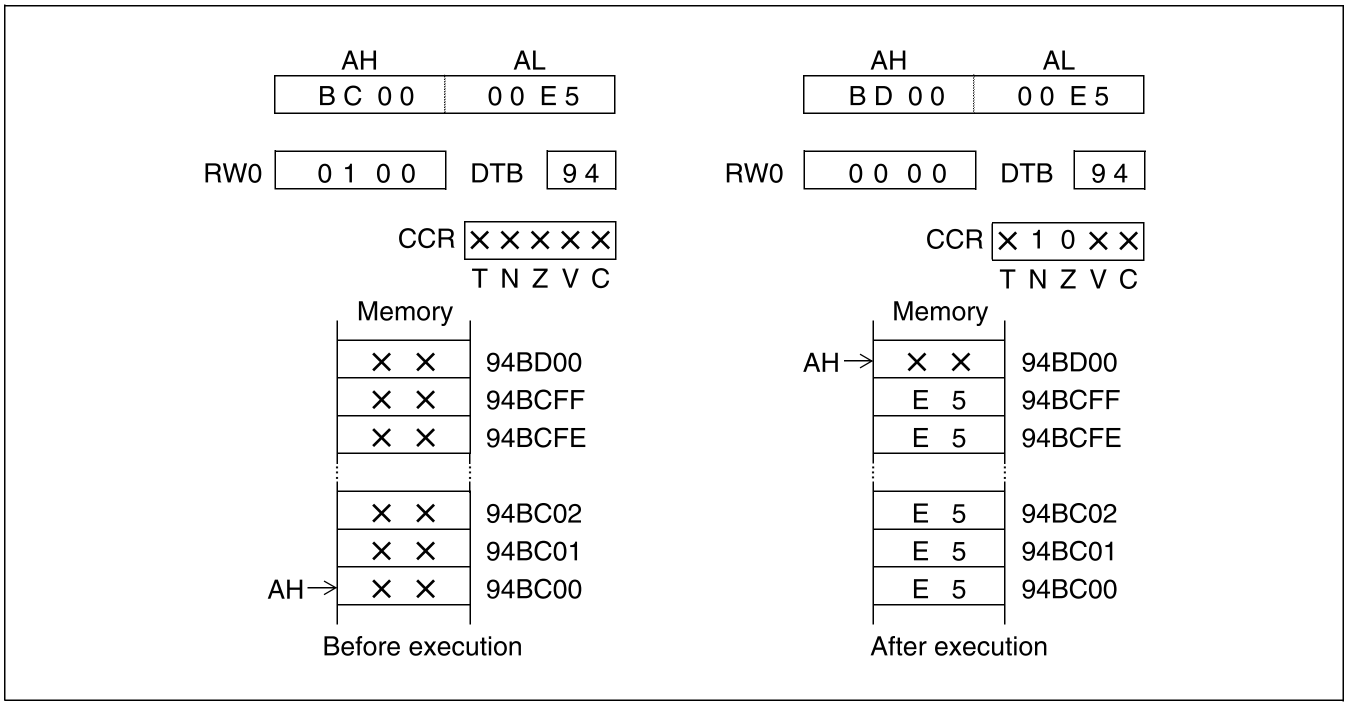
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6×(RW0)+6
Correction value:	(b)×(RW0)

For the explanation of (b), see Table 8.4-2 .

● Example:

FILS



9.1.42 FILSW (Fill String Word)

Transfer the contents of AL to the RW0-word area that starts from the address whose high-order eight bits are specified by the bank register specified by <bank> and whose low-order 16 bits are specified by the contents of AH.
If RW0 is equal to "0", transfer is not performed. If an interrupt occurs during the execution of the instruction, the execution of the instruction is suspended. After the interrupt has been handled, the execution of the instruction is resumed.
Four types of registers PCB, DTB, ADB, and SPB can be specified by <bank>. If <bank> is omitted, DTB is assumed.

■ FILSW (Fill String Word)

● Assembler format:

FILSW [I]   [<bank>]

● Operation:

While RW0 ≠ 0, the following operation is repeated:
((AH)) ← (AL) (Word transfer), (AH) ← (AH)+2,
(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

Number of bytes: 2

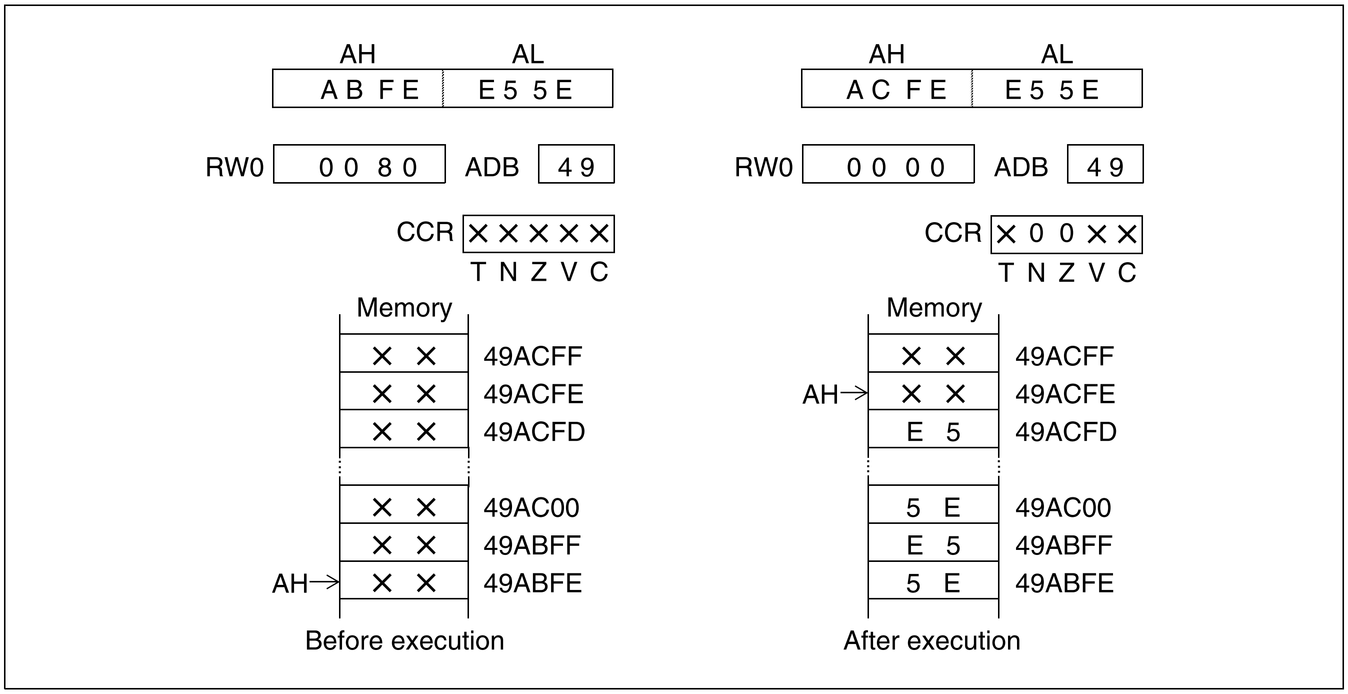
Number of cycles: 6×(RW0)+6

Correction value: (c)×(RW0)

For the explanation of (c), see Table 8.4-2 .

● Example:

FILSW ADB



9.1.43 INC (Increment Byte Data (Address Specification))

Increment the byte data specified by the operand by one and restore the result in the operand.

■ INC (Increment Byte Data (Address Specification))

- Assembler format:

INC ear INC eam

- Operation:

(Operand) ← (Operand)+1 (Byte increment)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

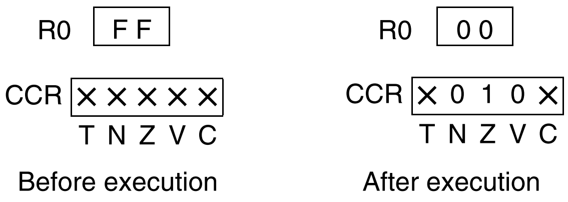
- Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	5+(a)
Correction value	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

INC R0



9.1.44 INCL (Increment Long Word Data)

Increment the long word data specified by the operand by one and restore the result in the operand.

■ INCL (Increment Long Word Data)

- Assembler format:

INCL ear INCL eam

- Operation:

(Operand) ← (Operand)+1 (Long word increment)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

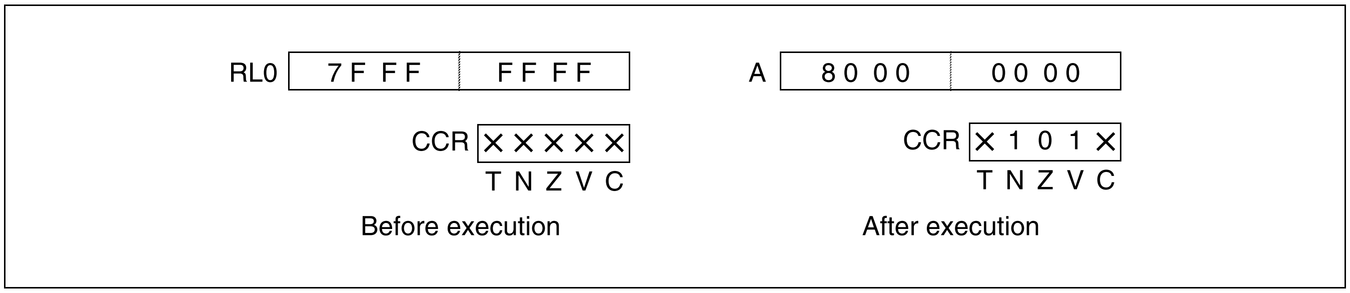
- Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	7	9+(a)
Correction value	0	2×(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

INCL RL0



9.1.45 INCW (Increment Word Data)

Increment the word data specified by the operand by one and restore the result in the operand.

■ INCW (Increment Word Data)

● Assembler format:

INCW ear INCW eam

● Operation:

(Operand) ← (Operand)+1 (Word increment)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Unchanged

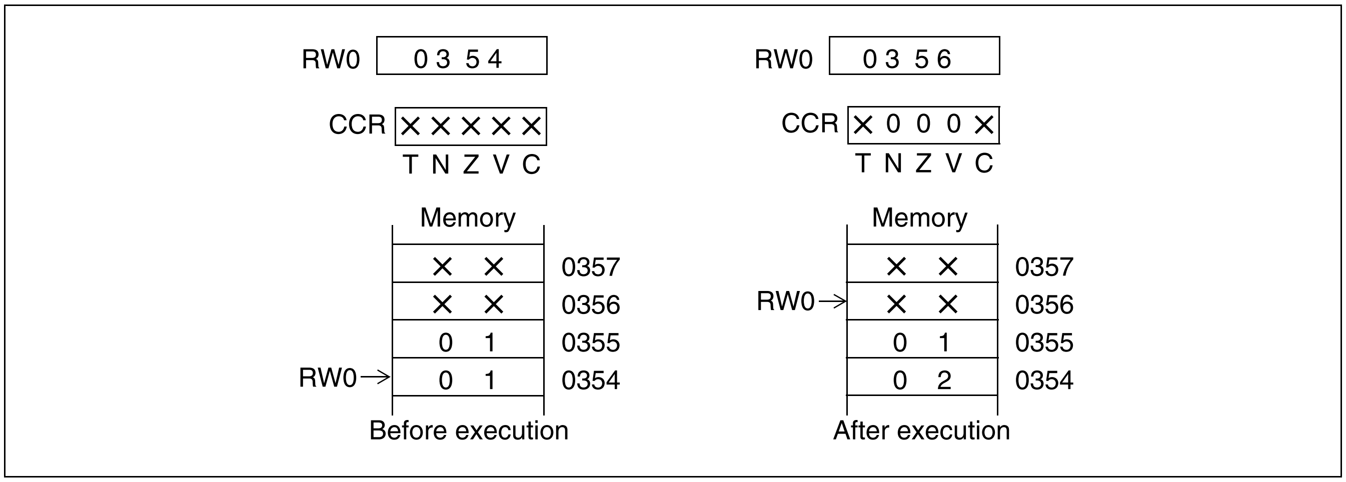
● Number of bytes, Number of cycles, and Correction value:

Operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	5+(a)
Correction value	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

INCW @RW0+



9.1.46 INT (Software Interrupt)

Cause a branch to the interrupt handling routine at the specified address in the bank $0FF_H$. By executing the RETI instruction in the interrupt handling routine to which control has been transferred, control returns to the instruction following this instruction.

■ INT (Software Interrupt)

● Assembler format:

INT addr16

● Operation:

(SSP) ← (SSP)–2, ((SSP)) ← (AH), (SSP) ← (SSP)–2, ((SSP)) ← (AL)

(SSP) ← (SSP)–2, ((SSP)) ← (DPR) : (ADB)(DPR and ADB are saved as a set, DPR as the high-order byte and ADB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (DTB) : (PCB) (DTB and PCB are saved as a set, DTB as the high-order byte and PCB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (PC+3), (SSP) ← (SSP)–2, ((SSP)) ← (PS)

(S) ← 1, (I) ← 0, (PCB) ← $0FF_H$, (PC) ← addr16

● CCR:

I	S	T	N	Z	V	C
R	S	–	–	–	–	–

I	:	Cleared
S	:	Set
T, N, Z, V, and C	:	Unchanged

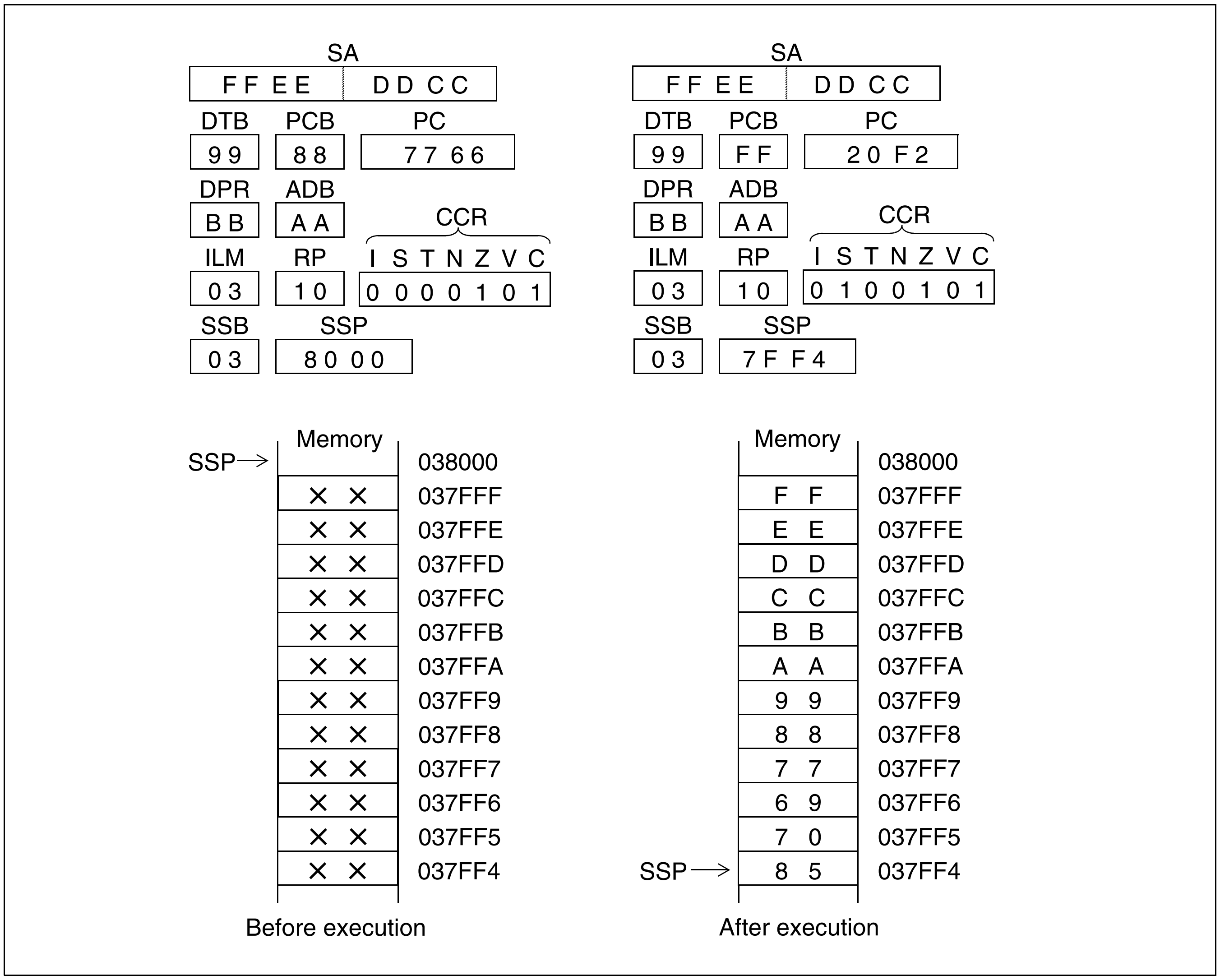
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	3
Number of cycles:	16
Correction value:	6×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

INT 020F2H



9.1.47 INT (Software Interrupt (Vector Specification))

Cause a branch to the interrupt handling routine pointed to by the interrupt vector specified by the operand.

■ INT (Software Interrupt (Vector Specification))

● Assembler format:

INT #vct8

● Operation:

(SSP) ← (SSP)–2, ((SSP)) ← (AH), (SSP) ← (SSP)–2, ((SSP) ← (AL)

(SSP) ← (SSP)–2, ((SSP)) ← (DPR) : (ADB)(DPR and ADB are saved as a set, DPR as the high-order byte and ADB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (DTB) : (PCB) (DTB and PCB are saved as a set, DTB as the high-order byte and PCB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (PC+2), (SSP) ← (SSP)–2, ((SSP)) ← (PS)

(S) ← 1, (I) ← 0, (PCB) ← Vector address (High-order byte)

(PC) ← Vector address (Low-order word)

● CCR:

I	S	T	N	Z	V	C
R	S	–	–	–	–	–

I	:	Cleared
S	:	Set
T, N, Z, V, and C	:	Unchanged

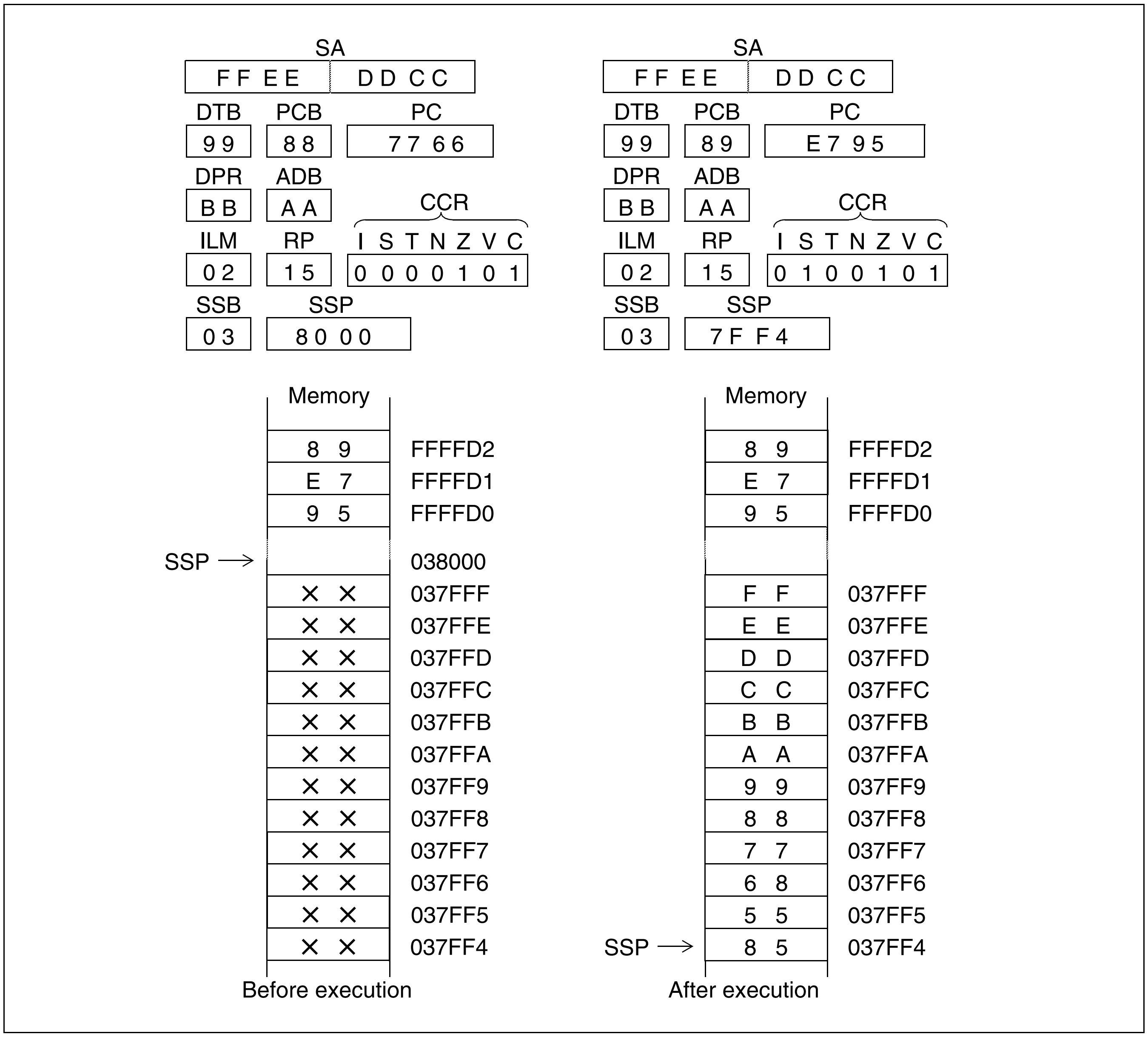
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	20
Correction value:	8×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

INT #11



9.1.48 INT9 (Software Interrupt)

Cause a branch to the interrupt handling routine pointed to by the vector.
By executing the RETI instruction in the interrupt handling routine to which control has been transferred, control returns to the instruction following this instruction.

■ INT9 (Software Interrupt)

● Assembler format:

INT9

● Operation:

(SSP) ← (SSP)–2, ((SSP)) ← (AH), (SSP) ← (SSP)–2, ((SSP)) ← (AL)

(SSP) ← (SSP)–2, ((SSP)) ← (DPR) : (ADB)(DPR and ADB are saved as a set, DPR as the high-order byte and ADB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (DTB) : (PCB) (DTB and PCB are saved as a set, DTB as the high-order byte and PCB as the low-order byte.)

(SSP) ← (SSP)–2, ((SSP)) ← (PC+1), (SSP) ← (SSP)–2, ((SSP)) ← (PS)

(S) ← 1, (I) ← 0, (PCB) ← Vector address (High-order byte)

(PC) ← Vector address (Low-order word)

● CCR:

I	S	T	N	Z	V	C
R	S	–	–	–	–	–

I	:	Cleared
S	:	Set
T, N, Z, V, and C	:	Unchanged

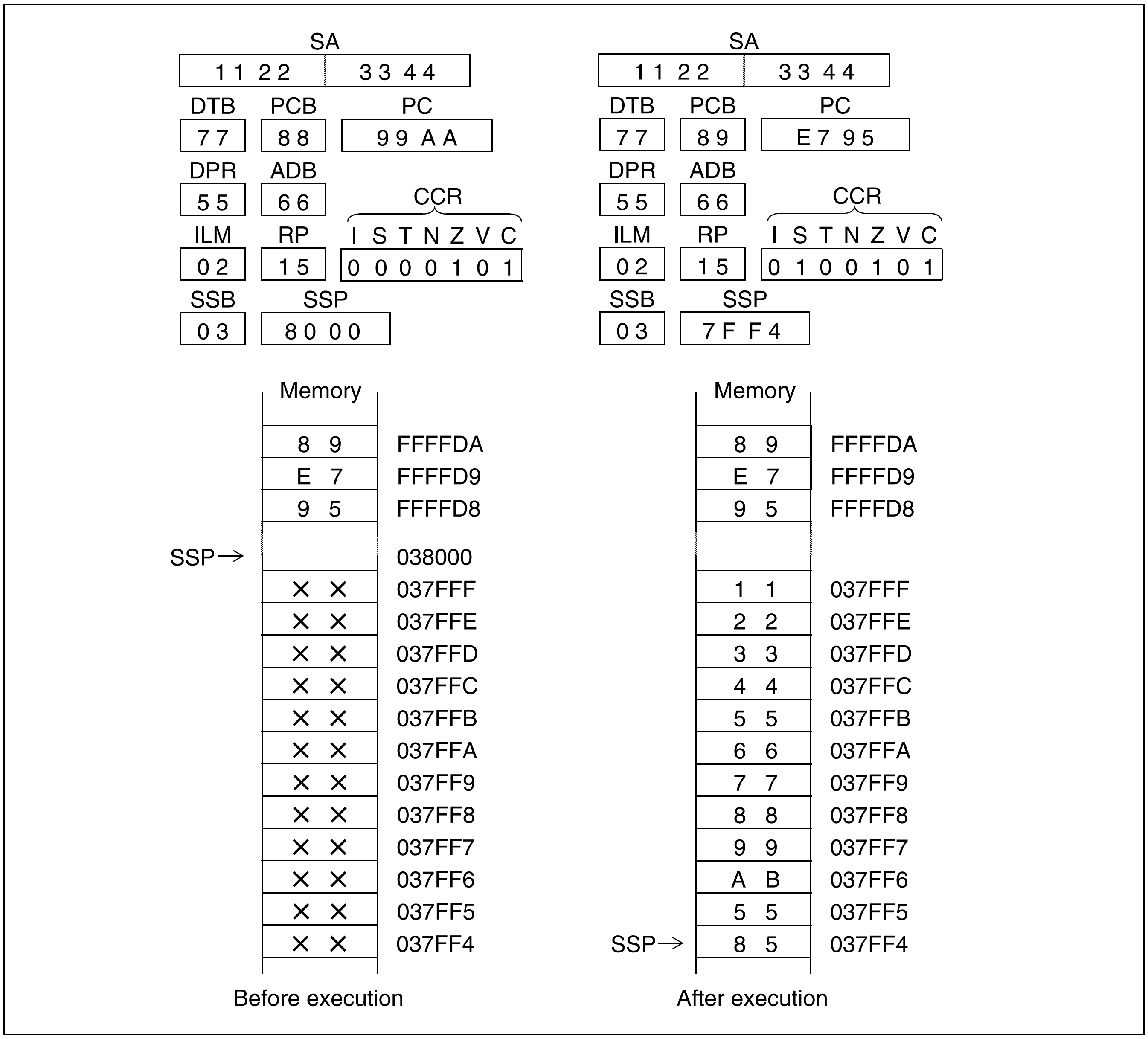
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	20
Correction value:	8×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

INT9



9.1.49 INTP (Software Interrupt)

Cause a branch to the interrupt handling routine at the 24-bit physical address specified by the operand. Any address in the entire 16MB space can be specified.
By executing the RETI instruction in the interrupt handling routine to which control has been transferred, control returns to the instruction following this instruction.

■ INTP (Software Interrupt)

● Assembler format:

INTP addr24

● Operation:

(SSP) ← (SSP)–2, ((SSP)) ← (AH), (SSP) ← (SSP)–2, ((SSP)) ← (AL)

(SSP) ← (SSP)–2, ((SSP)) ← (DPR) : (ADB)(DPR: High-order byte, ADB: Low-order byte)

(SSP) ← (SSP)–2, ((SSP)) ← (DTB) : (PCB) (DTB: High-order byte, PCB: Low-order byte)

(SSP) ← (SSP)–2, ((SSP)) ← (PC+4), (SSP) ← (SSP)–2, ((SSP)) ← (PS)

(S) ← 1, (I) ← 0, (PCB) ← Most significant byte of addr24,
(PC) ← Low-order word of addr24

● CCR:

I	S	T	N	Z	V	C
R	S	–	–	–	–	–

I :	Cleared
S:	Set
T, N, Z, V, and C:	Unchanged

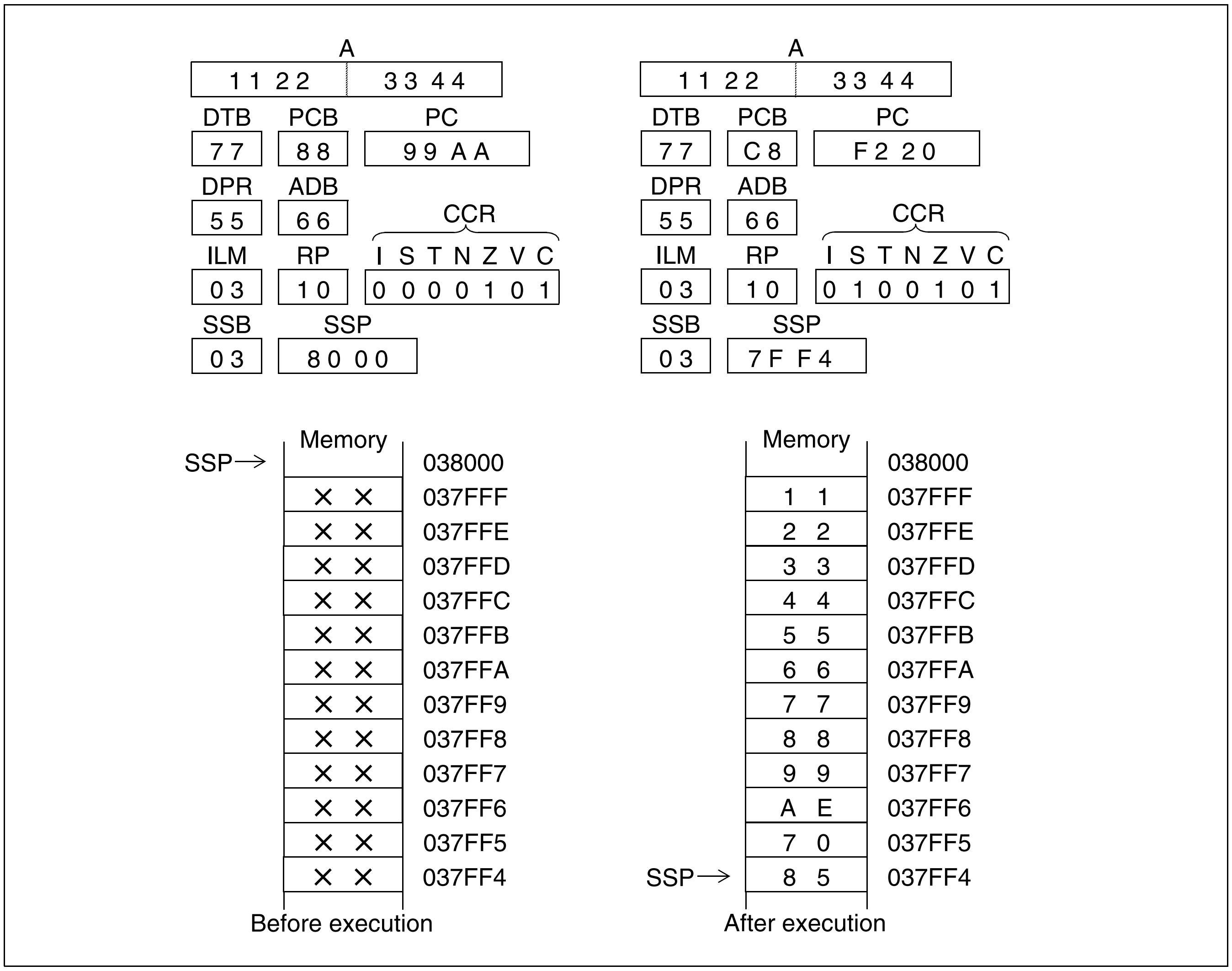
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	4
Number of cycles:	17
Correction value:	6×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

INTP 0C8F220H



9.1.50 JCTX (Jump Context)

Restore register contents or an address saved in memory.

■ JCTX (Jump Context)

● Assembler format:

JCTX @A

● Operation:

(temp)	← (AL)
(PS)	← ((temp)) : (temp) ← (temp)+2
(PC)	← ((temp)) : (temp) ← (temp)+2
(DTB), (PCB)	← ((temp)) : (temp) ← (temp)+2
(DPR), (ADB)	← ((temp)) : (temp) ← (temp)+2
(AL)	← ((temp)) : (temp) ← (temp)+2
(AH)	← ((temp))

● CCR:

I	S	T	N	Z	V	C
*	*	*	*	*	*	*

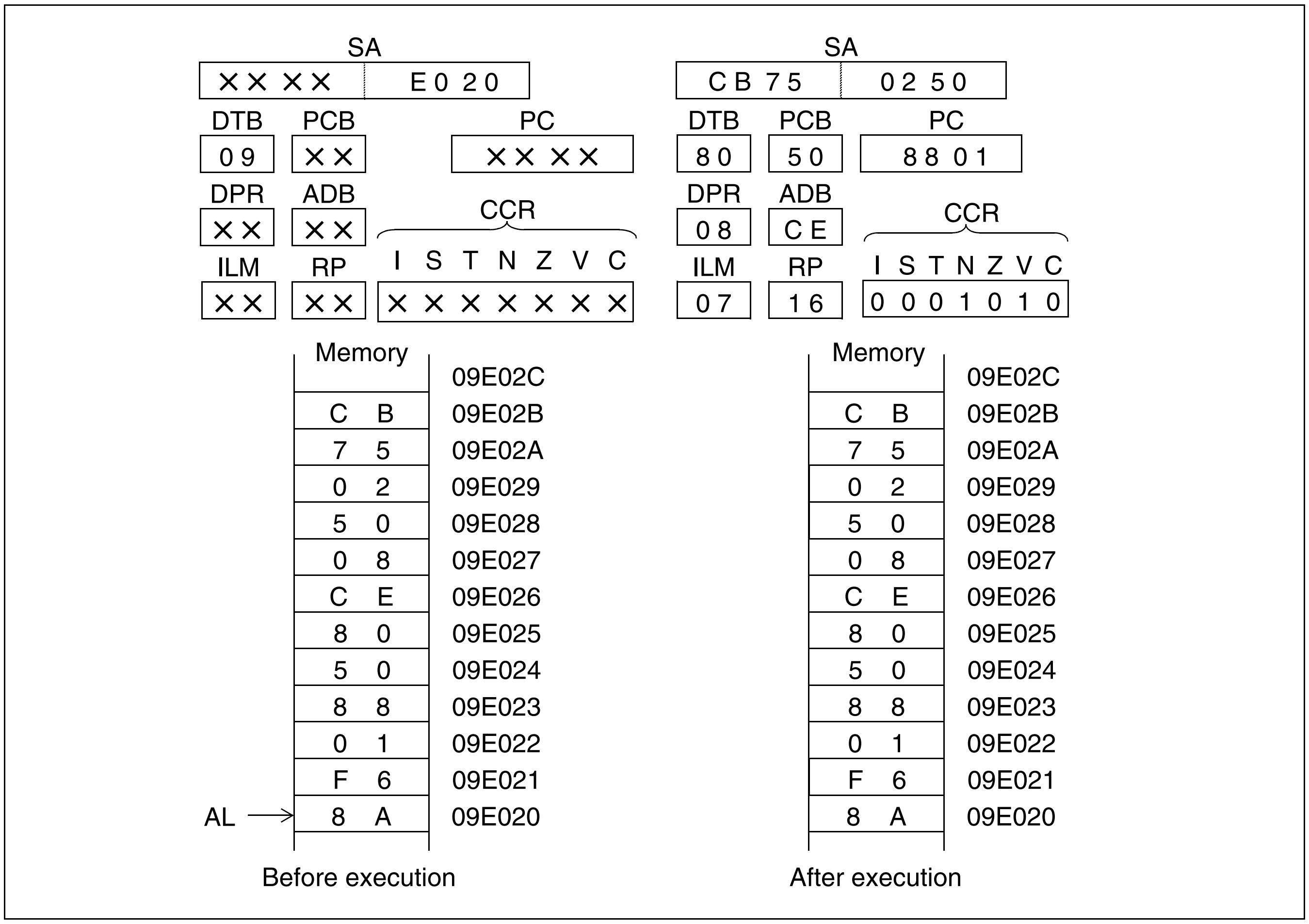
I : Stores bit 6 of the address indicated by AL.
S : Stores bit 5 of the address indicated by AL.
T : Stores bit 4 of the address indicated by AL.
N : Stores bit 3 of the address indicated by AL.
Z : Stores bit 2 of the address indicated by AL.
V : Stores bit 1 of the address indicated by AL.
C : Stores bit 0 of the address indicated by AL.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes: 1
Number of cycles: 14
Correction value: 6×(c)
For the explanation of (c), see Table 8.4-2 .

● Example:

JCTX @A



9.1.51 JMP (Jump Destination Address)

Read the word data from the address specified by the operand and cause a branch to the address specified by the word data.

■ JMP (Jump Destination Address)

- Assembler format:

JMP @A	JMP addr16
JMP @ear	JMP @eam

- Operation:

(PC) ← (Operand)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

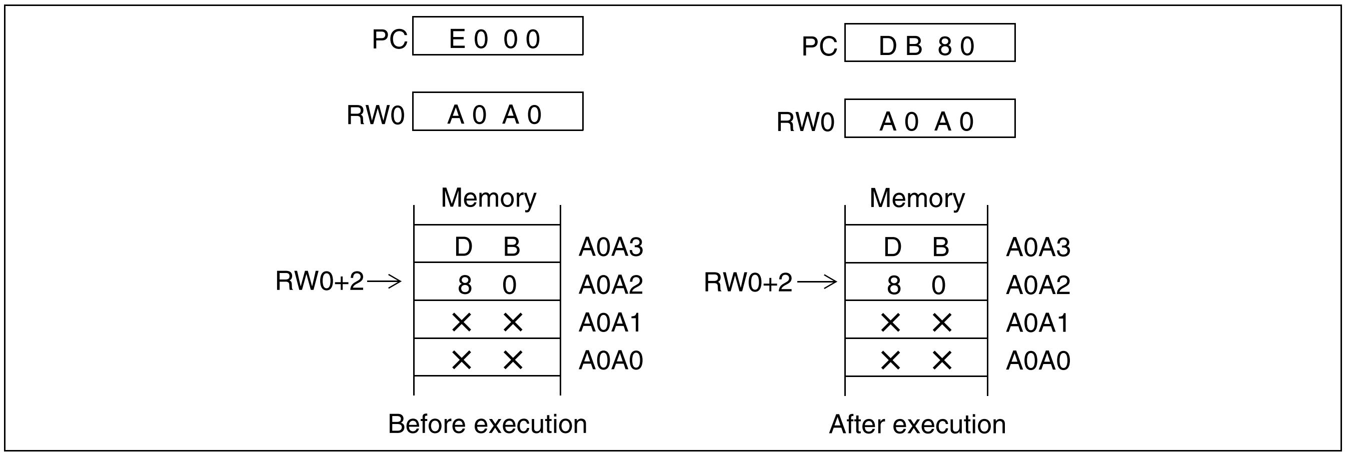
- Number of bytes, Number of cycles, and Correction value:

Operand	@A	@ear	@eam	ad16
Number of bytes	1	2	2+	3
Number of cycles	2	3	4+(a)	3
Correction value	0	0	(c)	0

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

JMP @@RW0+2



9.1.52 JMPP (Jump Destination Physical Address)

If the operand is addr24, this instruction causes a branch to the physical address specified by addr24.
If the operand is @ea, the instruction causes a branch to the physical address specified by the contents of the operand.

■ JMPP (Jump Destination Physical Address)

- Assembler format:

(1) JMPP addr24
(2) JMPP @ear JMPP @eam

- Operation:

(1):	(PC)	←	Low-order word of addr24	
	(PCB)	←	Most significant byte of addr24	
(2):	(PC)	←	(ea)	(Word transfer)
	(PCB)	←	(ea+2)	(Byte transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

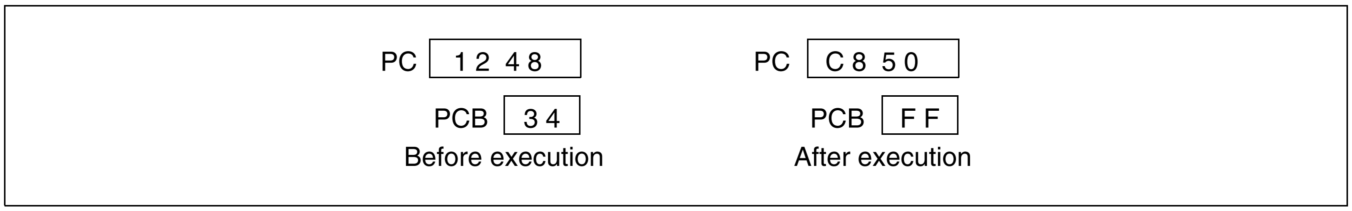
- Number of bytes, Number of cycles, and Correction value:

Operand	ad24	@ear	@eam
Number of bytes	4	2	2+
Number of cycles	4	5	6+(a)
Correction value	0	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

JMPP 0FFC850H



9.1.53 LINK (Link and create new stack frame)

Store the current value of the frame pointer (RW3) in a stack and set a new frame pointer. This allows an area for a new local variable to be reserved. This instruction is used before a function is called.

■ LINK (Link and create new stack frame)

- Assembler format:

LINK #imm8

- Operation:

(sp) ← (sp)–2 ; ((sp)) ← (RW3) ; (RW3) ← (sp) ; (sp) ← (sp)–imm8

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

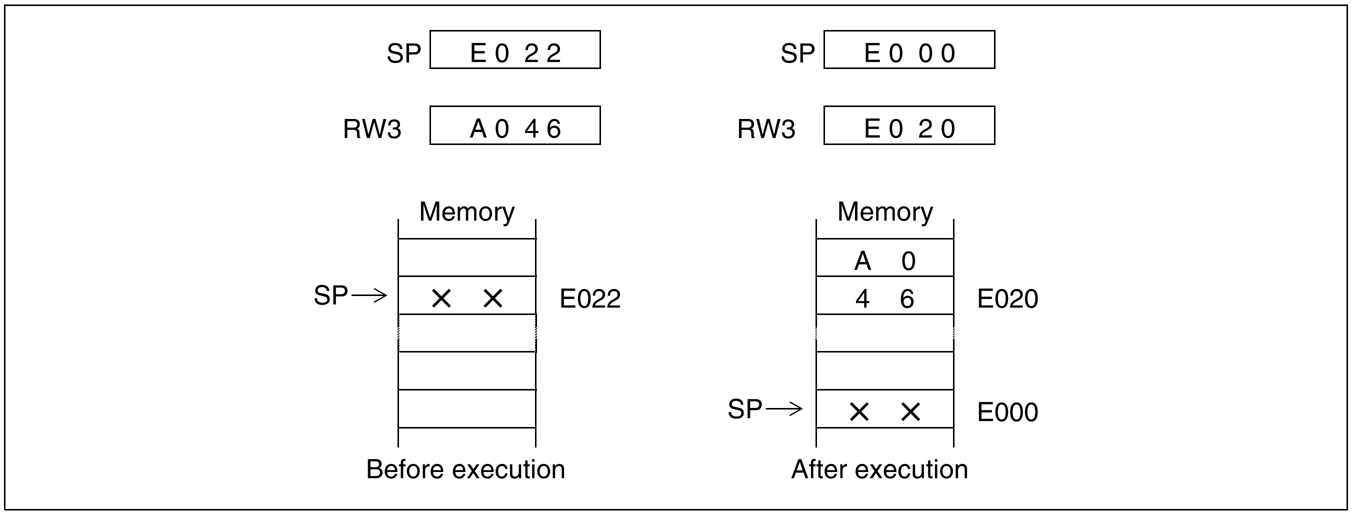
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6
Correction value:	(c)

For the explanation of (c), see Table 8.4-2 .

- Example:

LINK #20H



9.1.54 LSL (Logical Shift Byte Data of Accumulator to Left)

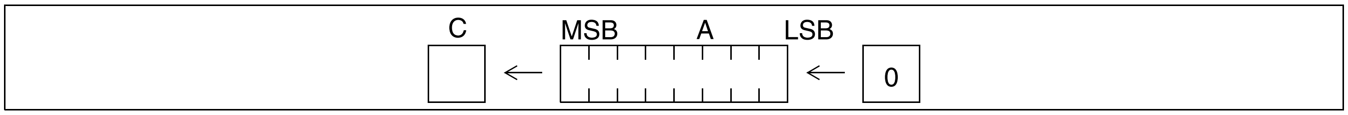
Shift the least significant byte data of the accumulator (A) to the left by the number of bits specified by the second operand. The least significant bit of A is set to "0". The bit last shifted out from the most significant bit of the least significant byte data for A is stored in the carry bit (C).

■ LSL (Logical Shift Byte Data of Accumulator to Left)

● Assembler format:

LSL A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

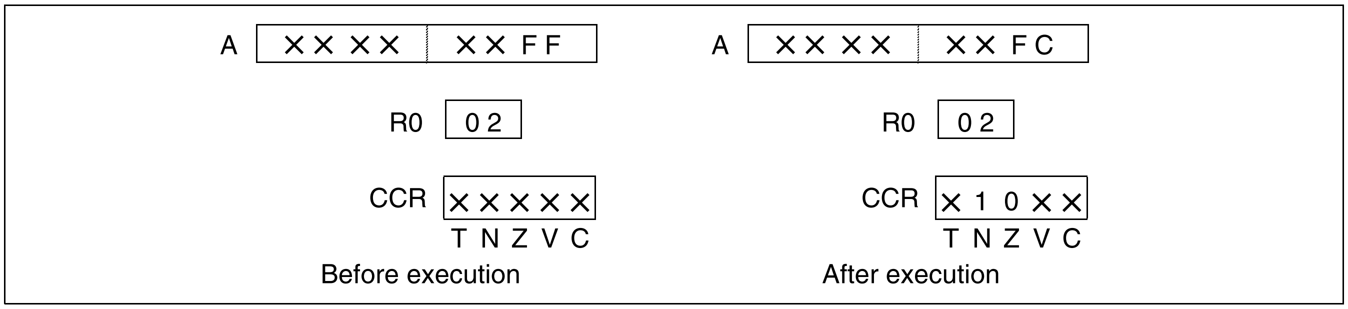
I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the MSB of A. Cleared when the shift amount is "0".

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0 ; otherwise, 5 + (R0)
Correction value:	0

● Example:

LSL A,R0



9.1.55 LSLL (Logical Shift Long Word Data of Accumulator to Left)

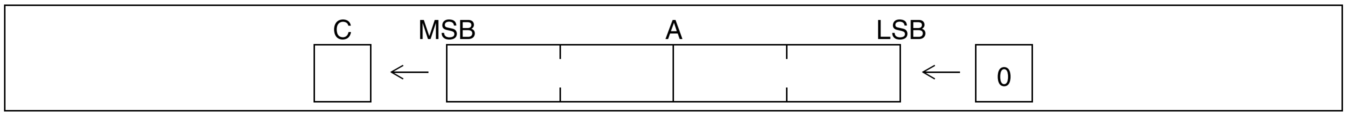
Shift the long word data of the accumulator (A) to the left by the number of bits specified by the second operand. The least significant bit of A is set to "0". The bit last shifted out from the most significant bit is stored in the carry bit (C).

■ LSLL (Logical Shift Long Word Data of Accumulator to Left)

● Assembler format:

LSLL A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

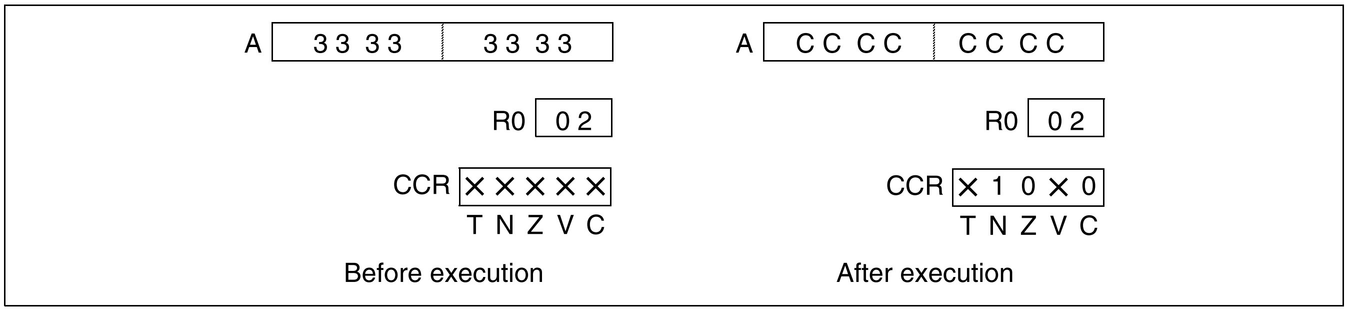
I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the MSB of A. Cleared when the shift amount is "0".

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 6 + (R0)
Correction value:	0

● Example:

LSLL A,R0



9.1.56 LSLW (Logical Shift Word Data of Accumulator to Left)

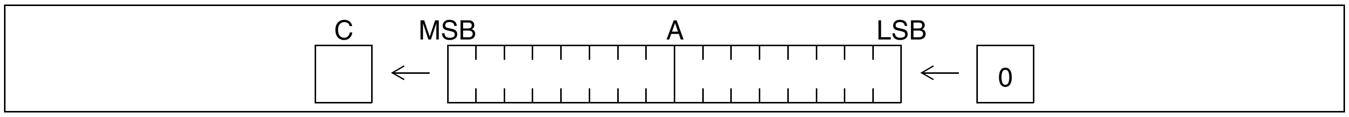
Shift the low-order word data of the accumulator (A) to the left by one bit. The least significant bit of A is set to "0". The bit shifted out from the most significant bit of the low-order word data for A is stored in the carry bit (C).

■ LSLW (Logical Shift Word Data of Accumulator to Left)

● Assembler format:

LSLW A/SHLW A

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

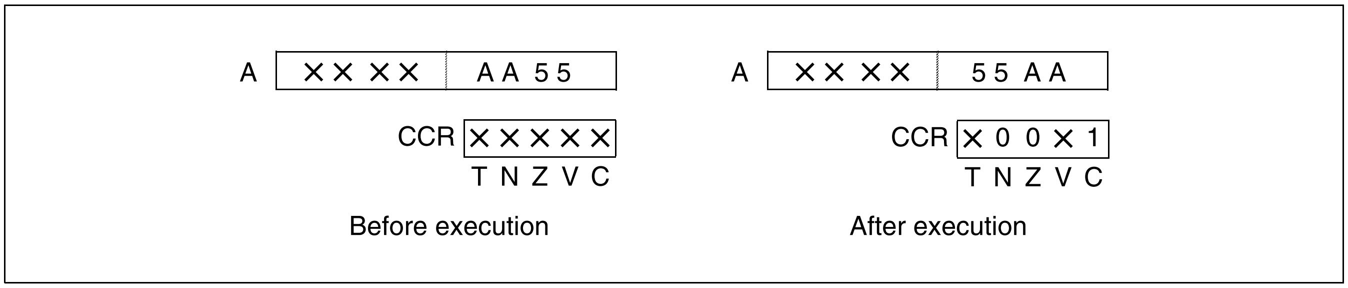
I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit shifted out from the MSB of A.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

LSLW A



9.1.57 LSLW (Logical Shift Word Data of Accumulator to Left)

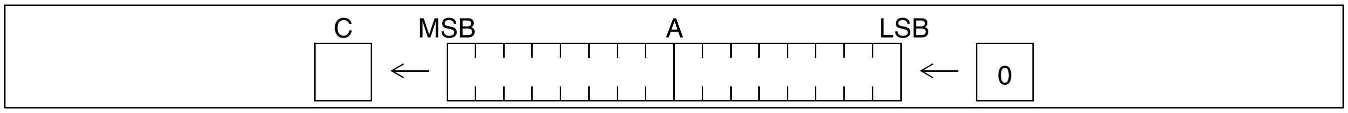
Shift the low-order word data of the accumulator (A) to the left by the number of bits specified by the second operand. The least significant bit of A is set to "0". The bit last shifted out from the most significant bit of the low-order word data for A is stored in the carry bit (C).

■ LSLW (Logical Shift Word Data of Accumulator to Left)

● Assembler format:

LSLW A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

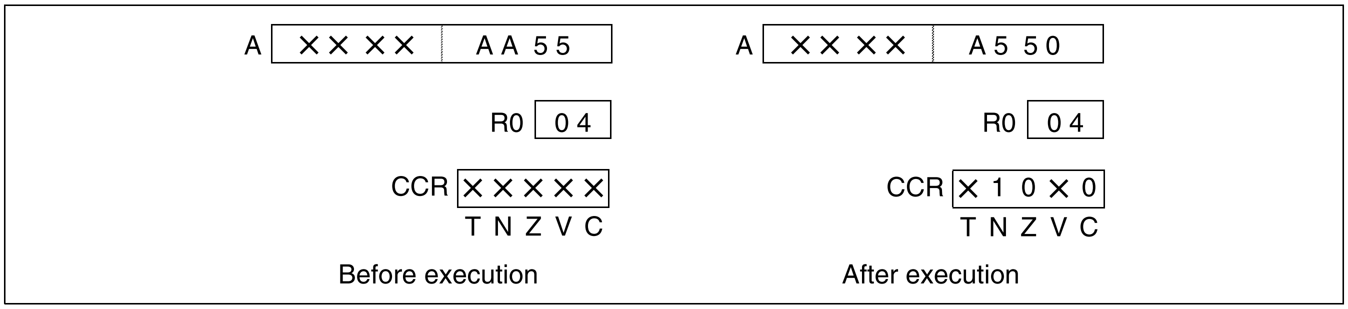
I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the MSB of A. Cleared when the shift amount is "0".

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 5 + (R0)
Correction value:	0

● Example:

LSLW A,R0



9.1.58 LSR (Logical Shift Byte Data of Accumulator to Right)

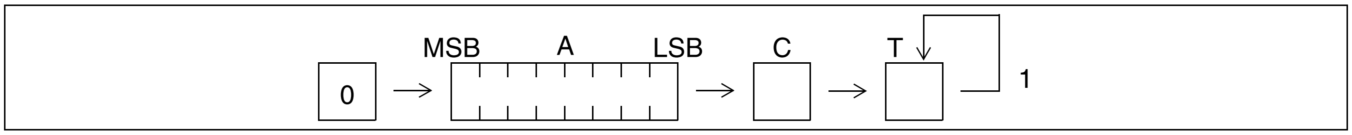
Shift the least significant byte data of the accumulator (A) to the right by the number of bits specified by the second operand. The most significant bit of the least significant byte for A is set to "0". The bit last shifted out from the least significant bit is stored in the carry bit (C).

■ LSR (Logical Shift Byte Data of Accumulator to Right)

● Assembler format:

LSR A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

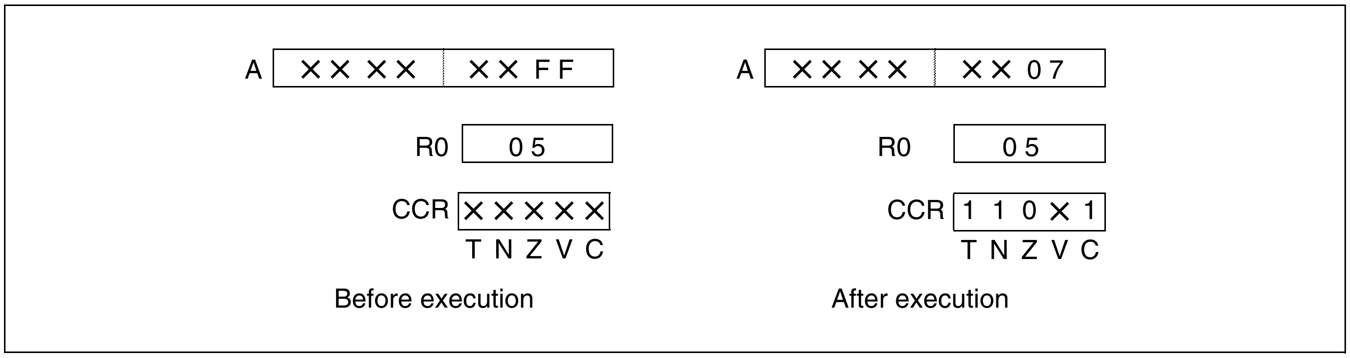
I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 5 + (R0)
Correction value:	0

● Example:

LSR A,R0



9.1.59 LSRL (Logical Shift Long Word Data of Accumulator to Right)

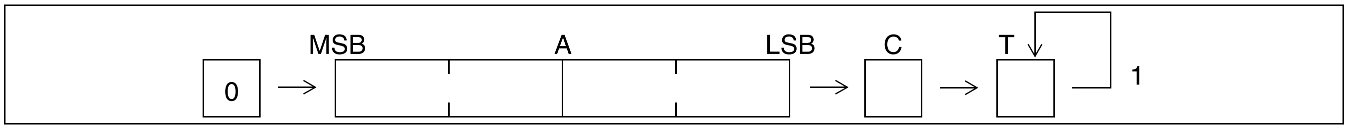
Shift the long word data of the accumulator (A) to the right by the number of bits specified by the second operand. The most significant bit of A is set to "0". The bit last shifted out from the least significant bit of A is stored in the carry bit (C).

■ LSRL (Logical Shift Long Word Data of Accumulator to Right)

- Assembler format:

LSRL A,R0

- Operation:



- CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

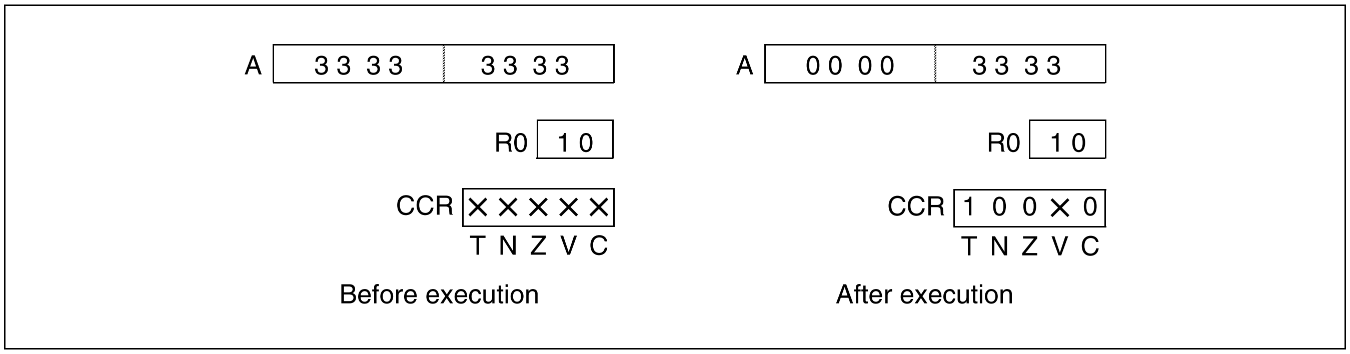
I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 6 + (R0)
Correction value:	0

● Example:

LSRL A,R0



9.1.60 LSRW (Logical Shift Word Data of Accumulator to Right)

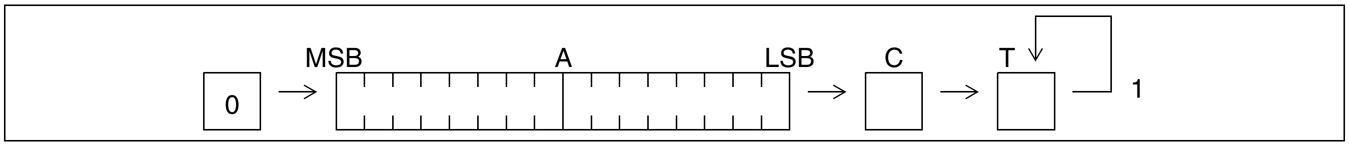
Shift the low-order word data of the accumulator (A) to the right by one bit. The most significant bit of the low-order word data for A is set to "0". The least significant bit is stored in the carry bit (C).

■ LSRW (Logical Shift Word Data of Accumulator to Right)

● Assembler format:

LSRW A/SHRW A

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	R	*	–	*

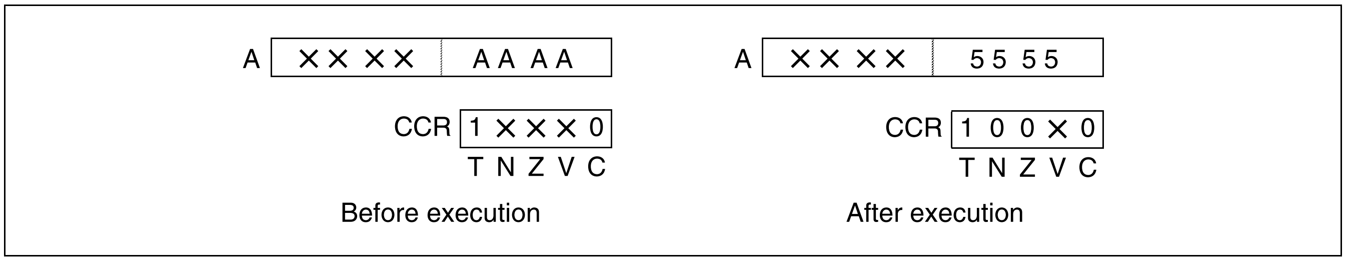
I and S:	Unchanged
T:	Stores the OR of the shifted-out data from the carry and the old T flag value.
N:	Cleared
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit shifted out from the LSB of A.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

LSRW A



9.1.61 LSRW (Logical Shift Word Data of Accumulator to Right)

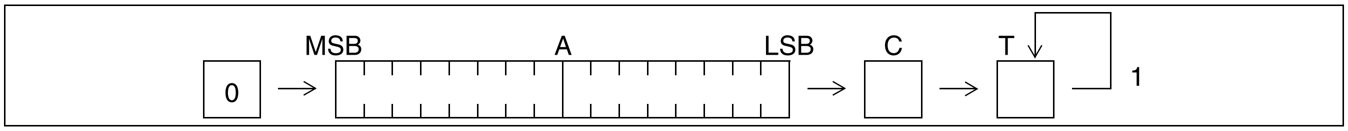
Shift the low-order word data of the accumulator (A) to the right by the number of bits specified by the second operand. The most significant bit of the low-order word data for A is set to "0". The bit last shifted out from the least significant bit is stored in the carry bit (C).

■ LSRW (Logical Shift Word Data of Accumulator to Right)

● Assembler format:

LSRW A,R0

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	*	*	*	–	*

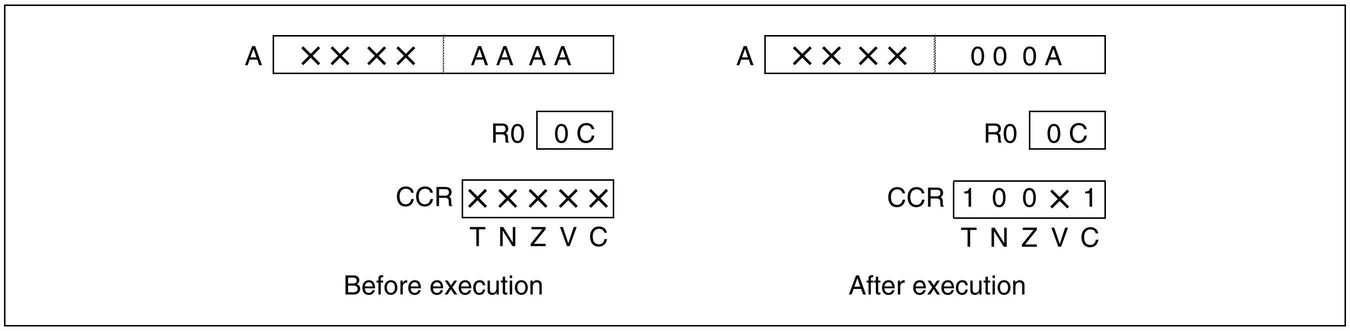
I and S:	Unchanged
T:	Set when the shifted-out data from the carry contains one or more "1" bits, cleared otherwise. Also cleared when the shift amount is "0".
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit last shifted out from the LSB of A. Cleared when the shift amount is "0".

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	6 when (R0) is equal to 0; otherwise, 5 + (R0)
Correction value:	0

● Example:

LSRW A,R0



9.1.62 MOV (Move Byte Data from Source to Accumulator)

Transfer the values of bits 0 to 15 for the accumulator (A) to bits 16 to 31, then transfer "0" to bits 8 to 15. The byte data specified by the second operand is transferred to bits 0 to 7.
If the second operand is @A, transfer to bits 16 to 31 is not performed.

■ MOV (Move Byte Data from Source to Accumulator)

- Assembler format:

MOV A,#imm8	MOV A,Ri
MOV A,@A	MOV A,dir
MOV A,@RLi + disp8	MOV A,addr16
MOV A,io	MOV A,brg1
MOV A,eam	MOV A,ear

- Operation:

(A) ← (Second operand)   (Byte transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

- Number of bytes, Number of cycles, and Correction value:

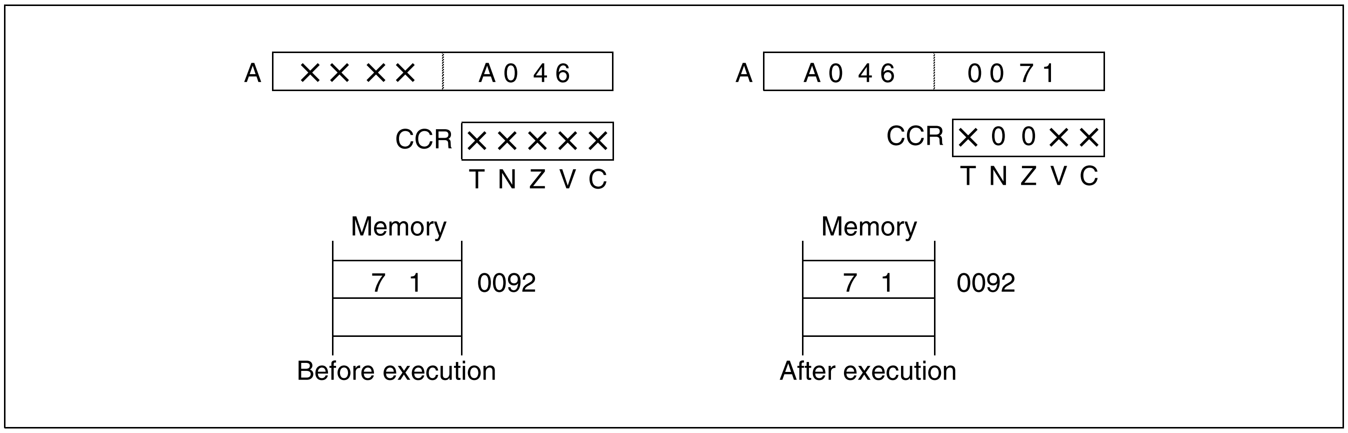
Second operand	#im8	@A	@RLi+8	io	ad16	Ri	dir	ear	eam	brg1
Number of bytes	2	2	3	2	3	1	2	2	2+	2
Number of cycles	2	3	10	3	4	2	3	2	3+(a)	*
Correction value	0	(b)	(b)	(b)	(b)	0	(b)	0	(b)	0

*: One cycle for the program bank register (PCB), additional data bank register (ADB), system stack bank register (SSB), and user stack bank register (USB). Two cycles for the data bank register (DTB) and direct page register (DPR).

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOV A,0092H



9.1.63 MOV (Move Byte Data from Accumulator to Destination)

Transfer the least significant byte data of the accumulator (A) to the address specified by the first operand.

■ MOV (Move Byte Data from Accumulator to Destination)

● Assembler format:

MOV dir,A	MOV Ri,A
MOV @RLi+disp8,A	MOV io,A
MOV addr16,A	MOV brg2,A
MOV ear,A	MOV eam,A

● Operation:

(First operand) ← (A) (Byte transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

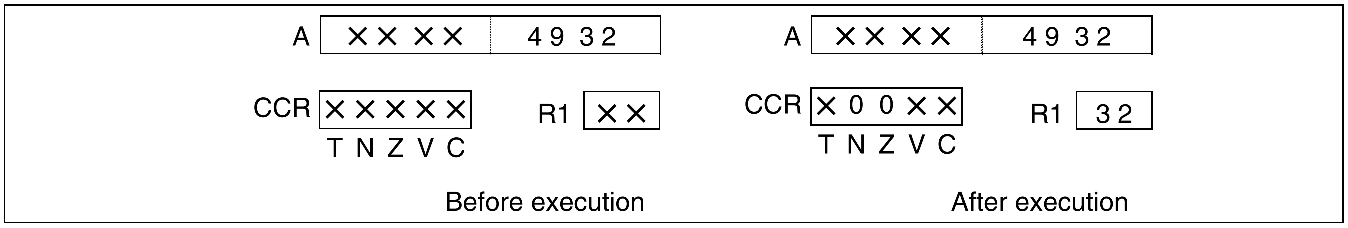
● Number of bytes, Number of cycles, and Correction value:

Second operand	dir	@RLi+8	ad16	io	Ri	ear	eam	brg2
Number of bytes	2	3	3	2	1	2	2+	2
Number of cycles	3	10	4	3	2	2	3+(a)	1
Correction value	(b)	(b)	(b)	(b)	0	0	(b)	0

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOV R1,A



9.1.64 MOV (Move Byte Immediate Data to Destination)

Transfer the 8-bit immediate data specified by the second operand to the address specified by the first operand.
When the first operand is @PC + disp16, the data is transferred to the "address of the location containing the machine instruction for the MOV instruction + 4 + rel", not the "address of the location containing the machine instruction for the instruction following the MOV instruction + rel".

■ MOV (Move Byte Immediate Data to Destination)

● Assembler format:

MOV RP,#imm8	MOV ILM,#imm8
MOV io,#imm8	MOV dir,#imm8
MOV ear,#imm8	MOV eam,#imm8

● Operation:

(First operand) ← #imm8

● CCR:

If the data is transferred to a general-purpose registers (R0 to R7) or bank register

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

If the data is transferred to a register other than the general-purpose registers (R0 to R7) and the bank register

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

I, S, and T:	Unchanged
N:	Unchanged if the data is transferred to a register other than the general-purpose registers. If the data is transferred to the general-purpose register, N is set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Unchanged if the data is transferred to a register other than the general-purpose registers. If the data is transferred to the general-purpose register, Z is set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

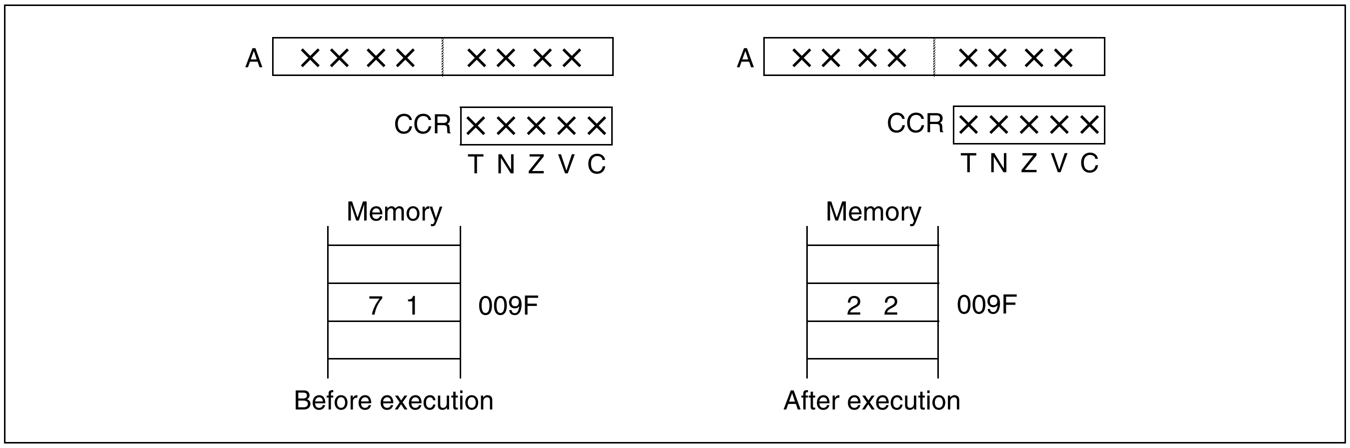
● Number of bytes, Number of cycles, and Correction value:

First operand	RP	ILM	dir	io	ear	eam
Second operand	#im8	#im8	#im8	#im8	#im8	#im8
Number of bytes	2	2	3	3	3	3+
Number of cycles	2	2	5	5	2	4+(a)
Correction value	0	0	(b)	(b)	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOV 009FH,#22H



9.1.65 MOV (Move Byte Data from Source to Destination)

Transfer the byte data specified by the second operand to the first operand.
MOV Ri, #imm8, described below, is an instruction contained in the basic page map (see C.1 Table C-1), with code different from that contained in MOV ear, #imm8.

■ MOV (Move Byte Data from Source to Destination)

● Assembler format:

MOV Ri,#imm8	
MOV Ri,ear	MOV Ri,eam
MOV ear,Ri	MOV eam,Ri

● Operation:

(First operand) ← (Second operand) (Byte transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

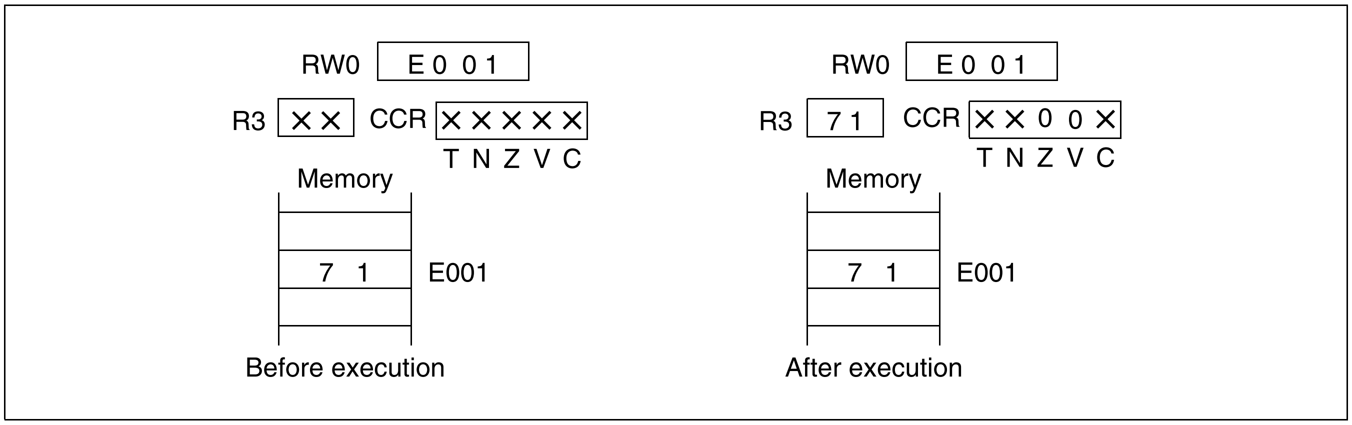
● Number of bytes, Number of cycles, and Correction value:

First operand	Ri	Ri	Ri	ear	eam
Second operand	#im8	ear	eam	Ri	Ri
Number of bytes	2	2	2+	2	2+
Number of cycles	2	3	4+(a)	4	5+(a)
Correction value	0	0	(b)	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOV R3,@RW0



9.1.66 MOV (Move Byte Data from AH to Memory)

Transfer the low-order byte data of AH to the memory location specified by the contents of AL.

■ MOV (Move Byte Data from AH to Memory)

- Assembler format:

MOV @AL,AH

- Operation:

((AL)) ← (AH)   (Byte transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

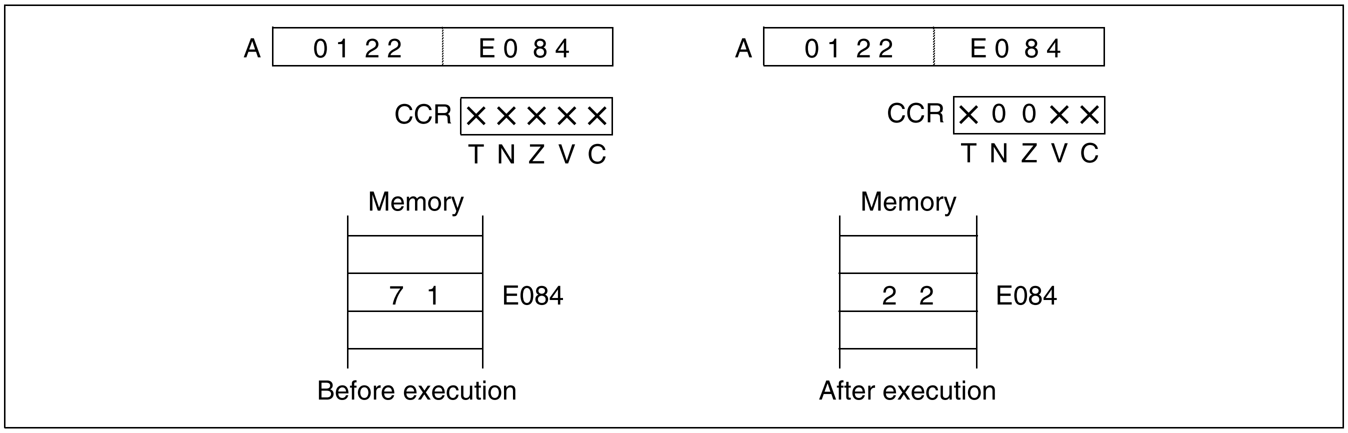
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3
Correction value:	(b)

For the explanation of (b), see Table 8.4-2 .

- Example:

MOV @AL,AH



9.1.67 MOVB (Move Bit Data from Bit Address to Accumulator)

Transfer zeros to bits 8 to 15 of the accumulator (A). "00_H" is transferred to bits 0 to 7 of A if the bit of the address specified by the second operand is equal to "0" and "FF_H" is transferred if the bit is equal to "1".

■ MOVB (Move Bit Data from Bit Address to Accumulator)

● Assembler format:

MOVB A,addr16:bp

MOVB A,dir:bp

MOVB A,io:bp

● Operation:

If (Second operand)=0 : (A) ← 00_H (Byte transfer)

If (Second operand)=1 : (A) ← FF_H (Byte transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the transferred bit is "1", cleared when "0".
Z:	Set when the transferred bit is "0", cleared when "1".
V and C:	Unchanged

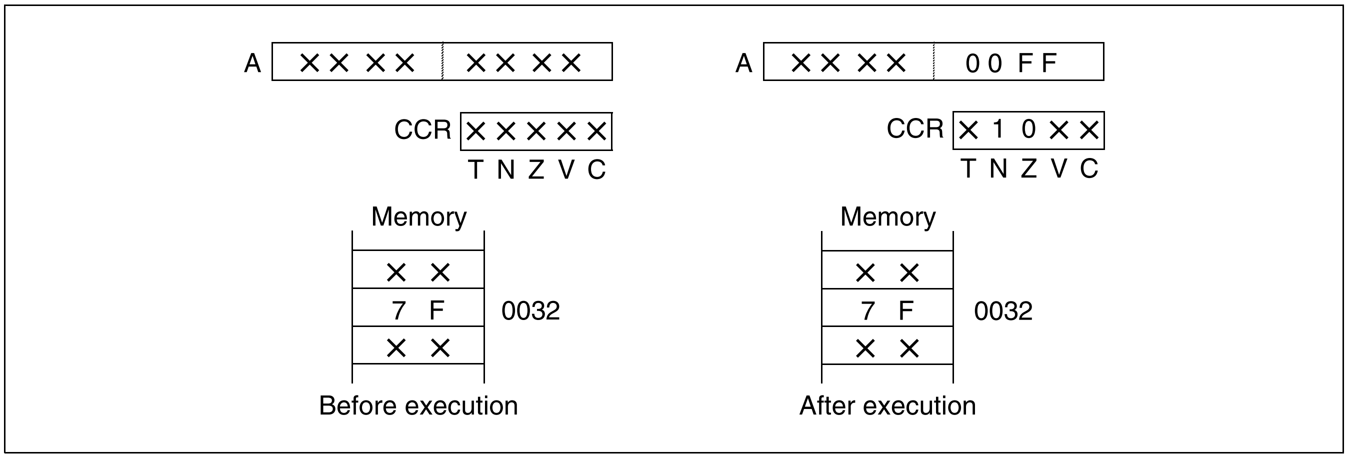
● Number of bytes, Number of cycles, and Correction value:

Second operand	ad16:bp	dir:bp	io:bp
Number of bytes	4	3	3
Number of cycles	5	5	4
Correction value	(b)	(b)	(b)

For the explanation of (b) in the table, see Table 8.4-2 .

● Example:

MOVB A,32H:3



9.1.68 MOVB (Move Bit Data from Accumulator to Bit Address)

Transfer bit data 0 to the bit address specified by the first operand if the least significant byte data of the accumulator (A) is 00_H.
Bit data 1 is transferred to the bit address specified by the first operand if the least significant byte data of A is not 00_H.

■ MOVB (Move Bit Data from Accumulator to Bit Address)

● Assembler format:

MOVB addr16:bp,A

MOVB dir:bp,A

MOVB io:bp,A

● Operation:

If the byte data of (A) is 00_H : (First operand) b=0 (Bit transfer)

If the byte data of (A) is not 00_H : (First operand) b=1 (Bit transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the byte data for A is "1", cleared otherwise.
Z:	Set when the byte data of A is "0", cleared otherwise.
V and C:	Unchanged

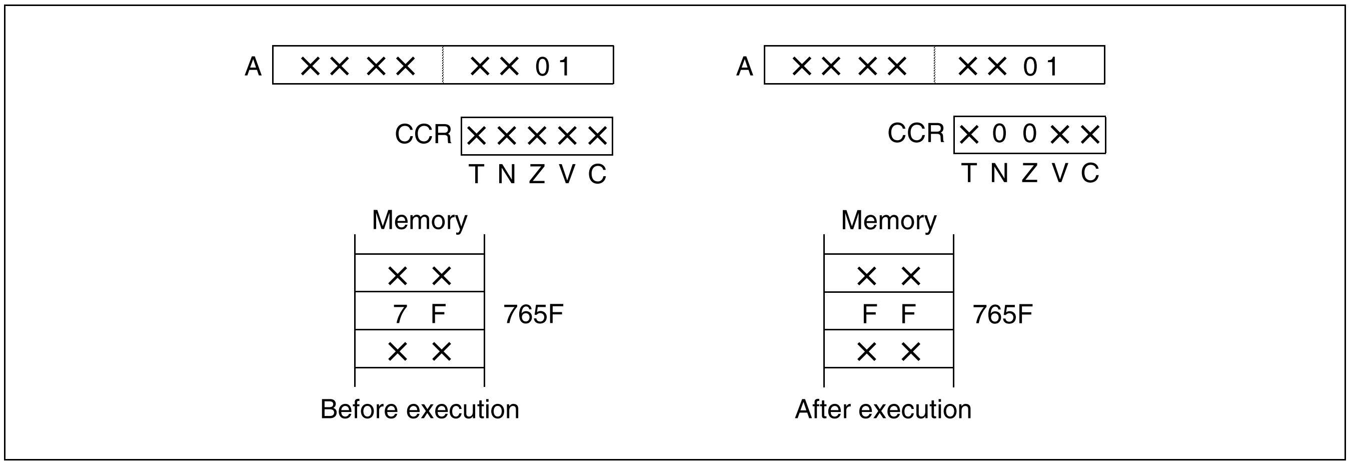
● Number of bytes, Number of cycles, and Correction value:

Second operand	ad16:bp	dir:bp	io:bp
Number of bytes	4	3	3
Number of cycles	7	7	6
Correction value	2×(b)	2×(b)	2×(b)

For the explanation of (b) in the table, see Table 8.4-2 .

● Example:

MOVB 765FH: 7,A



9.1.69 MOVEA (Move Effective Address to Destination)

Transfer the value specified by the second operand (effective address) to the first operand. If a general-purpose register is specified by the second operand, the address of the general-purpose register is transferred.
If the destination (first operand) is the accumulator (A), the pre-transfer values of bits 0 to 15 are transferred to bits 16 to 31 of A.

■ MOVEA (Move Effective Address to Destination)

● Assembler format:

MOVEA <destination>,ear MOVEA <destination>,eam

● Operation:

First operand ← ea (Word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

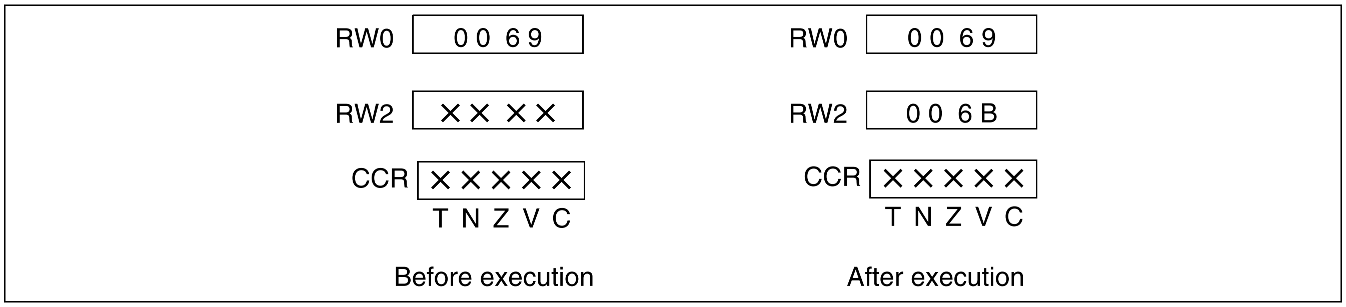
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	RWi	RWi
Second operand	ear	eam	ear	eam
Number of bytes	2	2+	2	2+
Number of cycles	1	1+(a)	3	2+(a)
Correction value	0	0	0	0

For the explanation of (a) in the table, see Table 8.4-1 .

● Example:

MOVEA RW2,@RW0+2



9.1.70 MOVL (Move Long Word Data from Source to Accumulator)

Transfer the long word data specified by the second operand to the accumulator (A).

■ MOVL (Move Long Word Data from Source to Accumulator)

● Assembler format:

MOVL A,#imm32
MOVL A,ear   MOVL A,eam

● Operation:

(A) ← (Second operand)   (Long word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

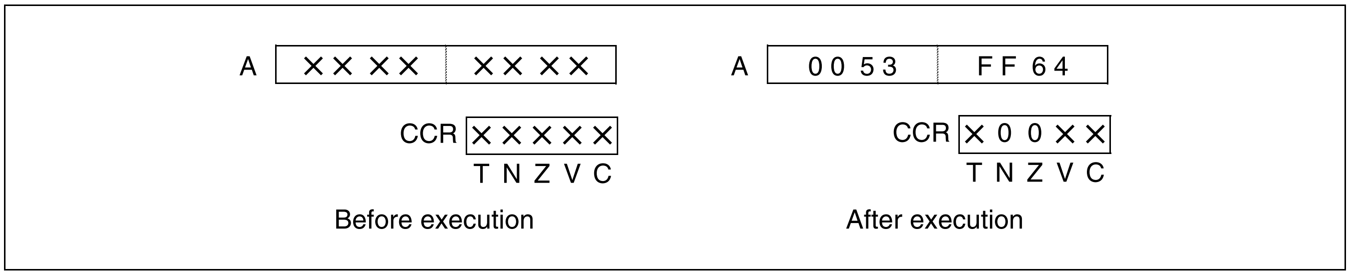
● Number of bytes, Number of cycles, and Correction value:

Second operand	#i32	ear	eam
Number of bytes	5	2	2+
Number of cycles	3	4	5+(a)
Correction value	0	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVL A,#0053FF64H



9.1.71 MOVL (Move Long Word Data from Accumulator to Destination)

Transfer the long word data of the accumulator (A) to the first operand.

■ MOVL (Move Long Word Data from Accumulator to Destination)

● Assembler format:

MOVL ear,A MOVL eam,A

● Operation:

(First operand) ← (A) (Long word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

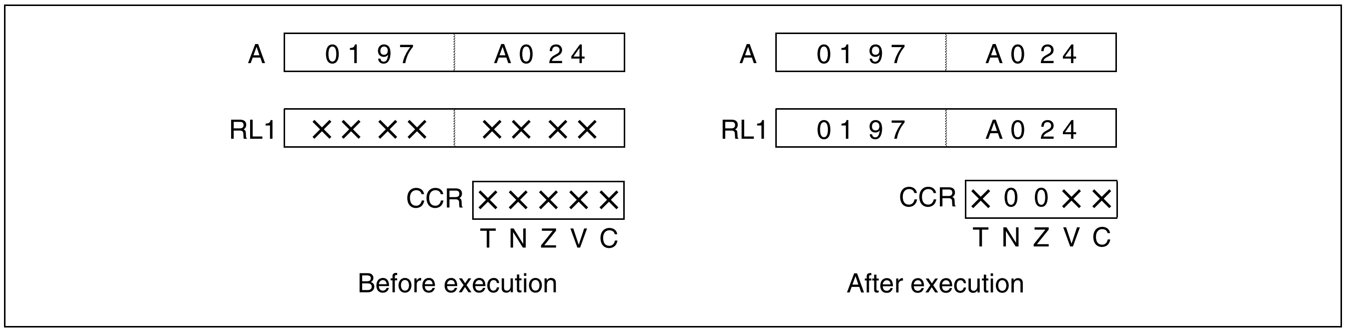
● Number of bytes, Number of cycles, and Correction value:

First operand	ear	eam
Number of bytes	2	2+
Number of cycles	4	5+(a)
Correction value	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVL RL1,A



9.1.72 MOVN (Move Immediate Nibble Data to Accumulator)

Transfer the values of bits 0 to 15 for the accumulator (A) to bits 16 to 31. "0" is transferred to bits 4 to 15 and the nibble data specified by the second operand is transferred to bits 0 to 3.

■ MOVN (Move Immediate Nibble Data to Accumulator)

● Assembler format:

MOVN A,#imm4

● Operation:

(A) ← imm4 (Byte transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	R	*	–	–

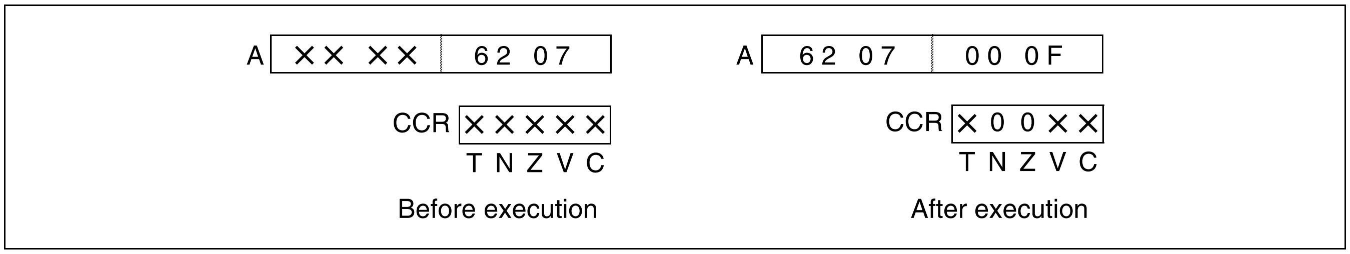
I, S, and T:	Unchanged
N:	Cleared
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	1
Correction value:	0

● Example:

MOVN A,#0FH



9.1.73 MOVS (Move String Byte)

Transfer byte data from the address specified by AL in the space specified by <source bank> to the address specified by AH in the space specified by <destination bank>. The transfer is repeated the number of times specified by RW0, with the addresses being changed each time. The transfer is not performed if RW0 is equal to "0". Four types of registers PCB, DTB, ADB, and SPB can be used as <destination bank> and <source bank>. By default, DTB is assumed.
The addresses can be either incremented or decremented. By default, the addresses are incremented.
If an interrupt occurs during the transfer, the transfer is suspended to handle the interrupt. The transfer is resumed after the interrupt has been handled.

■ MOVS (Move String Byte)

● Assembler format:

MOVSI [<destination bank>] [,<source bank>] (When the addresses are incremented)

MOVSD [<destination bank>] [,<source bank>] (When the addresses are decremented)

● Operation:

The following is repeated until RW0 becomes equal to "0":

((AH)) ← ((AL)) (Byte transfer)

(AH) ← (AH)±1, (AL) ← (AL)±1 (+ if MOVSWI, – if MOVSWD)

(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

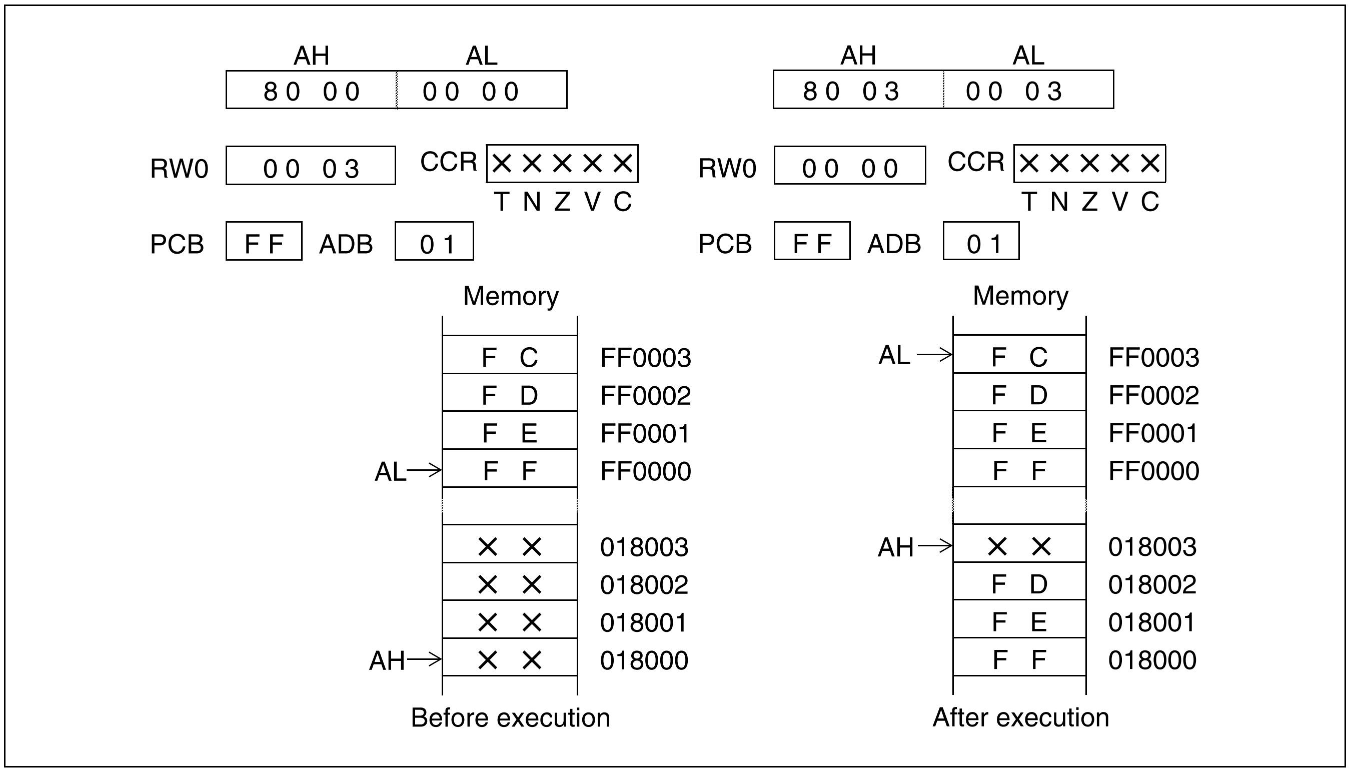
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	5 if (RW0) is equal to "0"; otherwise, 4+8×(RW0)
Correction value:	2×(b)×(RW0)

For the explanation of (b), see Table 8.4-2 .

● Example:

MOVSI ADB,PCB



9.1.74 MOVSW (Move String Word)

Transfer word data from the address specified by AL in the space specified by <source bank> to the address specified by AH in the space specified by <destination bank>. The transfer is repeated the number of times specified by RW0, with the addresses being changed each time. The transfer is not performed if RW0 is equal to "0". Four types of registers PCB, DTB, ADB, and SPB can be used as <destination bank> and <source bank>. By default, DTB is assumed.
The addresses can be either incremented or decremented. By default, the addresses are incremented.
If an interrupt occurs during the transfer, the transfer is suspended to handle the interrupt. The transfer is resumed after the interrupt has been handled.

■ MOVSW (Move String Word)

● Assembler format:

MOVSWI [<destination bank>] [,<source bank>] (When the addresses are incremented)
MOVSWD [<destination bank>] [,<source bank>] (When the addresses are decremented)

● Operation:

The following is repeated until RW0 becomes equal to "0":
((AH)) ← ((AL)) (Byte transfer)
(AH) ← (AH)±2, (AL) ← (AL)±2 (+ if MOVSWI, – for MOVSWD)
(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

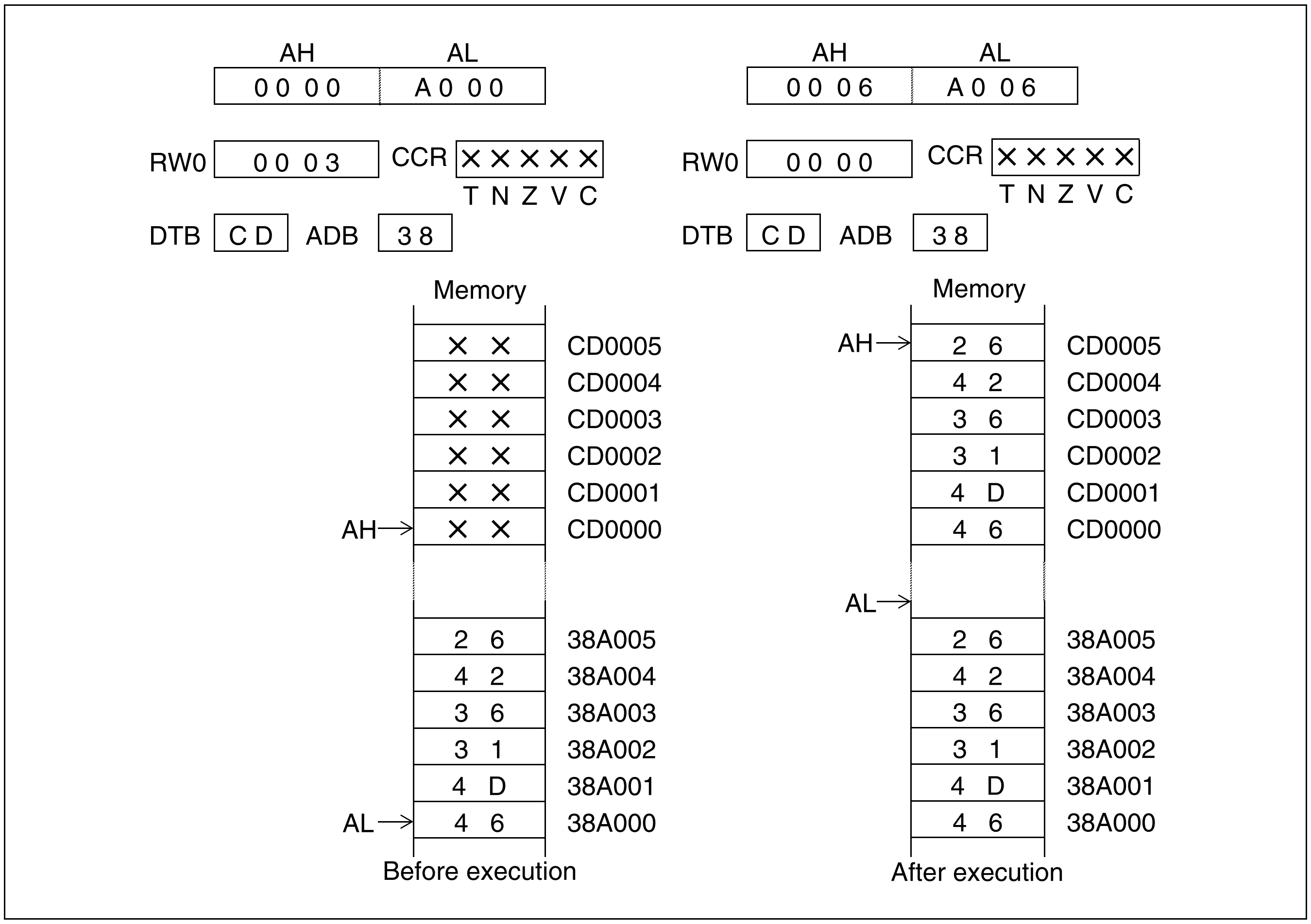
None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes: 2
Number of cycles: 5 if (RW0) is equal to "0"; otherwise, 4 + 8×(RW0)
Correction value: 2×(c)×(RW0)
For the explanation of (c), see Table 8.4-2 .

● Example:

MOVSW,ADB



9.1.75 MOVW (Move Word Data from Source to Accumulator)

Transfer the values of bits 0 to 15 for the accumulator (A) to bits 16 to 31. Then, the word data specified by the second operand is transferred to bit0 to bit15 of A. If the second operand is @A, transfer the values of bits 0 to 15 of A to bits 16 to 31 is not performed.

■ MOVW (Move Word Data from Source to Accumulator)

- Assembler format:

MOVW A,#imm16	MOVW A,@RWi+disp8
MOVW A,@A	MOVW A,addr16
MOVW A,@RLi+disp8	MOVW A,RWi
MOVW A,SP	MOVW A,dir
MOVW A,io	
MOVW A,ear	MOVW A,eam

- Operation:

(A) ← (Second operand) (Word transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

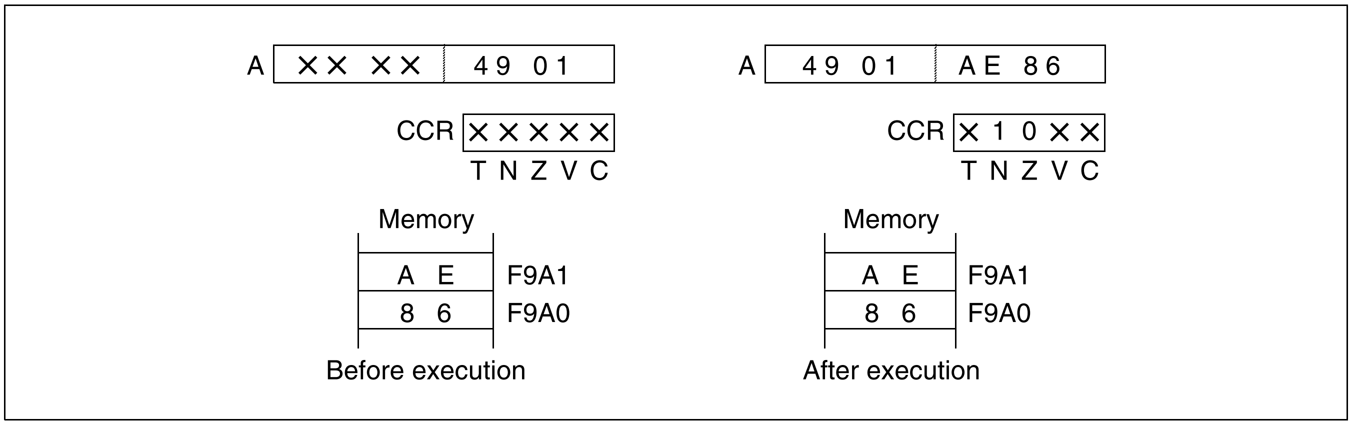
- Number of bytes, Number of cycles, and Correction value:

Second operand	#i16	@A	@RLi+8	SP	io	@RWi+8	ad16	RWi	dir	ear	eam
Number of bytes	3	2	3	1	2	2	3	1	2	2	2+
Number of cycles	2	3	10	1	3	5	4	2	3	2	3+(a)
Correction value	0	(c)	(c)	0	(c)	(c)	(c)	0	(c)	0	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVW A,0F9A0H



9.1.76 MOVW (Move Word Data from Accumulator to Destination)

Transfer the low-order word data of the accumulator (A) to the first operand.

■ MOVW (Move Word Data from Accumulator to Destination)

● Assembler format:

MOVW @RLi+disp8,A	MOVW addr16,A
MOVW SP,A	MOVW RWi,A
MOVW io,A	MOVW dir,A
MOVW @RWi+disp8,A	
MOVW ear,A	MOVW eam,A

● Operation:

(First operand) ← (A) (Word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

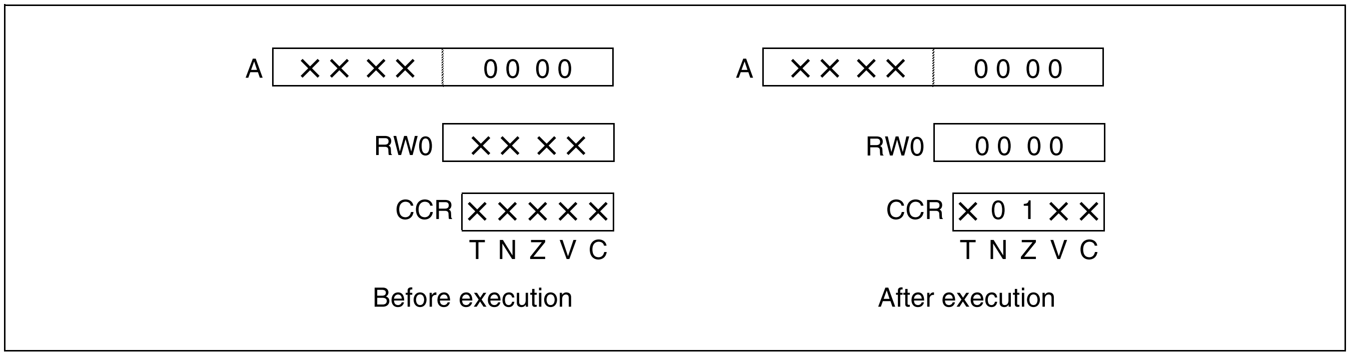
● Number of bytes, Number of cycles, and Correction value:

First operand	dir	@RLi+8	ad16	SP	io	@RWi+8	RWi	ear	eam
Number of bytes	2	3	3	1	2	2	1	2	2+
Number of cycles	3	10	4	1	3	5	2	2	3+(a)
Correction value	(c)	(c)	(c)	0	(c)	(c)	0	0	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVW RW0,A



9.1.77 MOVW (Move Immediate Word Data to Destination)

This instruction transfers the 16-bit immediate data to the first operand.
When the first operand is @PC + disp16, the transfer destination address is the address where the machine instruction of the MOVW instruction is stored + 4 + disp16. Note that this is not the address where the machine instruction of the instruction subsequent to the MOVW instruction is stored+disp16.

■ MOVW (Move Immediate Word Data to Destination)

● Assembler format:

MOVW ear,#imm16   MOVW eam,#imm16

● Operation:

(First operand) ← #imm16

● CCR:

If the data is transferred to a general-purpose register (RW0 to RW7)

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

If the data is transferred to a register other than the general-purpose registers (RW0 to RW7)

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

I, S, and T:	Unchanged
N:	Unchanged if the data is transferred to a register other than the general-purpose registers. If the data is transferred to the general-purpose register, N is set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Unchanged if the data is transferred to a register other than the general-purpose registers. If the data is transferred to the general-purpose register, Z is set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged and none of the flags is changed.

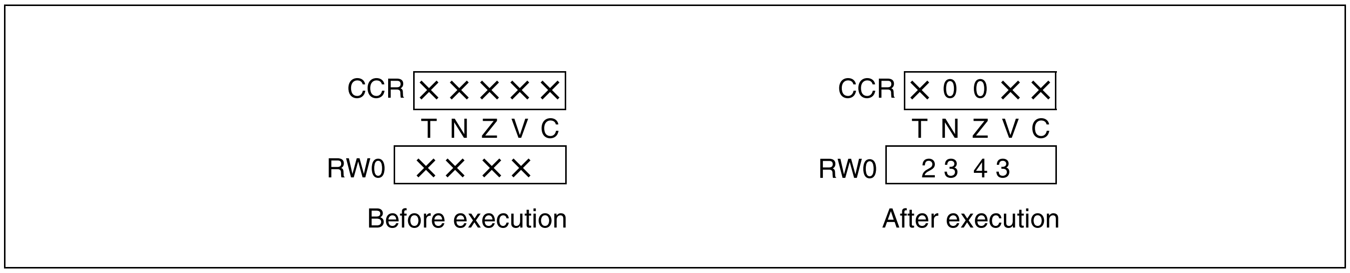
● Number of bytes, Number of cycles, and Correction value:

First operand	ear	eam
Number of bytes	4	4+
Number of cycles	2	4+(a)
Correction value	0	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVW RW0,#2343H



9.1.78 MOVW (Move Word Data from Source to Destination)

Transfer the word data specified by the second operand to the first operand.

■ MOVW (Move Word Data from Source to Destination)

● Assembler format:

MOVW RWi,#imm16	
MOVW ear,RWi	MOVW eam,RWi
MOVW RWi,ear	MOVW RWi,eam

● Operation:

(First operand) ← (Second operand) (Word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

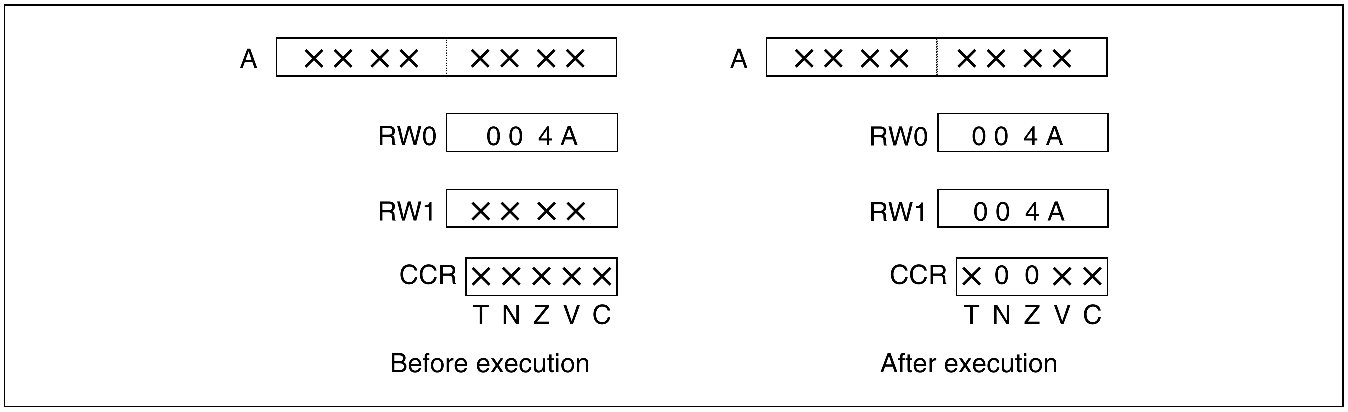
● Number of bytes, Number of cycles, and Correction value:

First operand	RWi	RWi	RWi	ear	eam
Second operand	#i16	ear	eam	RWi	RWi
Number of bytes	3	2	2+	2	2+
Number of cycles	2	4	5+(a)	3	4+(a)
Correction value	0	0	(c)	0	(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVW RW1,RW0



9.1.79 MOVW (Move Immediate Word Data to io)

Transfer 16-bit immediate data to the I/O area specified by the first operand.

■ MOVW (Move Immediate Word Data to io)

- Assembler format:

  MOVW io,#imm16

- Operation:

  (First operand) ← imm16 (Word transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

  Number of bytes: 4

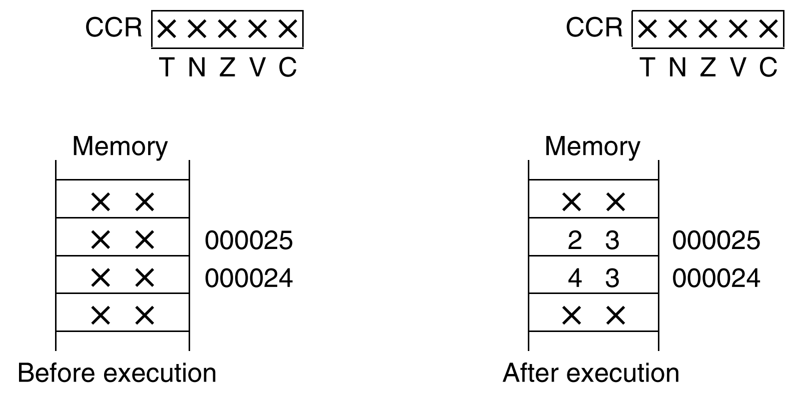
  Number of cycles: 5

  Correction value: (c)

  For the explanation of (c), see Table 8.4-2 .

- Example:

  MOVW 24H,#2343H



9.1.80 MOVW (Move Word Data from AH to Memory)

Transfer the word data of AH to the memory location specified by the contents of AL.

■ MOVW (Move Word Data from AH to Memory)

● Assembler format:

MOVW @AL,AH

● Operation:

((AL)) ← (AH)   (Word transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

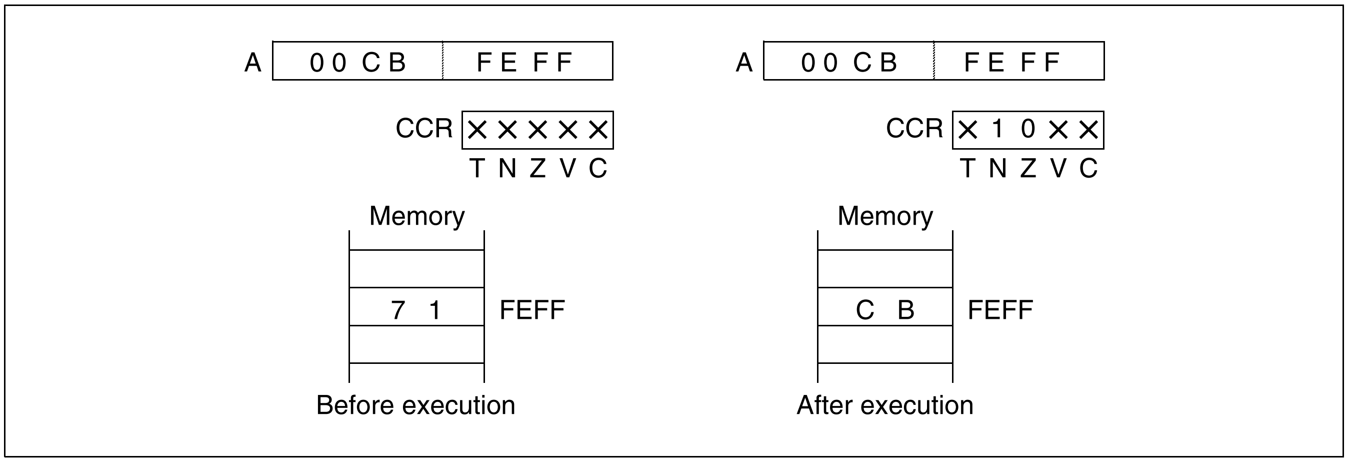
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3
Correction value:	(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

MOVW @AL,AH



9.1.81 MOVX (Move Byte Data with Sign Extension from Source to Accumulator)

Transfer the values of bits 0 to 15 for the accumulator (A) to bits 16 to 31. Then, the value resulting from sign-extending the second operand is transferred to bits 0 to 15 of A. If the second operand is @A, transfer to bits 16 to 31 is not performed.

■ MOVX (Move Byte Data with Sign Extension from Source to Accumulator)

● Assembler format:

MOVX A,#imm8	MOVX A,@RWi+disp8
MOVX A,@A	MOVX A,addr16
MOVX A,@RLi+disp8	MOVX A,Ri
MOVX A,dir	MOVX A,io
MOVX A,ear	MOVX A,eam

● Operation:

(A) ← (Second operand) (Byte transfer with sign extension)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	–

I, S, and T:	Unchanged
N:	Set when the MSB of the transferred data is "1", cleared otherwise.
Z:	Set when the transferred data is "0", cleared otherwise.
V and C:	Unchanged

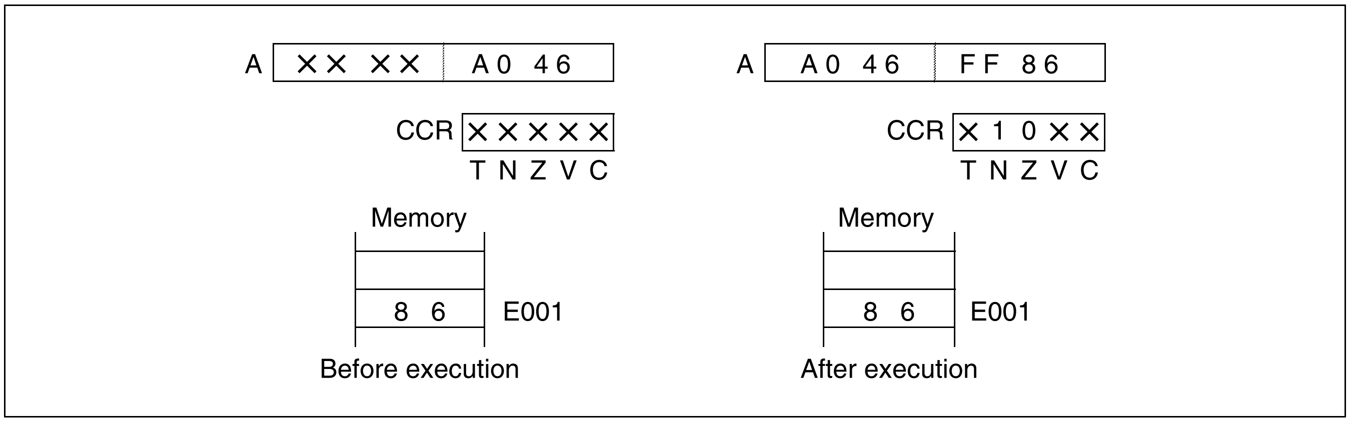
● Number of bytes, Number of cycles, and Correction value:

Second operand	#im8	@A	@RLi+8	dir	io	@RWi+8	ad16	Ri	ear	eam
Number of bytes	2	2	3	2	2	2	3	2	2	2+
Number of cycles	2	3	10	3	3	5	4	2	2	3+(a)
Correction value	0	(b)	(b)	(b)	(b)	(b)	(b)	0	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

MOVX A,0E001H



9.1.82 MUL (Multiply Byte Data of Accumulator)

This instruction multiplies the low-order byte data of AH by that of AL as signed binary numbers, then returns the result to AL of the accumulator (A).

■ MUL (Multiply Byte Data of Accumulator)

- Assembler format:

  MUL A

- Operation:

  word (A) ← byte (AH)×byte (AL)   (Byte multiplication)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

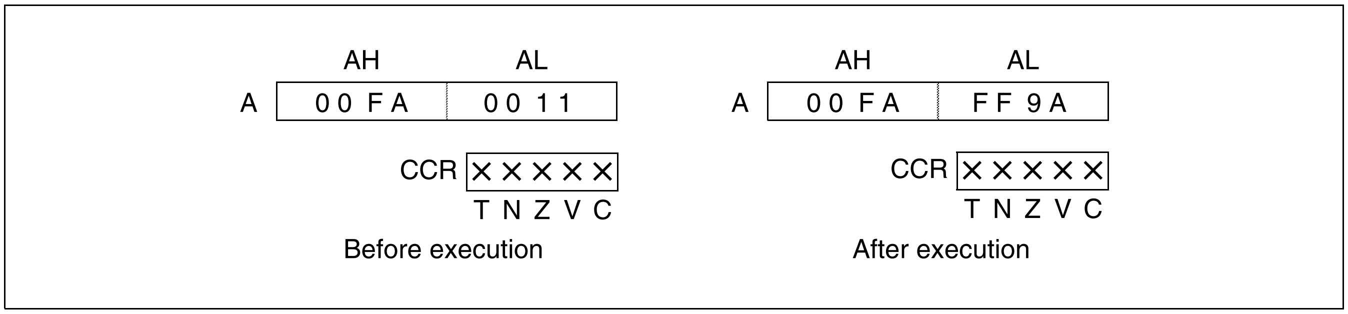
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3 if byte (AH) is equal to "0"; 12 if byte (AH) is not equal to "0" and the result is positive; 13 if the result is negative.
Correction value:	0

- Example:

  MUL A



9.1.83 MUL (Multiply Byte Data of Accumulator and Effective Address)

Multiply the byte data of the accumulator (A) by the byte data specified by the second operand as signed binary numbers and restore the result in bits 0 to 15 of A.

■ MUL (Multiply Byte Data of Accumulator and Effective Address)

● Assembler format:

MUL A,ear MUL A,eam

● Operation:

word (A) ← byte (A) × byte (ea) (Byte multiplication)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

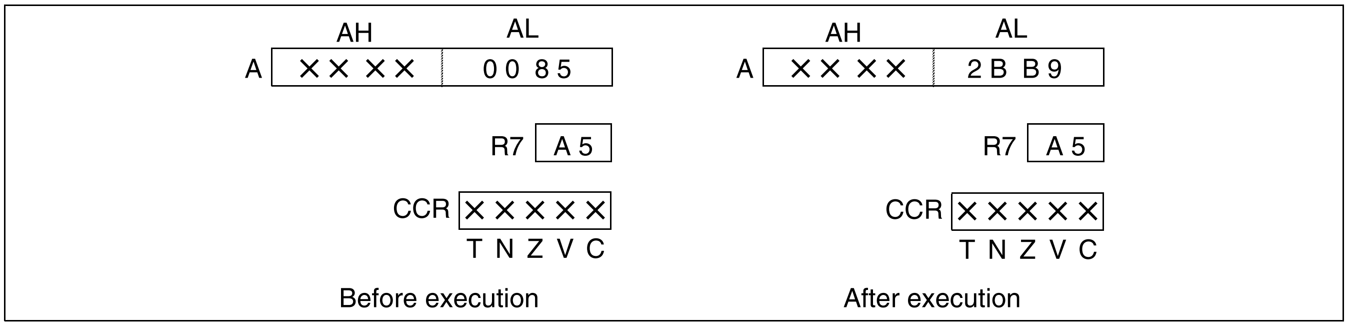
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	*1	*2
Correction value	0	(b)

*1: 4 if byte (ear) is equal to "0"; 13 if byte (ear) is not equal to"0" and the result is positive; 14 if the result is negative.
*2: 5 + (a) if byte (eam) is equal to "0"; 14 + (a) if byte (eam) is not equal to "0" and the result is positive; 15 + (a) if the result is negative.

For the explanation of (b) in the table and (a) in *2, see Table 8.4-1 and Table 8.4-2 .

● Example:

MUL A,R7



9.1.84 MULW (Multiply Word Data of Accumulator)

Multiply the word data of AH by the word data specified by AL as signed binary numbers and restore the result in the accumulator as long word data.

■ MULW (Multiply Word Data of Accumulator)

- Assembler format:

MULW A

- Operation:

Long (A) ← word (AH)×word (AL) (Word multiplication)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

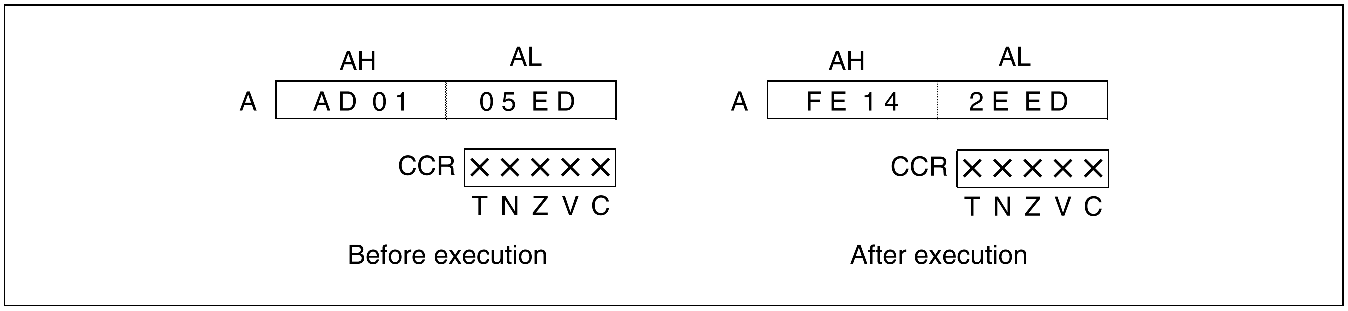
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3 if word (AH) is equal to "0"; 16 if word (AH) is not equal to "0" and the result is positive; 19 if the result is negative.
Correction value:	0

- Example:

MULW A



9.1.85 MULW (Multiply Word Data of Accumulator and Effective Address)

Multiply the word data of the accumulator (A) by the word data specified by the second operand as signed binary numbers and restore the result in A as long word data.

■ MULW (Multiply Word Data of Accumulator and Effective Address)

● Assembler format:

MULW A,ear MULW A,eam

● Operation:

Long (A) ← word (A)×word (Second operand) (Word multiplication)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

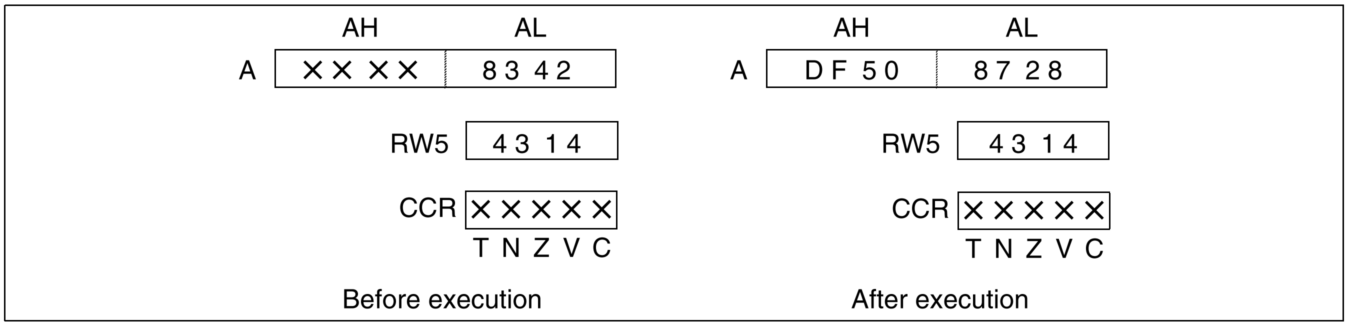
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	*1	*2
Correction value	0	(b)

*1: 4 if Word (ear) is equal to "0"; 17 if Word (ear) is not equal to "0" and the result is positive; 20 if the result is negative.
*2: 5 + (a) if Word (eam) is equal to "0"; 18 + (a) if Word (eam) is not equal to "0" and the result is positive; 21 + (a) if the result is negative.

For the explanation of (b) in the table and (a) in *2, see Table 8.4-1 and Table 8.4-2 .

● Example:

MULW A,RW5



9.1.86 MULU (Multiply Unsigned Byte Data of Accumulator)

Multiply the low-order byte data of AH by the low-order byte data of AL as unsigned binary numbers and restore the result in the AL of the accumulator (A).

■ MULU (Multiply Unsigned Byte Data of Accumulator)

- Assembler format:

MULU A

- Operation:

word (A) ← byte (AH)×byte (AL) (Byte multiplication)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

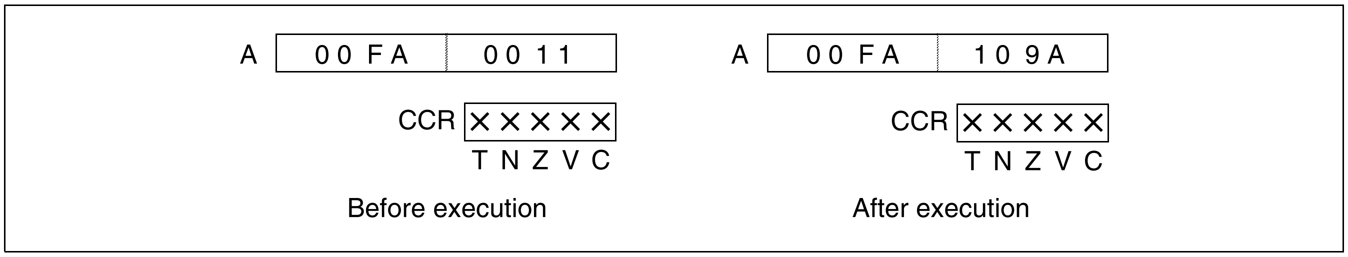
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3 if byte (AH) is equal to "0"; 7 if byte (AH) is not equal to "0".
Correction value:	0

- Example:

MULU A



9.1.87 MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address)

Multiply the byte data of the accumulator (A) by the byte data specified by the second operand as unsigned binary numbers and restore the result in bits 0 to 15 of A.

■ MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address)

● Assembler format:

MULU A, ear   MULU A, eam

● Operation:

word (A) ← byte (A) × byte (Second operand)   (Byte multiplication)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

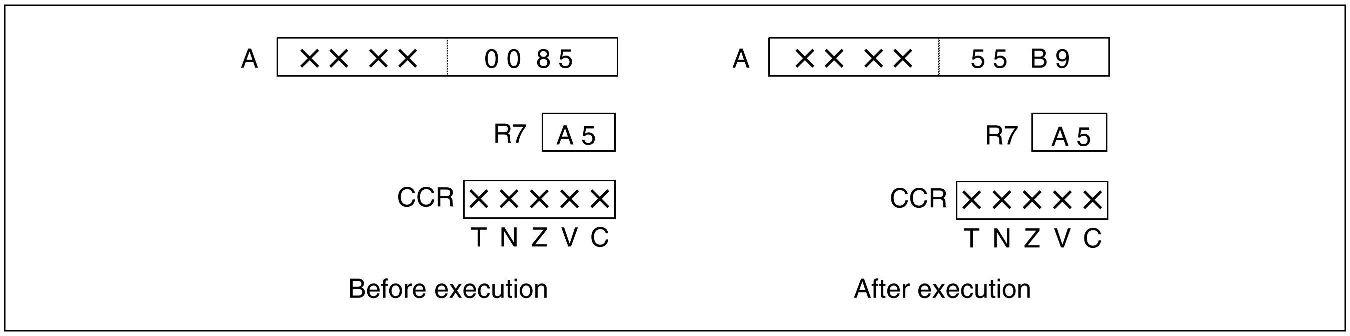
Second operand	ear	eam
Number of bytes	2	2 +
Number of cycles	*1	*2
Correction value	0	(b)

*1: 4 if byte (ear) is equal to "0"; 8 if byte (ear) is not equal to "0".
*2: 5 + (a) if byte (eam) is equal to "0"; 9 + (a) if not equal to "0".

For the explanation of (b) in the table and (a) in *2, see Table 8.4-1 and Table 8.4-2 .

● Example:

MULU A, R7



9.1.88 MULUW (Multiply Unsigned Word Data of Accumulator)

Multiply the word data of AH by the word data of AL as unsigned binary numbers and restore the result in the accumulator (A) as long word data.

■ MULUW (Multiply Unsigned Word Data of Accumulator)

- Assembler format:

MULUW A

- Operation:

Long (A) ← word (AH)×word (AL) (Word multiplication)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

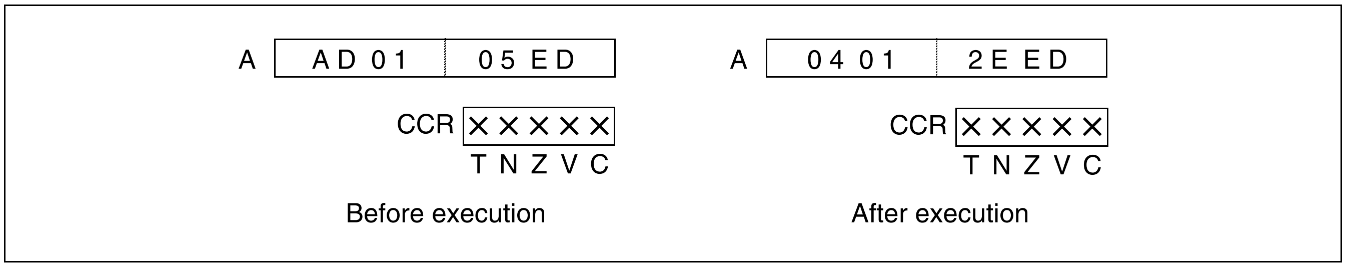
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3 if word (AH) is equal to "0"; 11 if word (AH) is not equal to "0".
Correction value:	0

- Example:

MULUW A



9.1.89 MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address)

Multiply the word data of the accumulator (A) by the word data specified by the second operand as unsigned binary numbers and restore the result in A as long word data.

■ MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address)

- Assembler format:

  MULUW A, ear   MULUW A, eam

- Operation:

  Long (A) ← word (A)×word (Second operand)   (Word multiplication)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

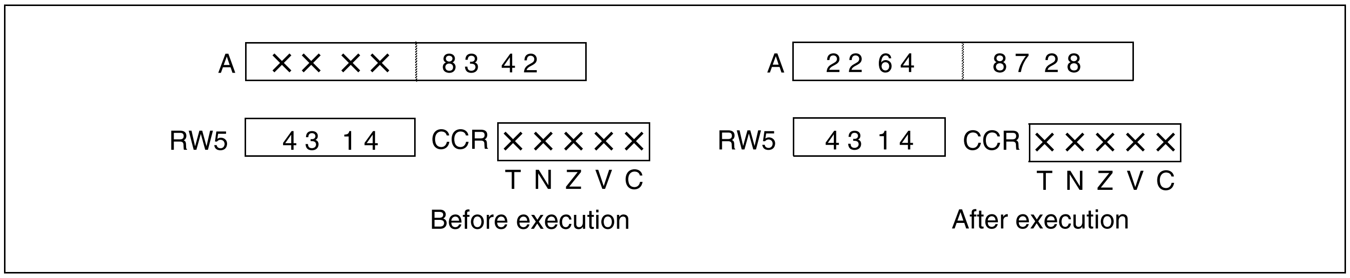
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	*1	*2
Correction value	0	(c)

*1: 4 if Word (ear) is equal to "0"; 12 if Word (ear) is not equal to "0".
*2: 5 + (a) if Word (eam) is "0"; 13 + (a) if Word (eam) is not equal to "0".

For the explanation of (c) in the table and (a) in *2, see Table 8.4-1 and Table 8.4-2 .

- Example:

  MULUW A, RW5



9.1.90 NEG (Negate Byte Data of Destination)

Take the 2's complement of the byte data specified by the operand and restore the result in the operand. If the operand is the accumulator (A), the value resulting from sign-extending the operation result is transferred to bits 8 to 15 of A.

■ NEG (Negate Byte Data of Destination)

● Assembler format:

NEG A

NEG ear NEG eam

● Operation:

(Operand) ← 0–(Operand) (Byte operation)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

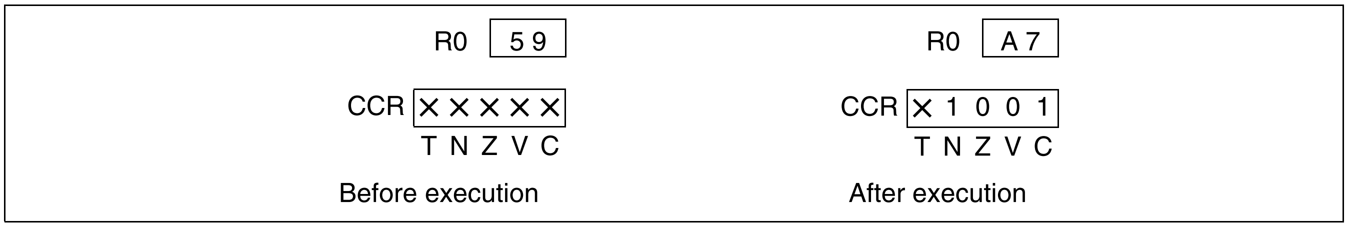
● Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	1	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

NEG R0



9.1.91 NEGW (Negate Word Data of Destination)

Take the 2's complement of the word data specified by the operand and restore the result in the operand.

■ NEGW (Negate Word Data of Destination)

- Assembler format:

NEGW A

NEGW ear　　　　NEGW eam

- Operation:

(Operand) ← 0–(Operand)　　　　(Word operation)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

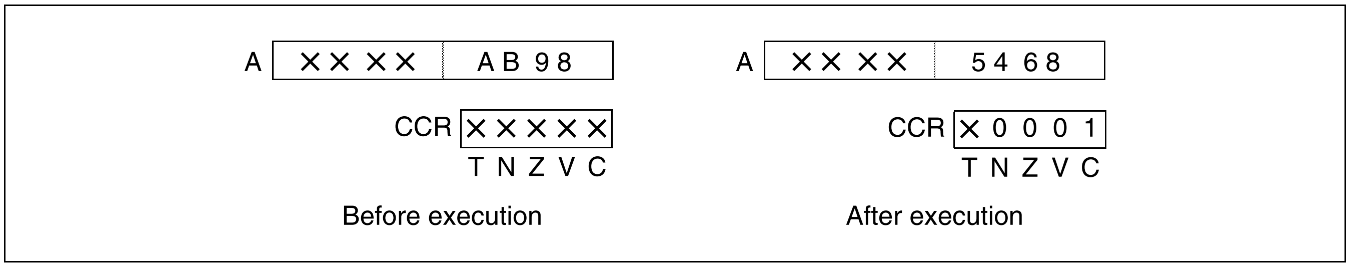
- Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	1	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

- Example:

NEGW A



9.1.92 NOP (No Operation)

Perform no operation.

■ NOP (No Operation)

- Assembler format:

NOP

- Operation:

No operation is performed.

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

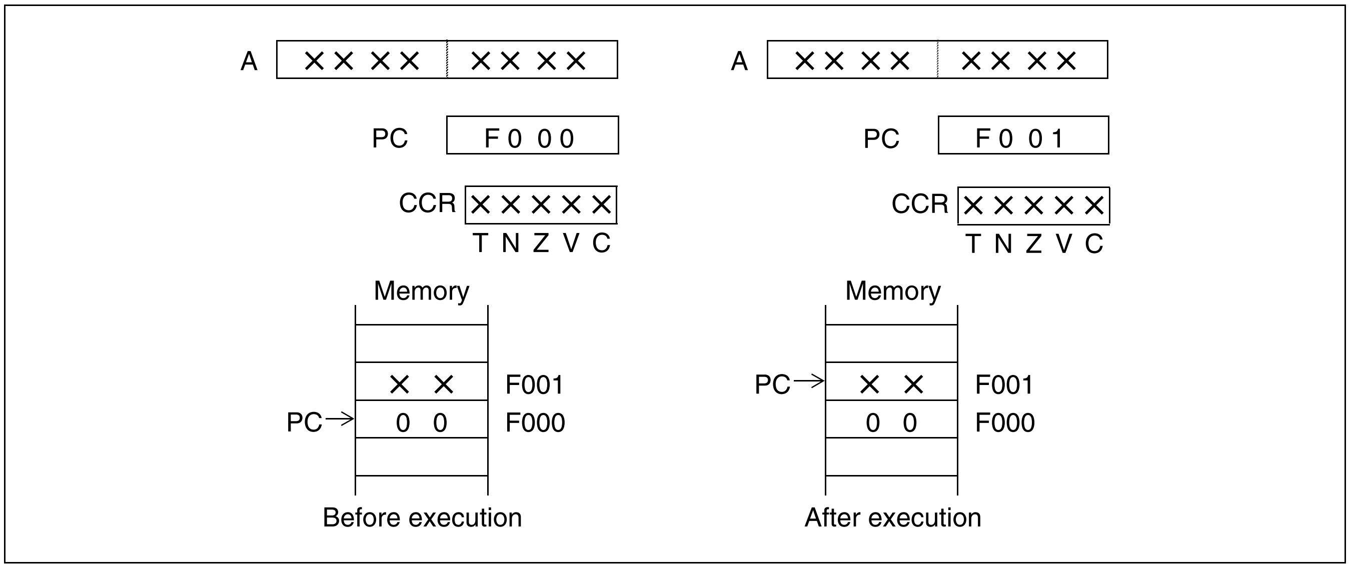
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	1
Correction value:	0

- Example:

NOP



9.1.93 NOT (Not Byte Data of Destination)

Take the logical NOT of the byte data specified by the operand and restore the result in the operand.

■ NOT (Not Byte Data of Destination)

● Assembler format:

NOT A

NOT ear NOT eam

● Operation:

(Operand) ← not (Operand) (Byte logical NOT)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

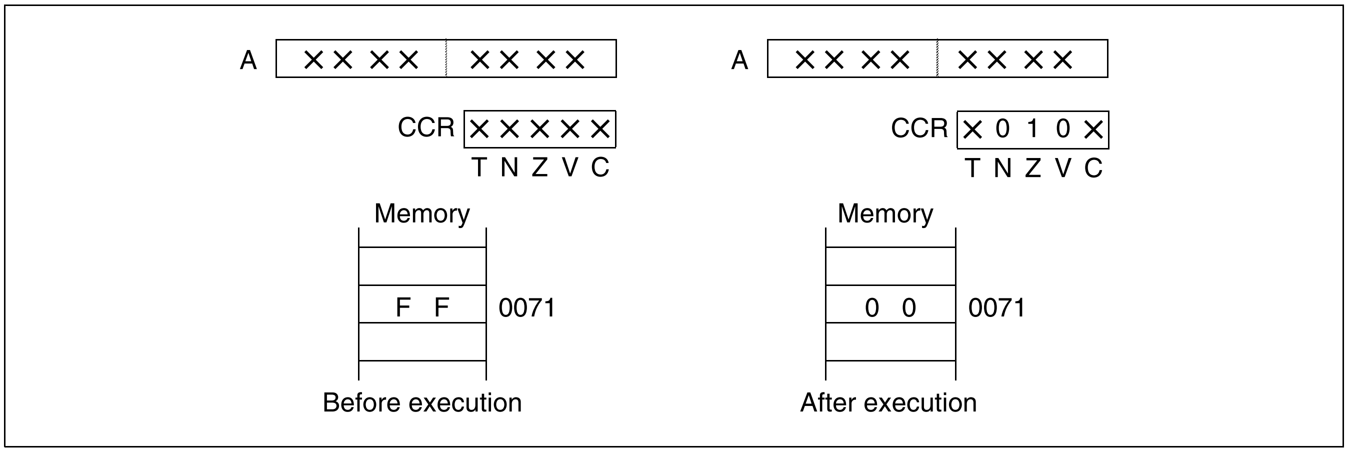
● Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	1	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

NOT 0071H



9.1.94 NOTW (Not Word Data of Destination)

Take the logical NOT of the word data specified by the operand and restore the result in the operand.

■ NOTW (Not Word Data of Destination)

● Assembler format:

NOTW A

NOTW ear NOTW eam

● Operation:

(Operand) ← not (Operand) (Word logical NOT)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

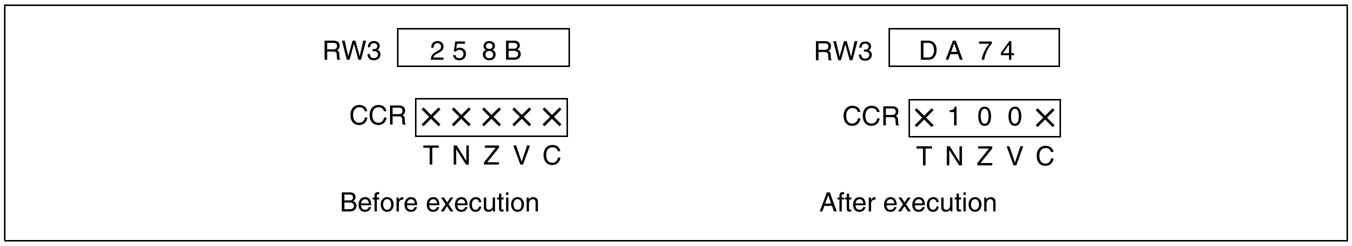
● Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	1	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

NOTW RW3



9.1.95 NRML (NORMALIZE Long Word)

Shift the long word data of the accumulator (A) to the left until the most significant bit of A becomes "1", if the long word data is not "0". R0 is set to the number of shifts required and the zero flag (Z) is cleared.
If the long word data of the accumulator (A) is "0", R0 is set to "0" and the zero flag (Z) is set.

■ NRML (NORMALIZE Long Word)

● Assembler format:

NRML A,R0

● Operation:

If A≠0: The long word data is shifted to the left until the most significant bit of A becomes 1.
(R0) ← Number of shifts required, Z ← 0

If A=0: (R0) ← 0, Z ← 1

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	*	–	–

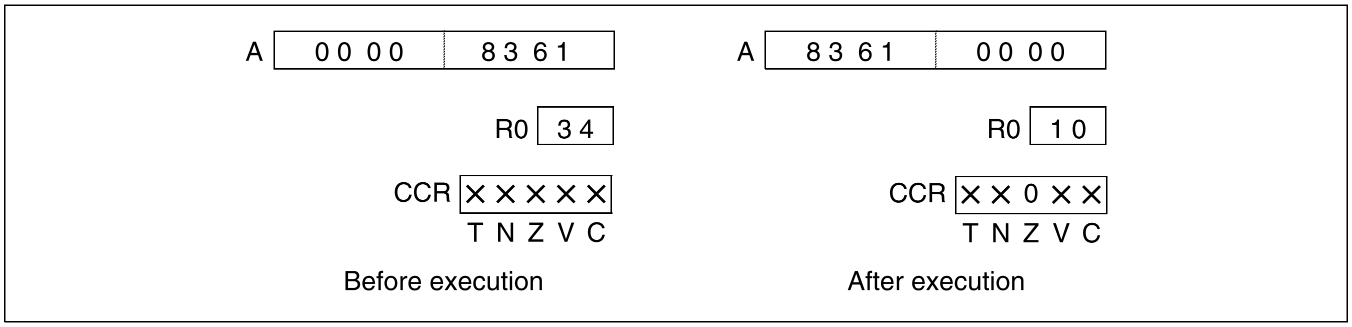
I, S, T, and N:	Unchanged
Z:	Set when A is equal to "0", cleared otherwise.
V and C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	4 when the accumulator is equal to "0"; otherwise, 6 + (Number of shifts required)
Correction value:	0

● Example:

NRML A,R0



9.1.96 OR (Or Byte Data of Destination and Source to Destination)

Take the logical OR of the byte data specified by the first operand and the byte data specified by the second operand and restore the result in the first operand.

■ OR (Or Byte Data of Destination and Source to Destination)

● Assembler format:

OR A,#imm8	
OR A,ear	OR A,eam
OR ear,A	OR eam,A

● Operation:

(First operand) ← (First operand) or (Second operand) (Byte logical OR)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

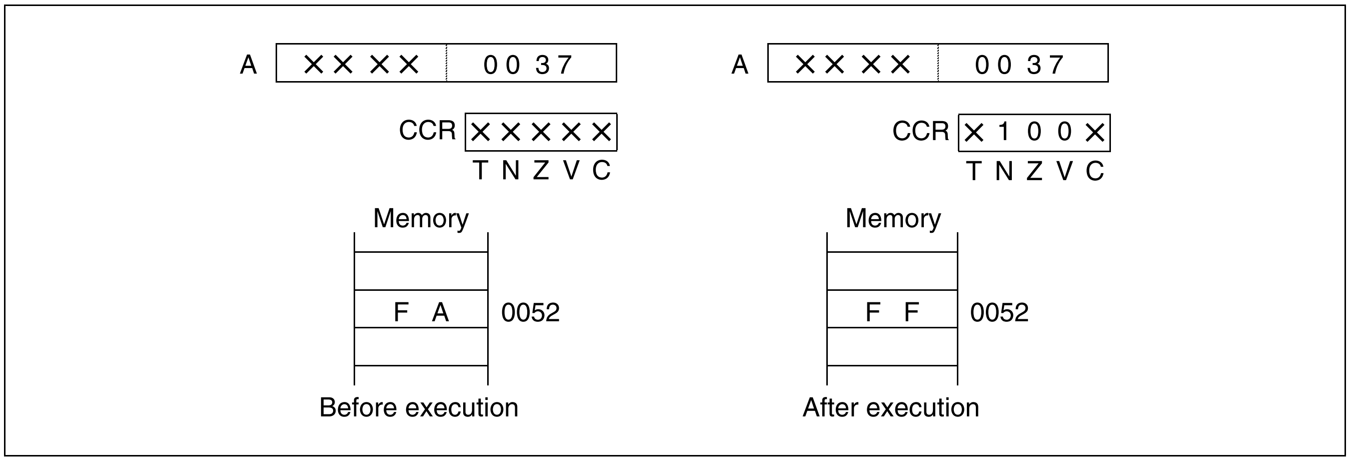
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	ear	eam
Second operand	#im8	ear	eam	A	A
Number of bytes	2	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

OR 0052H,A



9.1.97 OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register)

Take the logical OR of the byte data in the condition code register (CCR) and specified 8-bit immediate data and restore the result in the condition code register (CCR).
Bit 7 of the immediate data is ignored because the condition code register (CCR) is 7 bits long.

■ OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register)

● Assembler format:

OR CCR,#imm8

● Operation:

(CCR) ← (CCR) or #imm8 (Byte logical OR)

● CCR:

I	S	T	N	Z	V	C
*	*	*	*	*	*	*

I: Stores bit 6 of the operation result.

S: Stores bit 5 of the operation result.

T: Stores bit 4 of the operation result.

N: Stores bit 3 of the operation result.

Z: Stores bit 2 of the operation result.

V: Stores bit 1 of the operation result.

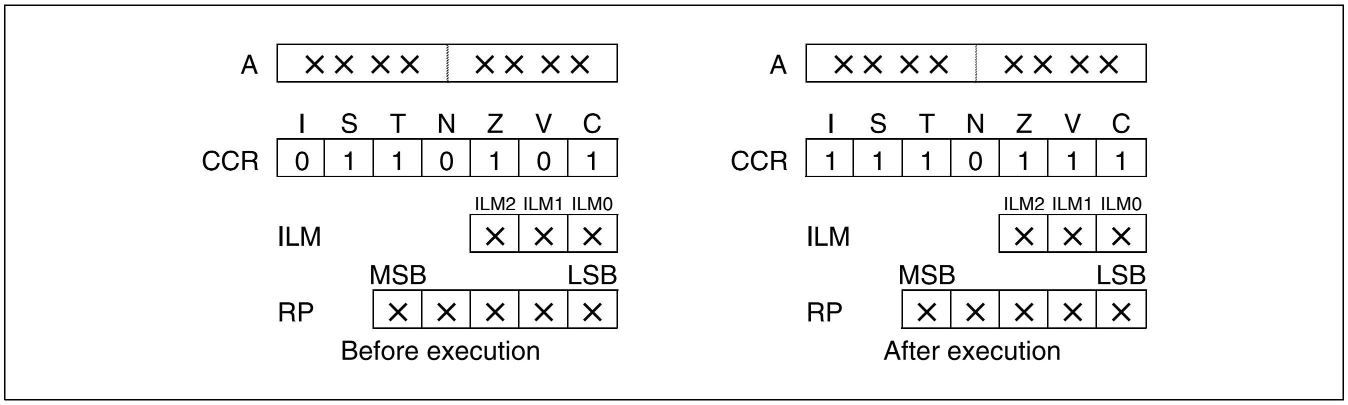
C: Stores bit 0 of the operation result.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	3
Correction value:	0

● Example:

OR CCR,#57H



9.1.98 ORL (Or Long Word Data of Destination and Source to Destination)

Take the logical OR of the long word data for the accumulator (A) and that specified by the second operand and restore the result in A.

■ ORL (Or Long Word Data of Destination and Source to Destination)

- Assembler format:

  ORL A,ear ORL A,eam

- Operation:

  (A) ← (A) or (Second operand) (Long word logical OR)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

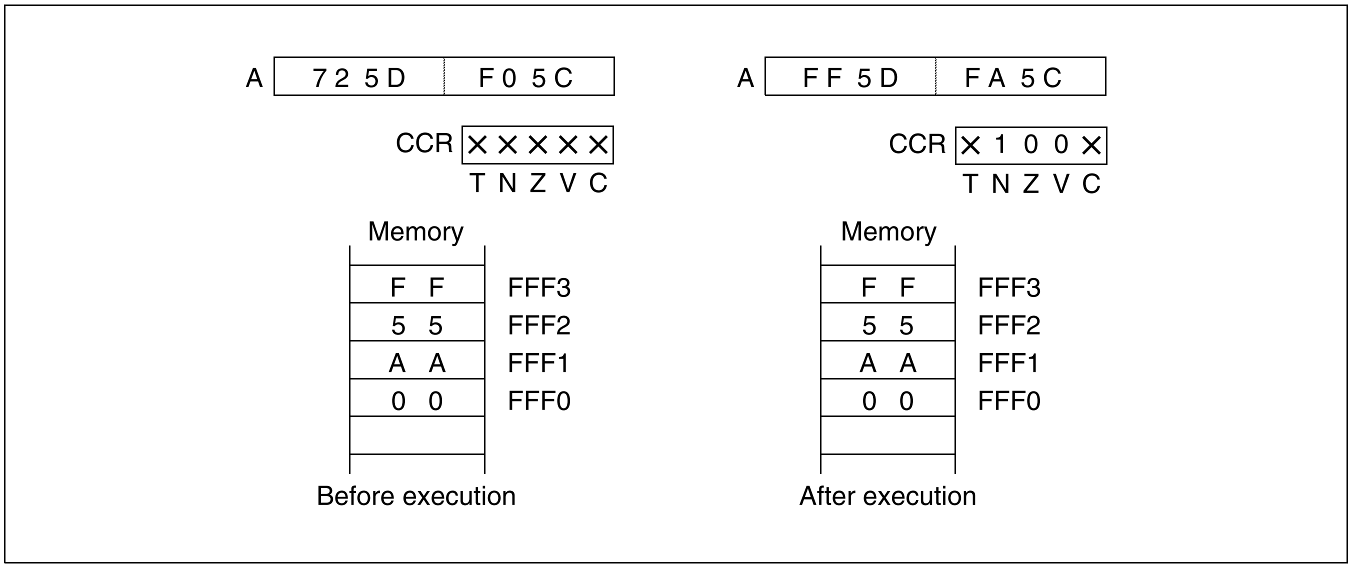
- Number of bytes, Number of cycles, and Correction value:

First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	6	7+(a)
Correction value	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ORL A,0FFF0H



9.1.99 ORW (Or Word Data of AH and AL to AL)

Take the logical OR of the word data for AH and that for AL and restore the result in AL.

■ ORW (Or Word Data of AH and AL to AL)

● Assembler format:

ORW A

● Operation:

(AL) ← (AH) or (AL)   (Word logical OR)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

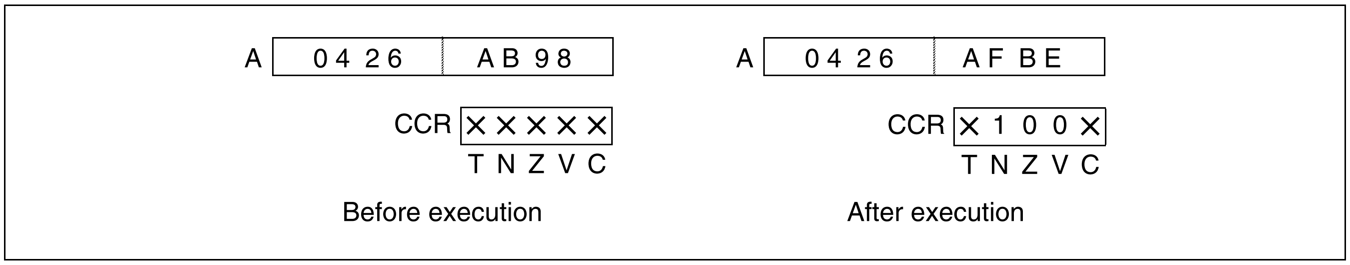
I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

ORW A



9.1.100 ORW (Or Word Data of Destination and Source to Destination)

Take the logical OR of the word data specified by the first operand and the word data specified by the second operand and restore the result in the first operand.

■ ORW (Or Word Data of Destination and Source to Destination)

- Assembler format:

ORW A,#imm16	
ORW A,ear	ORW A,eam
ORW ear,A	ORW eam,A

- Operation:

(First operand) ← (First operand) or (Second operand) (Word logical OR)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

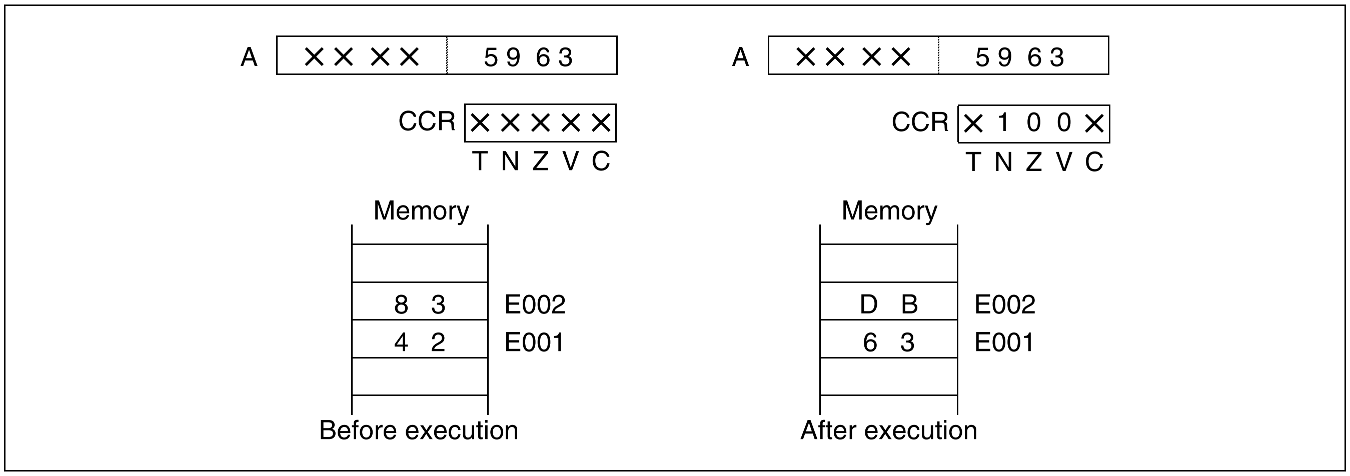
- Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	ear	eam
Second operand	#i16	ear	eam	A	A
Number of bytes	3	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ORW 0E001H,A



9.1.101 POPW (Pop Word Data of Accumulator from Stack Memory)

Transfer the values of bits 0 to 15 for the accumulator (A) to bits 16 to 31. Then, the word data of the memory location pointed to by the stack pointer (SP) is transferred to bits 0 to 15 of A. After the data is transferred, 0002_H is word-added to the value of SP (word data).

■ POPW (Pop Word Data of Accumulator from Stack Memory)

- Assembler format:

POPW A

- Operation:

(A) ←	((SP))	(Word transfer)
(SP) ←	(SP)+2	(Word addition)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

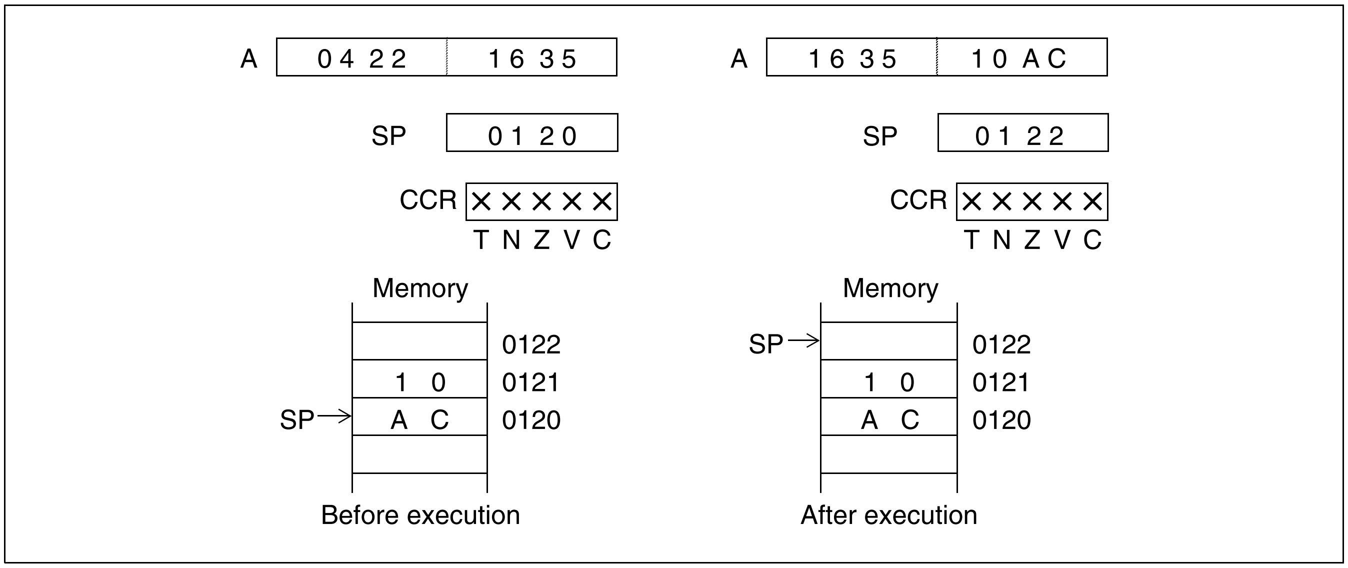
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3
Correction value:	(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

POPW A



9.1.102 POPW (Pop Word Data of AH from Stack Memory)

Transfer word data from the memory location pointed to by the stack pointer (SP) to AH. Then, 0002_H is word-added to the value of SP (word data).

■ POPW (Pop Word Data of AH from Stack Memory)

- Assembler format:

  POPW AH

- Operation:

  (AH) ← ((SP)) (Word transfer)

  (SP) ← (SP)+2 (Word addition)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

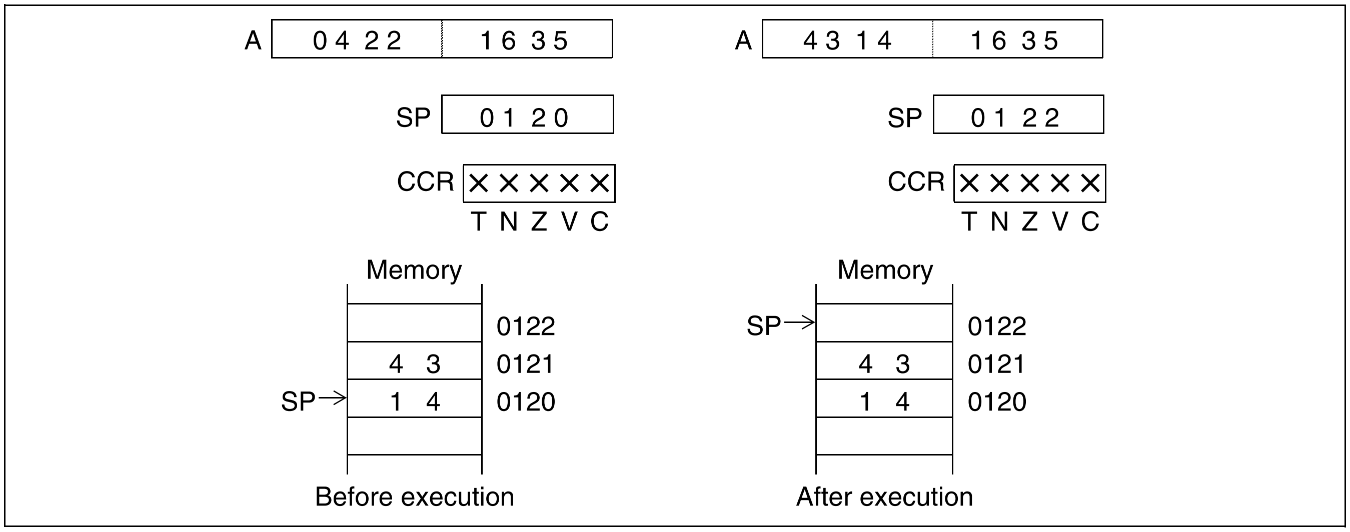
- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3
Correction value:	(c)

For the explanation of (c), see Table 8.4-2 .

- Example:

  POPW AH



9.1.103 POPW (Pop Word Data of Program Status from Stack Memory)

Transfer word data from the memory location pointed to by the stack pointer (SP) to the processor status (PS). Bit 7 of the word data is ignored. Then, 0002_H is word-added to the value of SP (word data).

■ POPW (Pop Word Data of Program Status from Stack Memory)

● Assembler format:

POPW PS

● Operation:

(PS) ← ((SP)) (Word transfer)

(SP) ← (SP)+2 (Word addition)

● CCR:

I	S	T	N	Z	V	C
*	*	*	*	*	*	*

The values of the corresponding bits for the stack memory are transferred.

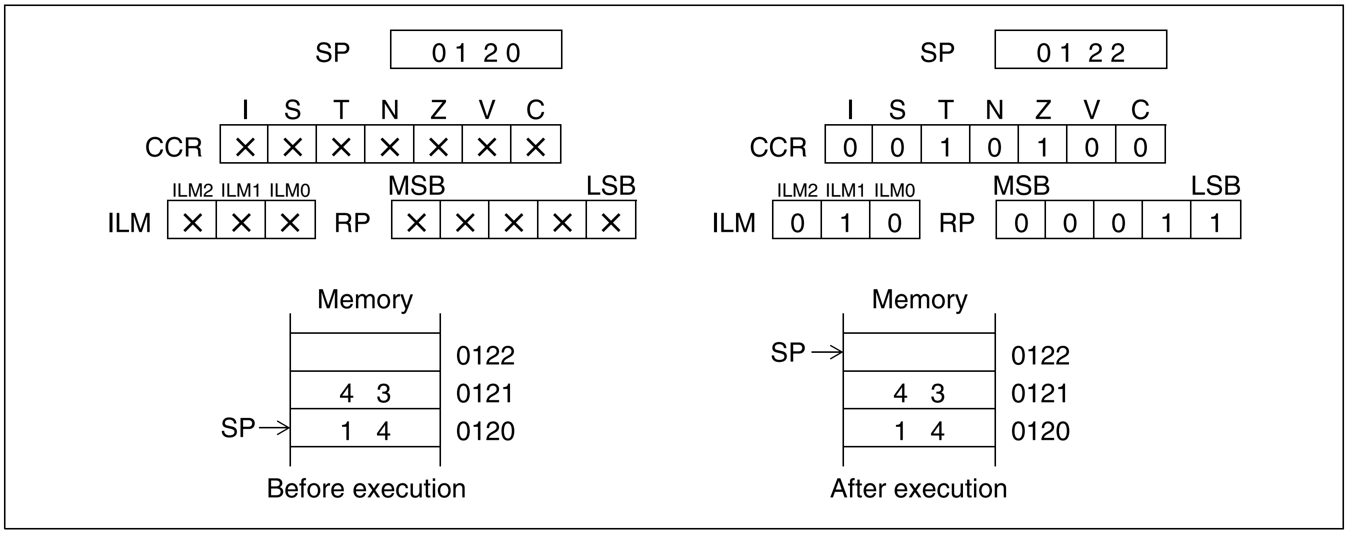
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	4
Correction value:	(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

POPW PS



9.1.104 POPW (Pop Registers from Stack Memory)

Transfer the data pointed to by the stack pointer (SP) to the multiple general-purpose word registers specified by the register list (rlst).
In assembler representation, register names are enumerated as a register list. After assembly, the register list turns into byte data.

■ POPW (Pop Registers from Stack Memory)

● Assembler format:

POPW rlst

● Operation:

(RWx) ← ((SP))	(Word transfer)
(SP) ← (SP)+2	(Word addition)

The above operation is repeated for all the registers specified by rlst.

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

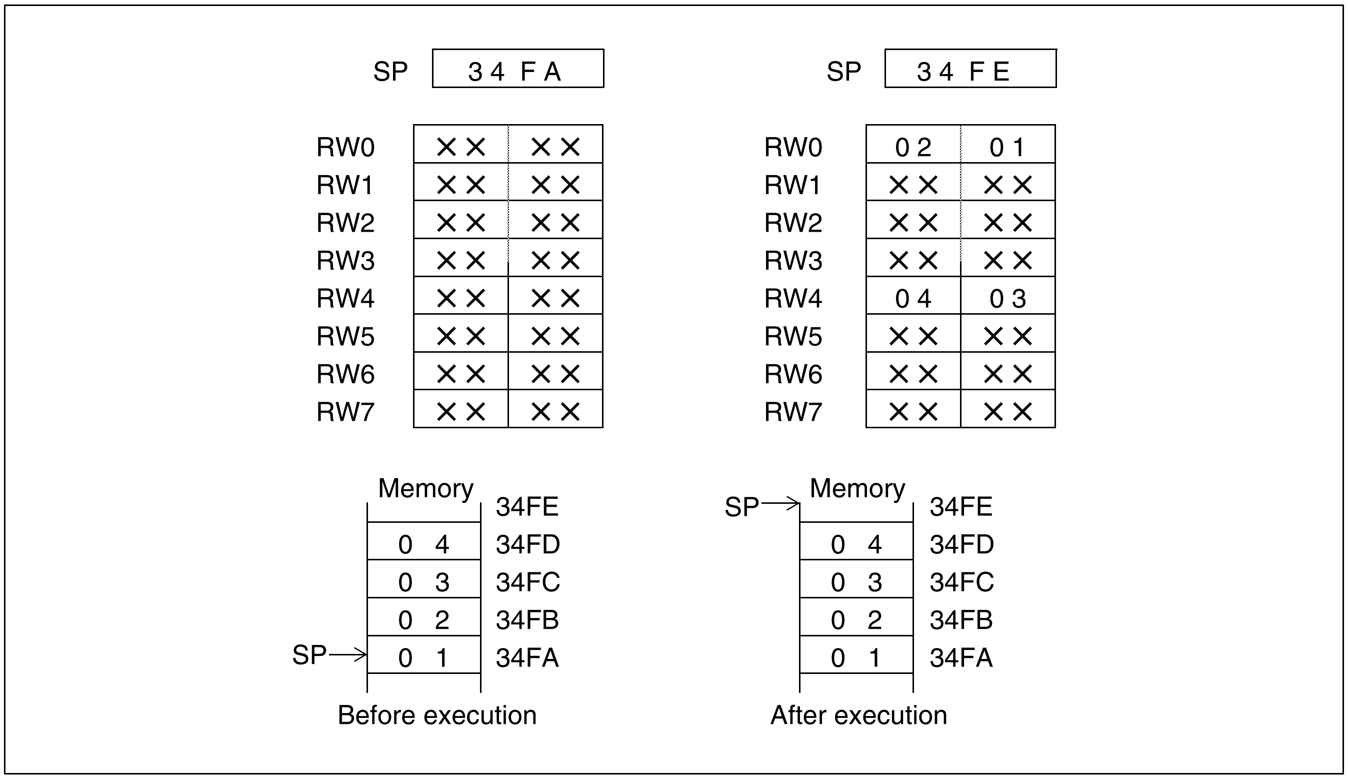
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	7 + 3 × (Number of transfers) + 2 × (Largest number in the transferred registers) 7 if rlst=0
Correction value:	(Number of transfers)×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

POPW (RW0,RW4)



9.1.105 PUSHW (Push Word Data of Inherent Register to Stack Memory)

Decrement the value of the stack pointer (SP) by two words and transfer the word data of the register to the memory location pointed to by the resulting SP value.

■ PUSHW (Push Word Data of Inherent Register to Stack Memory)

- Assembler format:

  PUSHW A

  PUSHW AH

  PUSHW PS

- Operation:

(SP) ← (SP)–2	(Word subtraction)
((SP)) ← (Operand)	(Word transfer)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

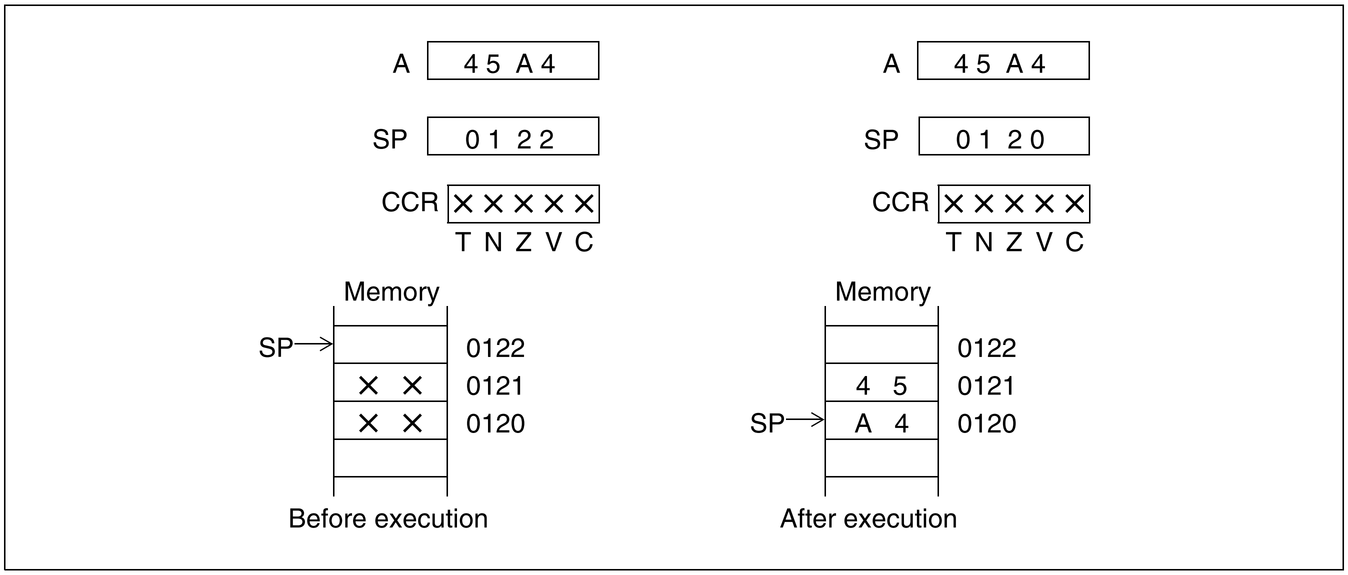
- Number of bytes, Number of cycles, and Correction value:

Operand	A	AH	PS
Number of bytes	1	1	1
Number of cycles	4	4	4
Correction value	(c)	(c)	(c)

For the explanation of (c) in the table, see Table 8.4-2 .

● Example:

PUSHW A



9.1.106 PUSHW (Push Registers to Stack Memory)

Transfer the contents of the multiple general-purpose word registers specified by the register list (rlst) to the memory location pointed to by the stack pointer (SP).
In assembler representation, register names are enumerated as a register list. After assembly, the register list turns into byte data.

■ PUSHW (Push Registers to Stack Memory)

● Assembler format:

PUSHW rlst

● Operation:

(SP) ← (SP)–2	(Word subtraction)
((SP))← (RWx)	(Word transfer)

The above operation is repeated for all the registers specified by rlst.

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

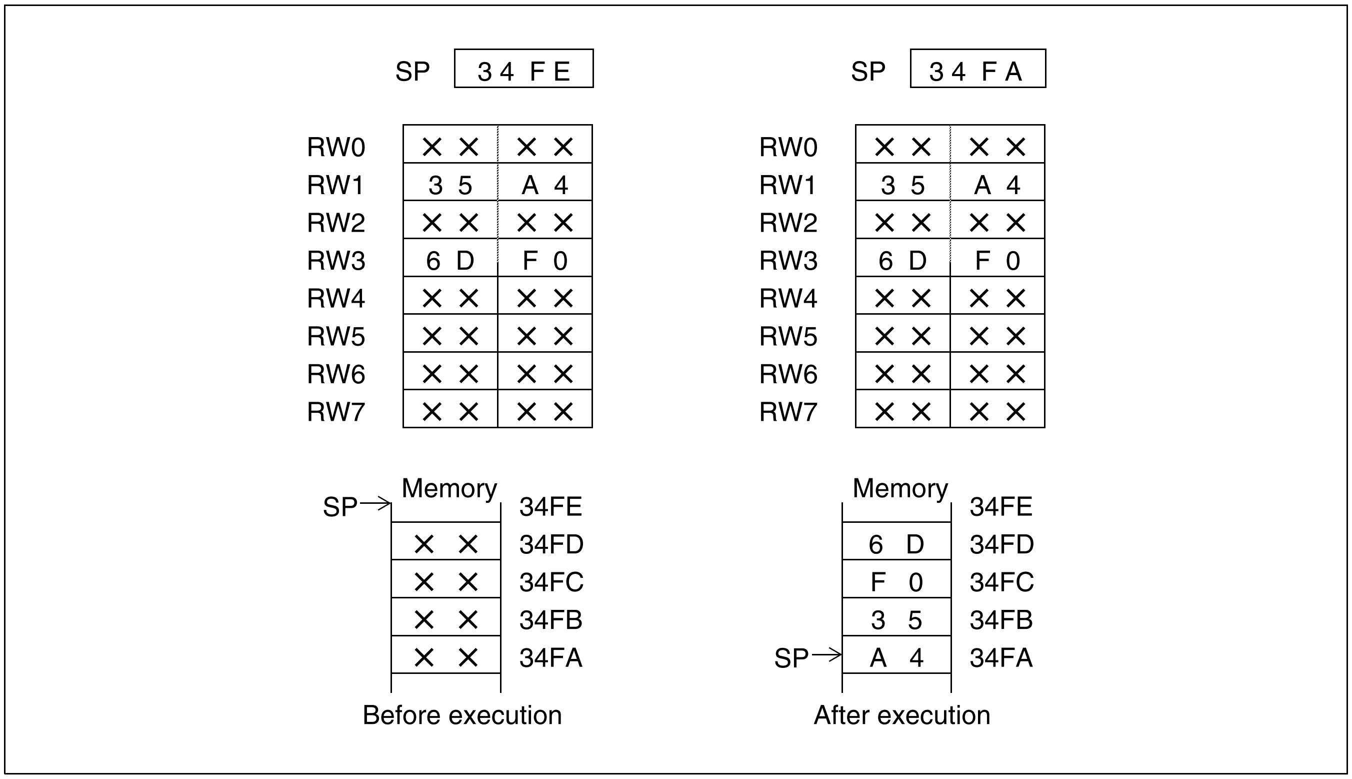
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	29 + 3 × (Number of transfers) – 3 × (8 – Smallest number of the transferred registers) 8 if rlst = 0
Correction value:	(Number of transfers)×(c)

For the explanation of (c), see Table 8.4-2 .

● Example:

PUSHW (RW1,RW3)



9.1.107 RET (Return from Subroutine)

Cause a branch to the address pointed to by the stack pointer (SP). If this instruction is used in combination with a subroutine call instruction (CALL, CALLV), control returns to the instruction following the subroutine call instruction after the branch operation is completed.

■ RET (Return from Subroutine)

- Assembler format:

  RET

- Operation:

  (PC) ← ((SP)) (Word transfer)

  (SP) ← (SP)+2 (Word addition)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

  Number of bytes: 1

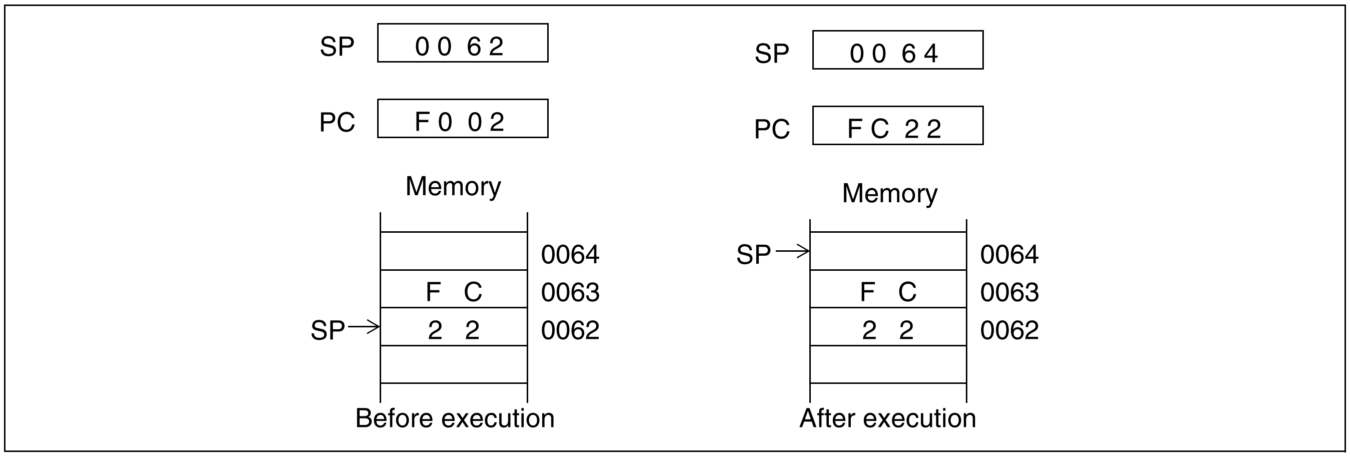
  Number of cycles: 4

  Correction value: (c)

  For the explanation of (c), see Table 8.4-2 .

- Example:

  RET



9.1.108 RETI (Return from Interrupt)

This instruction returns the data in the memory that is indicated by (SSP) to PS to detect interrupt requests performed using IF or ILM.
When the next interrupt request is received, the procedure branches to the detected interruption vector. If no next interrupt is received, the procedure will return from the interruption process.

■ RETI (Return from Interrupt)

● Assembler format:

RETI

● Operation:

(1) If the next interrupt is accepted

(PS)	←	((SSP))
(S)	←	1, (PCB), (PC) ← Interrupt vector address
(ILM)	←	Accepted interrupt level

DTB, PCB, DPR, ADB, AL, and AH are not restored.

(2) If control is returned from the next interrupt

(PS)	←	((SSP)), (SSP)	←	(SSP)+2;
(PC)	←	((SSP)), (SSP)	←	(SSP)+2;
(DTB),(PCB)	←	((SSP)), (SSP)	←	(SSP)+2;
(DPR),(ADB)	←	((SSP)), (SSP)	←	(SSP)+2;
(AL)	←	((SSP)), (SSP)	←	(SSP)+2;
(AH)	←	((SSP)), (SSP)	←	(SSP)+2

● CCR

(1) If the next interrupt is accepted

I	S	T	N	Z	V	C
*	S	*	*	*	*	*

I: Restored to the saved I value.
S: Set
T: Restored to the saved T value.
N: Restored to the saved N value.
Z: Restored to the saved Z value.
V: Restored to the saved V value.
C: Restored to the saved C value.

(2) If control is returned from the next interrupt

I	S	T	N	Z	V	C
*	*	*	*	*	*	*

I: Restored to the saved I value.
S: Restored to the saved S value.
T: Restored to the saved T value.
N: Restored to the saved N value.
Z: Restored to the saved Z value.
V: Restored to the saved V value.
C: Restored to the saved C value.

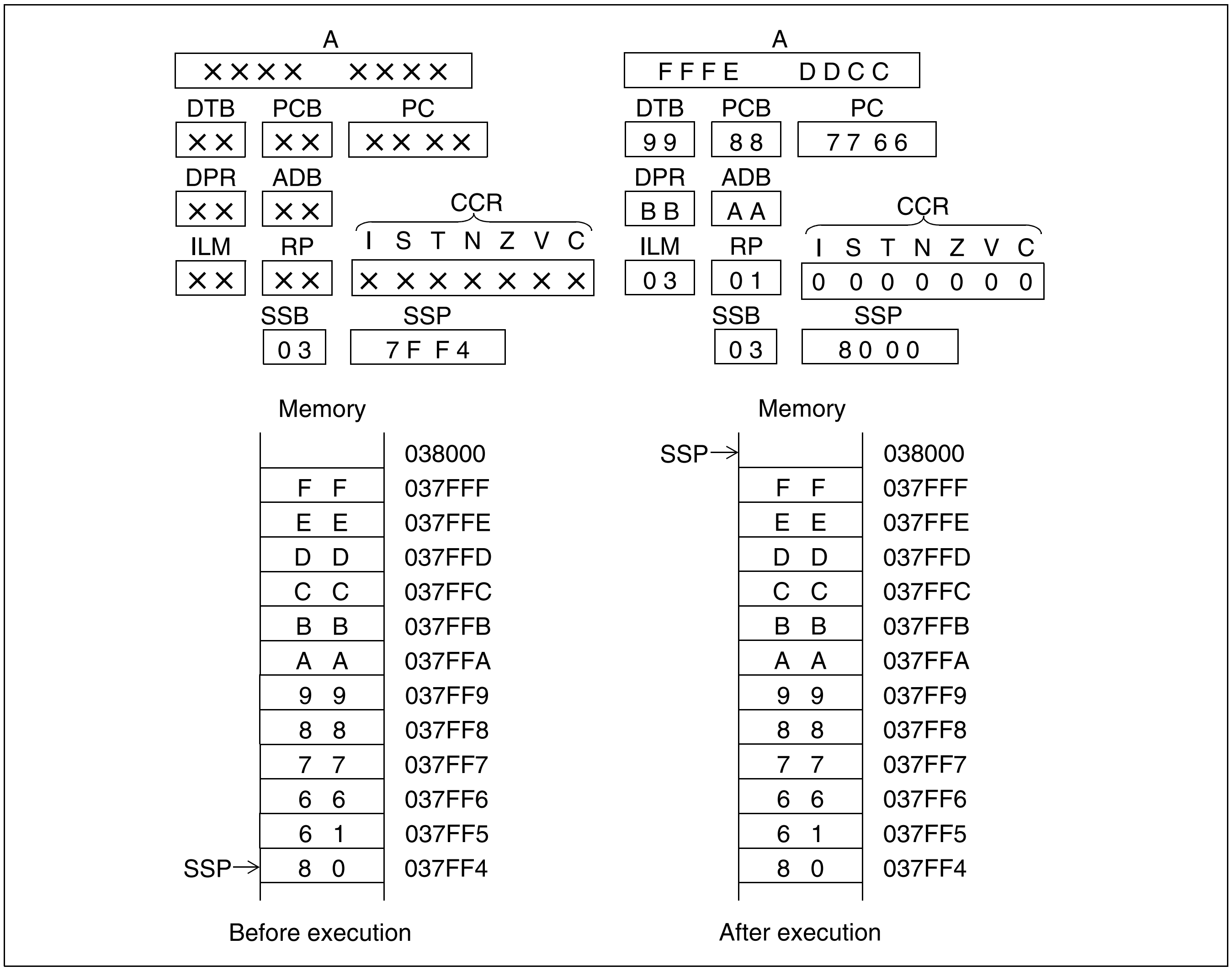
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	15 if the next interrupt is accepted; 17 if control is returned from the next interrupt
Correction value:	3 × (b) + 2 × (c) if the next interrupt is accepted; 6 × (c) if control is returned from the next interrupt

For the explanation of (b) and (c), see Table 8.4-2 .

● Example:

RETI (if control is returned from the interrupt)



9.1.109 RETP (Return from Physical Address)

Cause a branch to the physical address pointed to by the stack pointer (SP). If this instruction is used in combination with the CALLP instruction, control returns to the instruction following the CALLP instruction after the branch operation is completed.

■ RETP (Return from Physical Address)

● Assembler format:

RETP

● Operation:

(PC) ← ((SP)), (SP) ← (SP)+2	(Word addition)
(PCB) ← ((SP)) (Byte transfer), (SP) ← (SP)+2	(Word addition)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

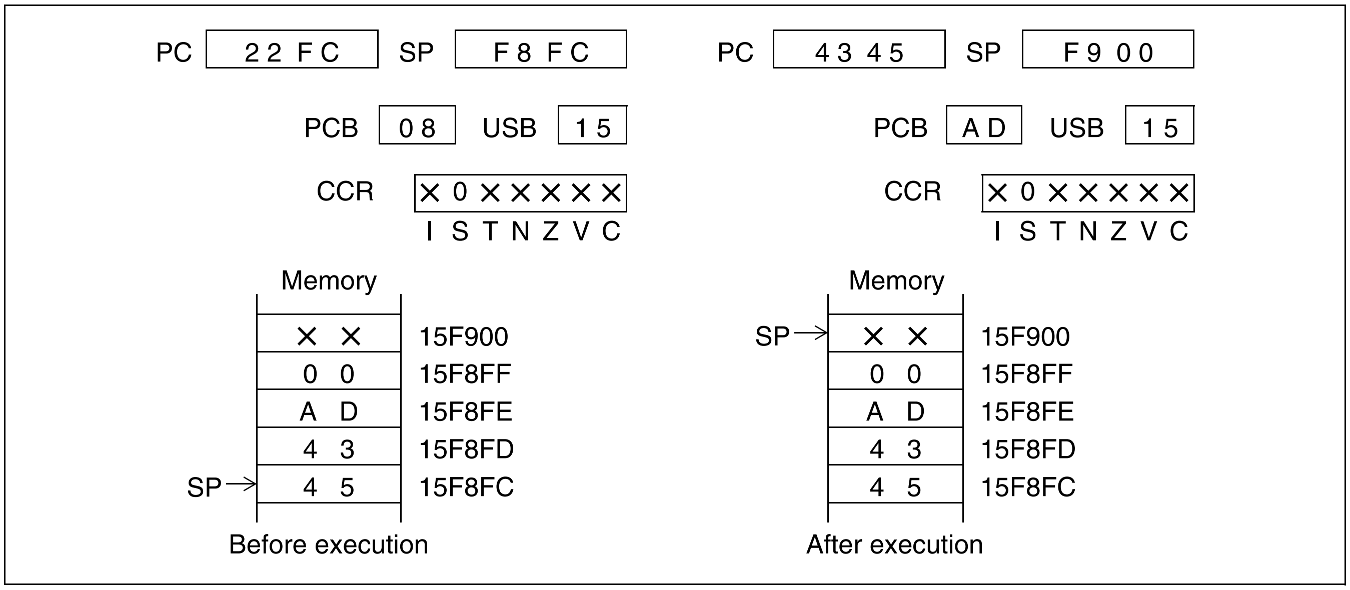
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	6
Correction value:	(d)

For the explanation of (d), see Table 8.4-2 .

● Example:

RETP



9.1.110 ROLC (Rotate Byte Data of Accumulator with Carry to Left)

Rotate or shift the byte data specified by the operand to the left by one bit, including the carry bit (C). The most significant bit of the operand is placed in the carry bit (c).

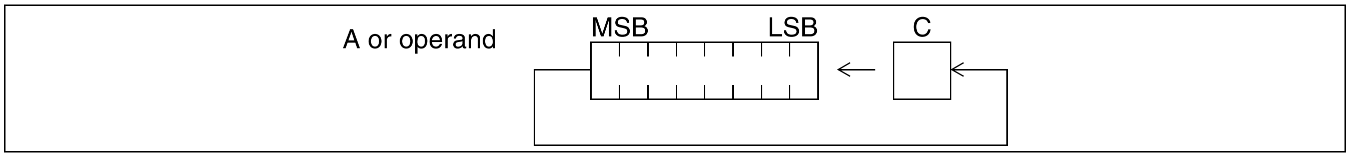
■ ROLC (Rotate Byte Data of Accumulator with Carry to Left)

● Assembler format:

ROLC A

ROLC ear ROLC eam

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit shifted out from the MSB of A.

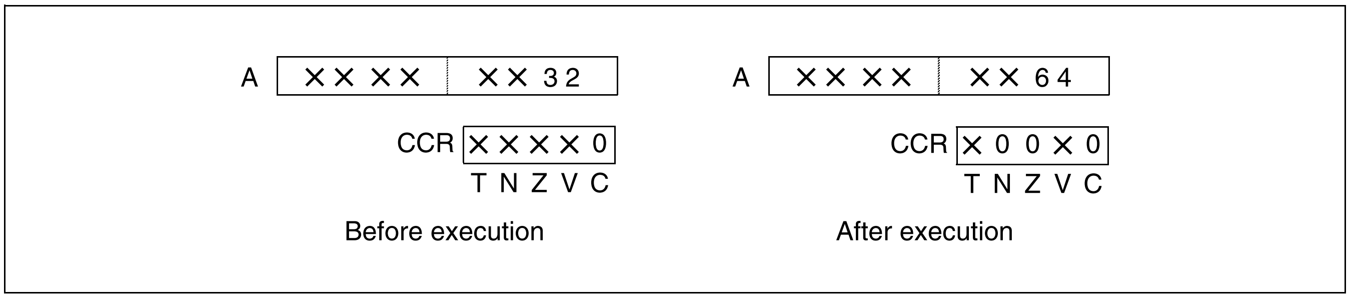
● Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	2	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

ROLC A



9.1.111 RORC (Rotate Byte Data of Accumulator with Carry to Right)

Rotate or shift the byte data specified by the operand to the right by one bit, including the carry bit (C). The least significant bit of the operand is placed in the carry bit (c).

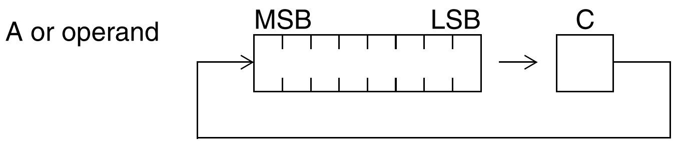
■ RORC (Rotate Byte Data of Accumulator with Carry to Right)

● Assembler format:

RORC A

RORC ear RORC eam

● Operation:



● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	–	*

I, S, and T:	Unchanged
N:	Set when the MSB of the shifting result is "1", cleared otherwise.
Z:	Set when the shifting result is "0", cleared otherwise.
V:	Unchanged
C:	Stores the bit shifted out from the LSB of A.

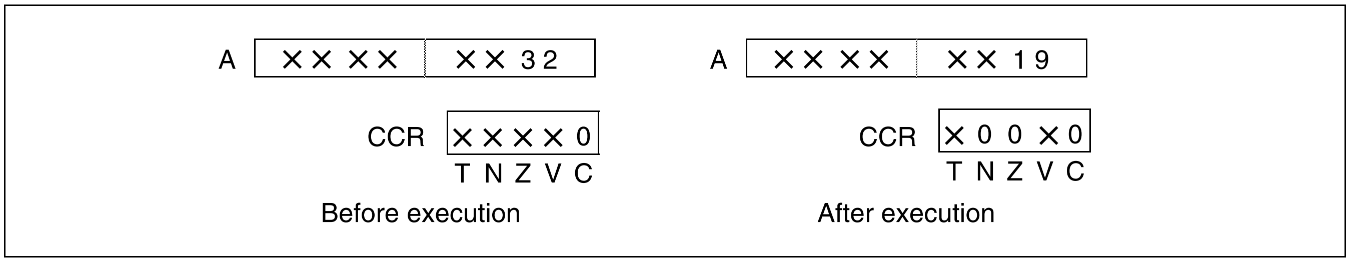
● Number of bytes, Number of cycles, and Correction value:

Operand	A	ear	eam
Number of bytes	2	2	2+
Number of cycles	2	3	5+(a)
Correction value	0	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

RORC A



9.1.112 SBBS (Set Bit and Branch if Bit Set)

Cause a branch if the bit data specified by the first operand is "1". Control is transferred to the address resulting from word-adding the value resulting from sign-extending the second operand to the address of the instruction following the SBBS instruction.
After the instruction has been executed, the bit specified by the first operand is set to "1".

■ SBBS (Set Bit and Branch if Bit Set)

● Assembler format:

SBBS addr16:bp,rel

● Operation:

If the condition is satisfied:

(PC) ← (PC)+<Number of bytes>+rel (Word addition), (addr16:bp) ← 1

If the condition is not satisfied:

(PC) ← (PC)+<Number of bytes> (Word addition), (addr16:bp) ← 1

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	*	–	–

I, S, T, and N:	Unchanged
Z:	Set when the bit data is "0", cleared otherwise.
V and C:	Unchanged

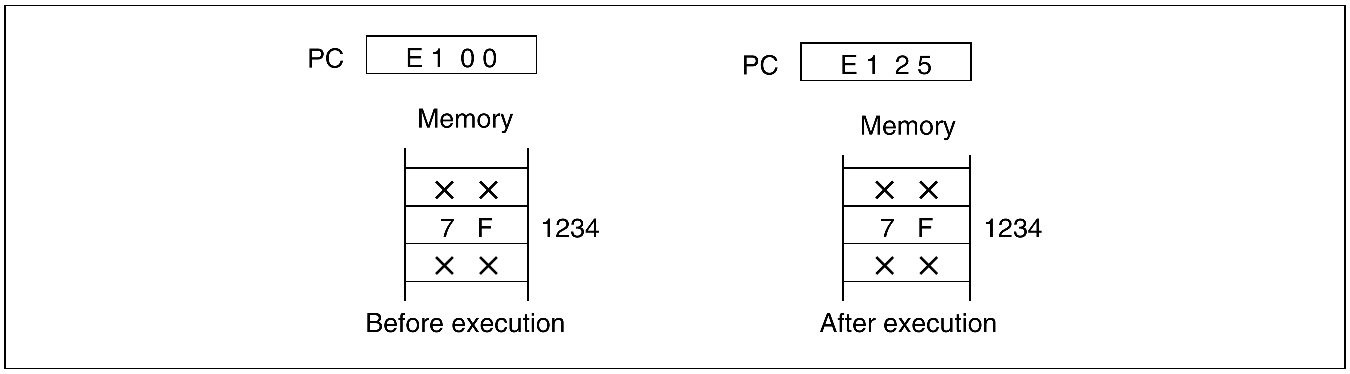
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	5
Number of cycles:	9 if the condition is not satisfied; 10 if the condition is satisfied
Correction value:	2×(b)

For the explanation of (b), see Table 8.4-2 .

● Example:

SBBS 1234H:5,20H



9.1.113 SCEQ (Scan String Byte Until Equal)

Compare the byte data specified by AH in the space specified by <bank> with the data of AL. The address is incremented/decremented and RW0 is decremented until the byte data matches the data or RW0 becomes equal to "0".
Four types of registers PCB, DTB, ADB, and SPB can be specified by <bank>. By default, DTB is assumed. The address can be either incremented or decremented. By default, the address is incremented.
If RW0 is equal to "0", comparison is not performed. If an interrupt occurs during the execution of the instruction, the execution of the instruction is suspended to handle the interrupt. After the interrupt has been handled, the execution of the instruction is resumed.

■ SCEQ (Scan String Byte Until Equal)

● Assembler format:

SCEQ [<bank>]   SCEQI [<bank>]   (When the address is incremented)

SCEQD [<bank>]   (When the address is decremented)

● Operation:

The following operation is repeated until RW0 = 0 or ((AH)) = (AL)   (Byte comparison):

(AH) ← (AH)±1

(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", N is set when the MSB of the last compare operation result is "1", cleared otherwise.
Z:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", Z is set when a match with the contents of AL is found; cleared when the instruction terminates with RW0 being set to "0".
V:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", V is set when an overflow has occurred as a result of the last compare operation; cleared otherwise.
C:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", V is set when a borrow has occurred as a result of the last compare operation; cleared otherwise.

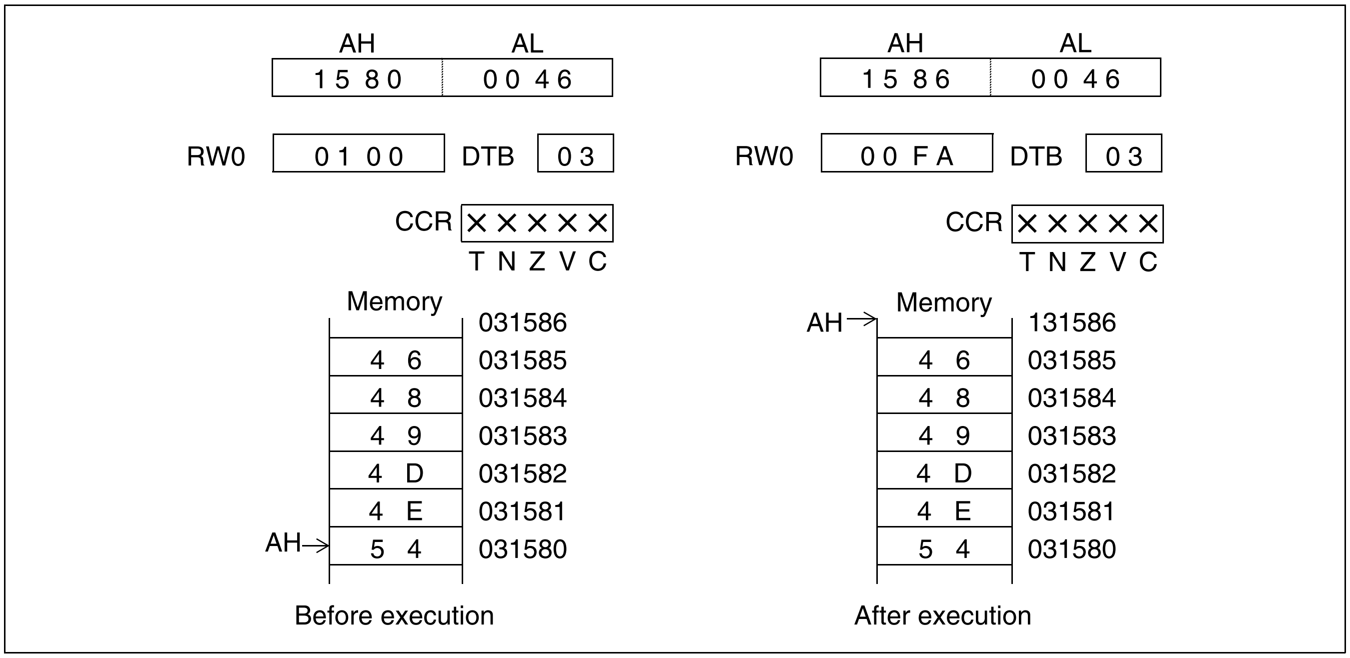
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	5 when RW0 is "0", 4 + 7 × (RW0) when count-out is detected, and 7n + 5 when the data in the AL register matches the byte data specified by the AH register in the space that is specified by bank
Correction value:	(Number of times the operation was repeated)×(b)

For the explanation of (b), see Table 8.4-2 .

● Example:

SCEQ



9.1.114 SCWEQ (Scan String Word Until Equal)

Compare the word data specified by AH in the space specified by <bank> with the data of AL. The address is incremented/decremented and RW0 is decremented until the word data matches the data or RW0 becomes equal to "0".
Four types of registers PCB, DTB, ADB, and SPB can be specified by <bank>. By default, DTB is assumed. The address can be either incremented or decremented. By default, the address is incremented.
If RW0 is equal to "0", comparison is not performed. If an interrupt occurs during the execution of the instruction, the execution of the instruction is suspended to handle the interrupt. After the interrupt has been handled, the execution of the instruction is resumed.

■ SCWEQ (Scan String Word until Equal)

● Assembler format:

SCWEQ [<bank>]   SCWEQI [<bank>] (When the address is incremented)

SCWEQD [<bank>]   (When the address is decremented)

● Operation:

The following operation is repeated until RW0 = 0 or ((AH)) = (AL)   (Word comparison):

(AH) ← (AH)±2

(RW0) ← (RW0)–1

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", N is set when the MSB of the last compare operation result is "1", cleared otherwise.
Z:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", Z is set when a match with the contents of AL is found; cleared when the instruction terminates with RW0 being set to "0".
V:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", V is set when an overflow has occurred as a result of the last compare operation; cleared otherwise.
C:	Unchanged if the initial value of RW0 is "0". If the initial value of RW0 is not "0", V is set when a borrow has occurred as a result of the last compare operation; cleared otherwise.

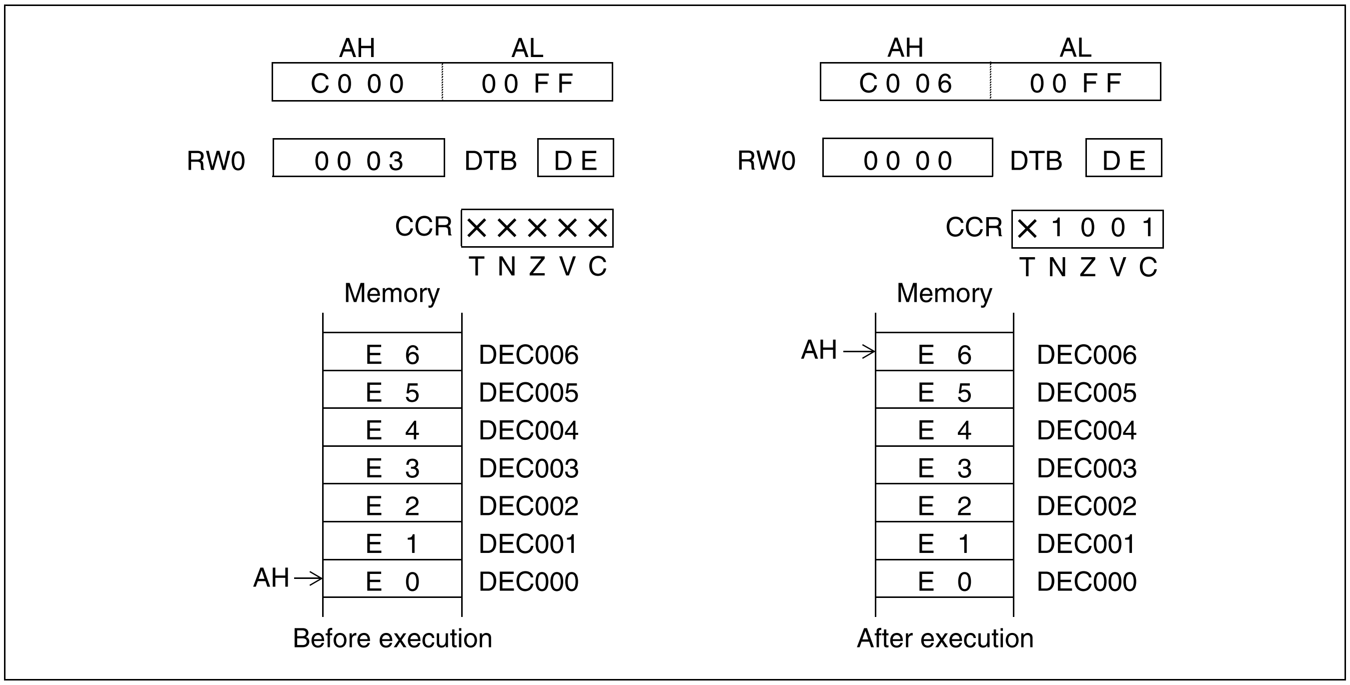
● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	2
Number of cycles:	5 when RW0 is "0", 4 + 7 × (RW0) when count-out is detected, and 7n + 5 when the data in the AL register matches the byte data specified by the AH register in the space that is specified by bank
Correction value:	(Number of times the operation was repeated) × (c)

For the explanation of (c), see Table 8.4-2 .

● Example:

SCWEQ



9.1.115 SETB (Set Bit)

Set the contents of the bit address specified by the operand to "1".

■ SETB (Set Bit)

● Assembler format:

SETB addr16:bp

SETB dir:bp

SETB io:bp

● Operation:

(Operand) b ← 1 (Bit transfer)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

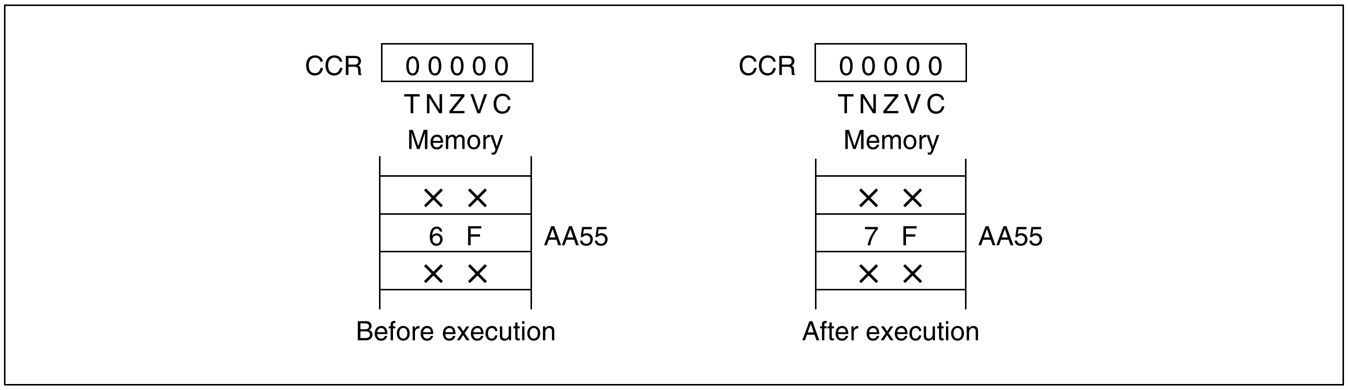
● Number of bytes, Number of cycles, and Correction value:

Operand	ad16:bp	dir:bp	io:bp
Number of bytes	4	3	3
Number of cycles	7	7	7
Correction value	2×(b)	2×(b)	2×(b)

For the explanation of (b) in the table, see Table 8.4-2 .

● Example:

SETB 0AA55H:4



9.1.116 SUB (Subtract Byte Data of Source from Destination to Destination)

Subtract the byte data specified by the second operand from the byte data specified by the first operand and restore the result in the first operand. If the first operand is A, "0" is transferred to bits 8 to 15 of A.

■ SUB (Subtract Byte Data of Source from Destination to Destination)

● Assembler format:

SUB A,#imm8	SUB A,dir
SUB A,ear	SUB A,eam
SUB ear,A	SUB eam,A

● Operation:

(First operand) ← (First operand)–(Second operand) (Byte subtraction)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

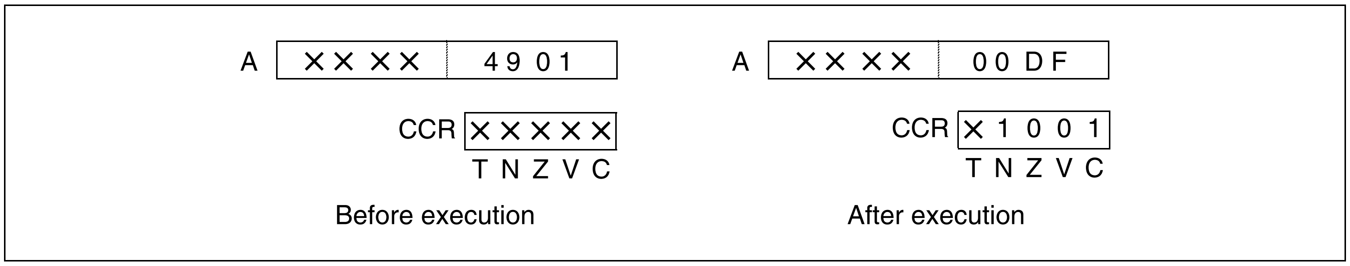
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	A	ear	eam
Second operand	#im8	dir	ear	eam	A	A
Number of bytes	2	2	2	2+	2	2+
Number of cycles	2	5	3	4+(a)	3	5+(a)
Correction value	0	(b)	0	(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

SUB A,#22H



9.1.117 SUBC (Subtract Byte Data of AL from AH with Carry to AL)

Subtract the low-order byte data of AL and the carry bit (C) from the low-order byte data of AH and restore the result in AL. "0" is transferred to bits 8 to 15 of the accumulator (A).

■ SUBC (Subtract Byte Data of AL from AH with Carry to AL)

● Assembler format:

SUBC A

● Operation:

(AL) ← (AH)–(AL)–(C) (Byte subtraction with a carry)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

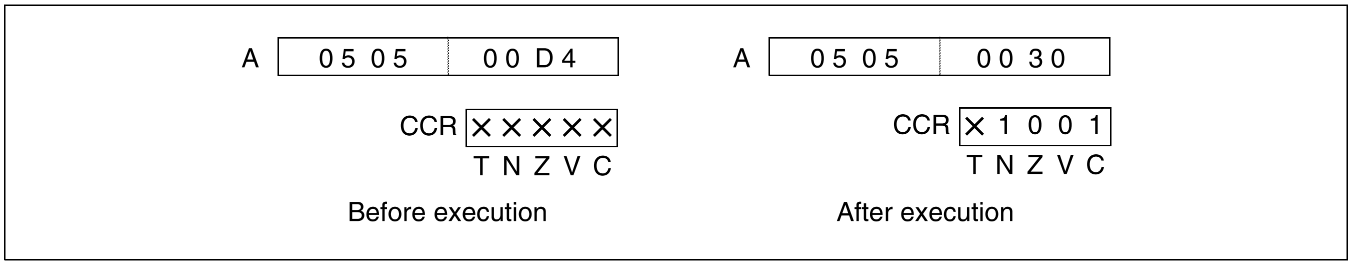
I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

SUBC A



9.1.118 SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator)

Subtract the byte data specified by the second operand and the carry bit (C) from the byte data of the accumulator (A) and restore the result in A. "0" is transferred to bits 8 to 15 of A.

■ SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator)

- Assembler format:

SUBC A,ear   SUBC A,eam

- Operation:

(A) ← (A)–(Second operand)–(C)   (Byte subtraction with a carry)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

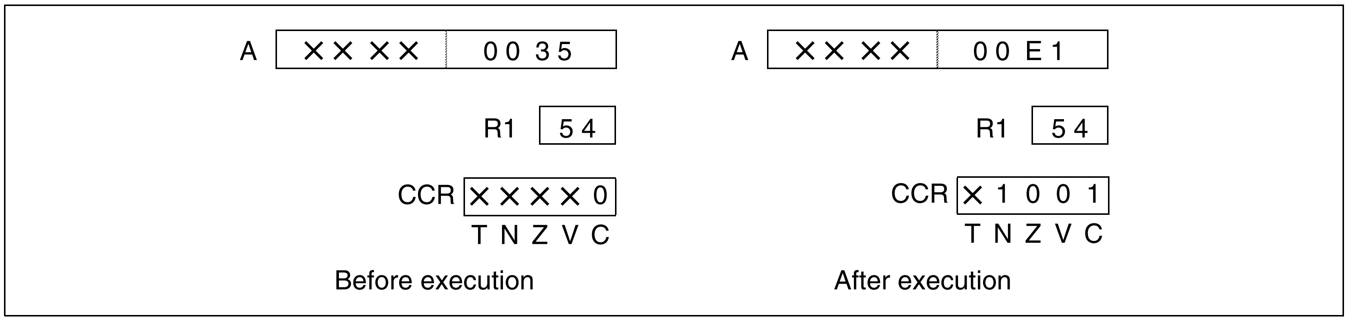
- Number of bytes, Number of cycles, and Correction value:

First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	4+(a)
Correction value	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

SUBC A,R1



9.1.119 SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator)

Subtract the word data specified by the second operand and the carry bit (C) from the low-order word data of the accumulator (A) and restore the result in A.

■ SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator)

● Assembler format:

SUBCW A,ear   SUBCW A,eam

● Operation:

(A) ← (A)–(Second operand)–(C)   (Word subtraction with a carry)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

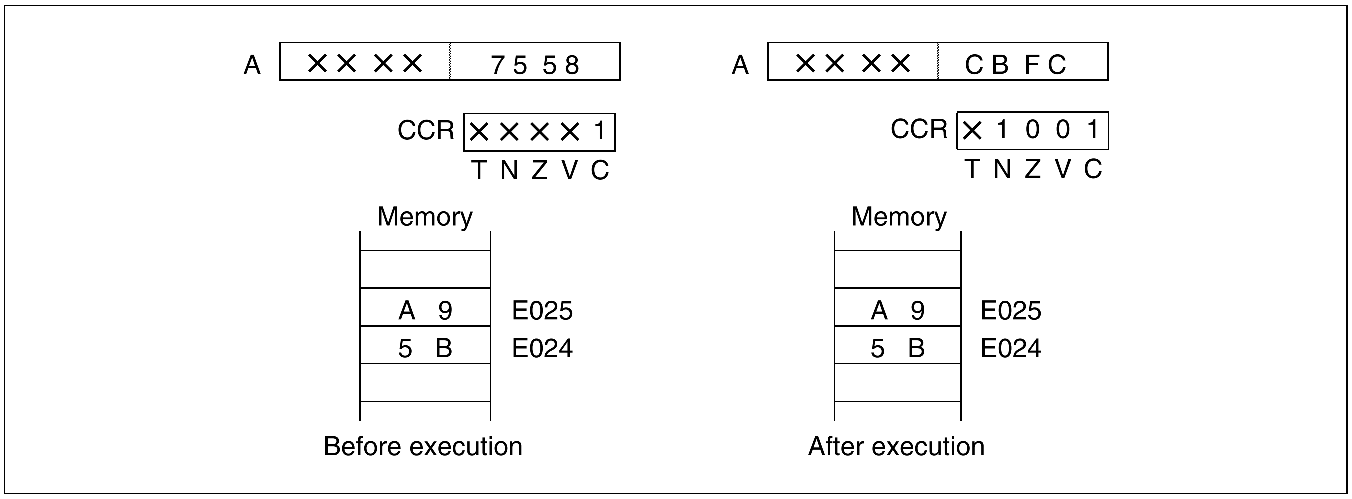
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	3	4+(a)
Correction value	0	(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

SUBCW A,0E024H



9.1.120 SUBDC (Subtract Decimal Data of AL from AH with Carry to AL)

Subtract the low-order byte data of AL and the carry bit (C) from the low-order byte data of AH and restore the result in AL. "0" is transferred to bits 8 to 15 of A.

■ SUBDC (Subtract Decimal Data of AL from AH with Carry to AL)

● Assembler format:

SUBDC A

● Operation:

(AL) ← (AH)–(AL)–(C) (Decimal subtraction with a carry)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

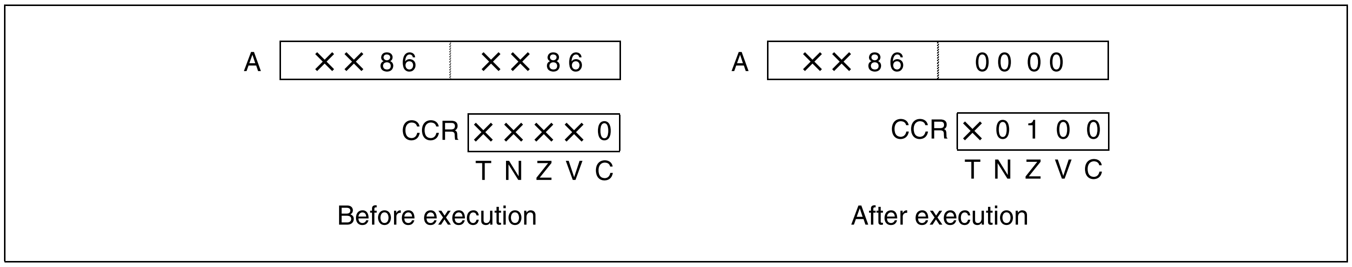
I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Undefined
C:	Set when a borrow has occurred as a result of the decimal operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3
Correction value:	0

● Example:

SUBDC A



9.1.121 SUBL (Subtract Long Word Data of Source from Destination to Destination)

Subtract the long word data specified by the second operand from the long word data of the accumulator (A) and restore the result in A.

■ SUBL (Subtract Long Word Data of Source from Destination to Destination)

- Assembler format:

SUBL A,#imm32

SUBL A,ear   SUBL A,eam

- Operation:

(First operand) ← (First operand)–(Second operand)   (Long word subtraction)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

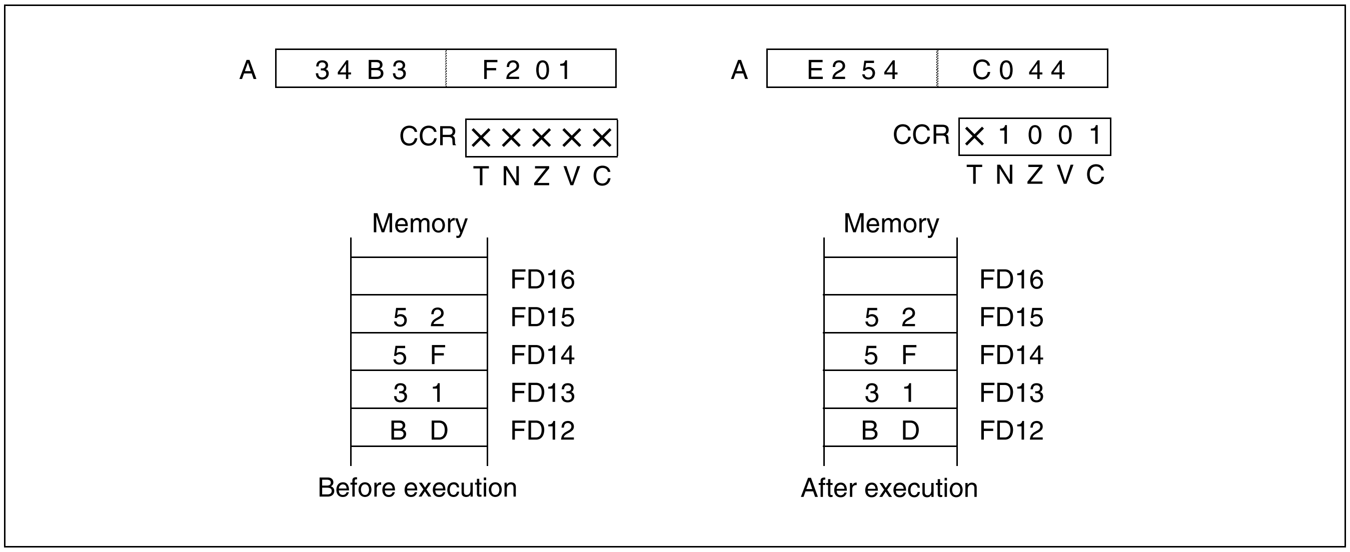
- Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A
Second operand	#i32	ear	eam
Number of bytes	5	2	2+
Number of cycles	4	6	7+(a)
Correction value	0	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

SUBL A,0FD12H



9.1.122 SUBW (Subtract Word Data of Source from Destination to Destination)

Subtract the word data specified by the second operand from the word data specified by the first operand and restore the result in the first operand.

■ SUBW (Subtract Word Data of Source from Destination to Destination)

● Assembler format:

SUBW A,#imm16	
SUBW A,ear	SUBW A,eam
SUBW ear,A	SUBW eam,A

● Operation:

(First operand) ← (First operand)–(Second operand) (Word subtraction)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

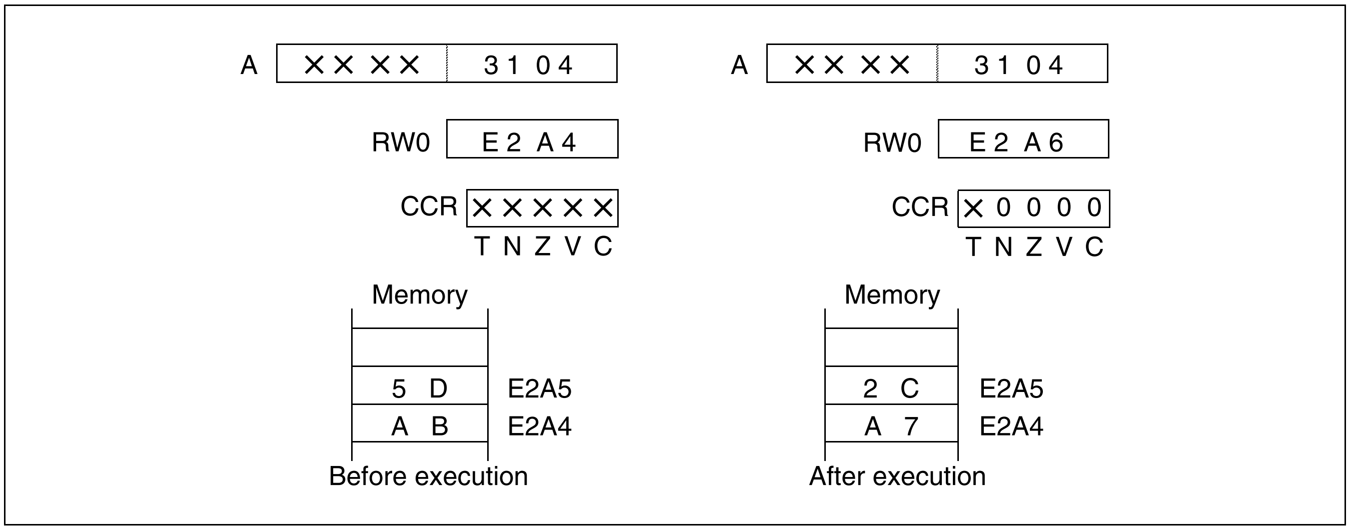
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	ear	eam
Second operand	#i16	ear	eam	A	A
Number of bytes	3	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

SUBW @RW0+,A



9.1.123 SUBW (Subtract Word Data of AL from AH to AL)

Subtract the word data of AL from the word data of AH and restore the result to AL.

■ SUBW (Subtract Word Data of AL from AH to AL)

● Assembler format:

SUBW A

● Operation:

(AL) ← (AH)–(AL) (Word subtraction)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	*	*

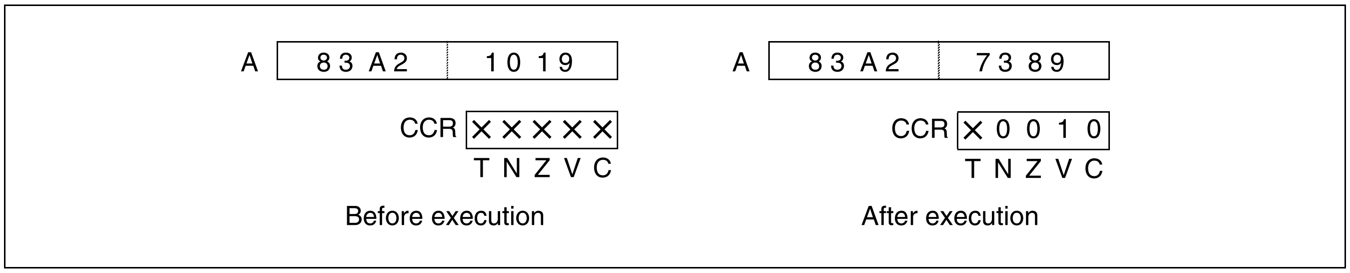
I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Set when an overflow has occurred as a result of the operation, cleared otherwise.
C:	Set when a borrow has occurred as a result of the operation, cleared otherwise.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

SUBW A



9.1.124 SWAP (Swap Byte Data of Accumulator)

Swap the high- and low-order bytes of the word data for the accumulator (A) with each other.

■ SWAP (Swap Byte Data of Accumulator)

● Assembler format:

SWAP

● Operation:

(A) 0 to 7 ↔ (A) 8 to 15 (Byte swapping)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

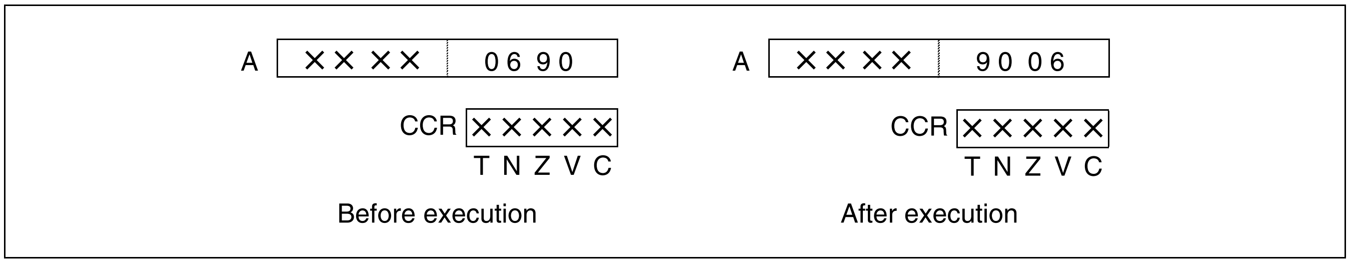
None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	3
Correction value:	0

● Example:

SWAP



9.1.125 SWAPW (Swap Word Data of Accumulator)

Swap the high- and low-order words of the long word data for the accumulator (A) with each other.

■ SWAPW (Swap Word Data of Accumulator)

- Assembler format:

  SWAPW

- Operation:

  Bits 0 to 15 of A ↔ Bits 16 to 31 of A (Word swapping)

- CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

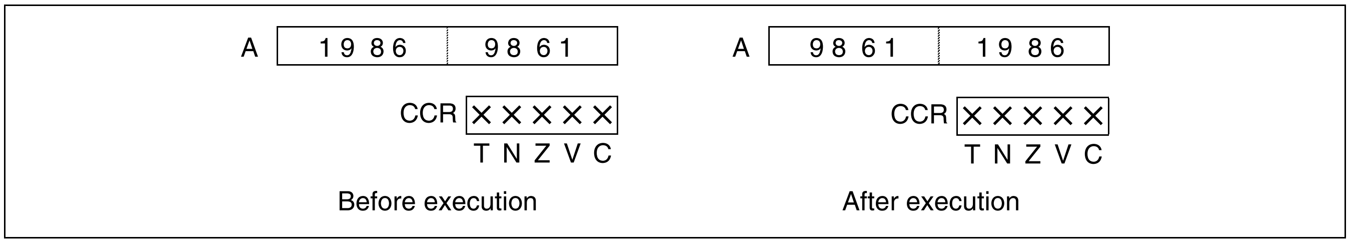
None of the flags is changed.

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

- Example:

  SWAPW



9.1.126 UNLINK (Unlink and Create New Stack Frame)

Restore an old frame pointer from a stack.

■ UNLINK (Unlink and Create New Stack Frame)

● Assembler format:

UNLINK

● Operation:

(sp) ← (RW3), (RW3) ← ((sp)), (sp) ← (sp)+2

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

Number of bytes: 1

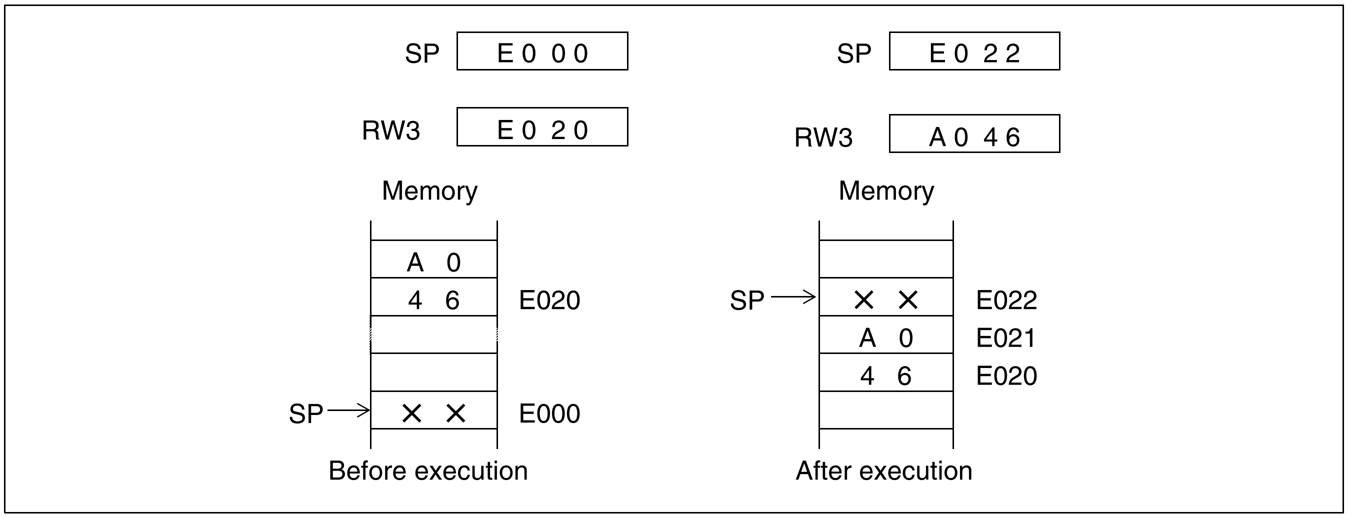
Number of cycles: 5

Correction value: (c)

For the explanation of (c), see Table 8.4-2 .

● Example:

UNLINK



9.1.127 WBTc (Wait until Bit Condition Satisfied)

This instruction keeps reading data from the bit address specified by the operand until that data satisfies the conditions. Once the data at the specified bit address satisfies the conditions, control is transferred to the instruction subsequent to the WBTc instruction.

■ WBTc (Wait until Bit Condition Satisfied)

● Assembler format:

WBTC io:bp

WBTS io:bp

● Operation:

Data is read from the bit address specified by io:bp until the data satisfies the condition. If the data from the bit address satisfies the condition, control is transferred to the next instruction.

Interrupts are acceptable while the read operation is repeated with the condition not satisfied. If an interrupt is generated in this state, the RETI instruction causes control to return to the WBTc instruction, not to the instruction following the WBTc instruction.

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

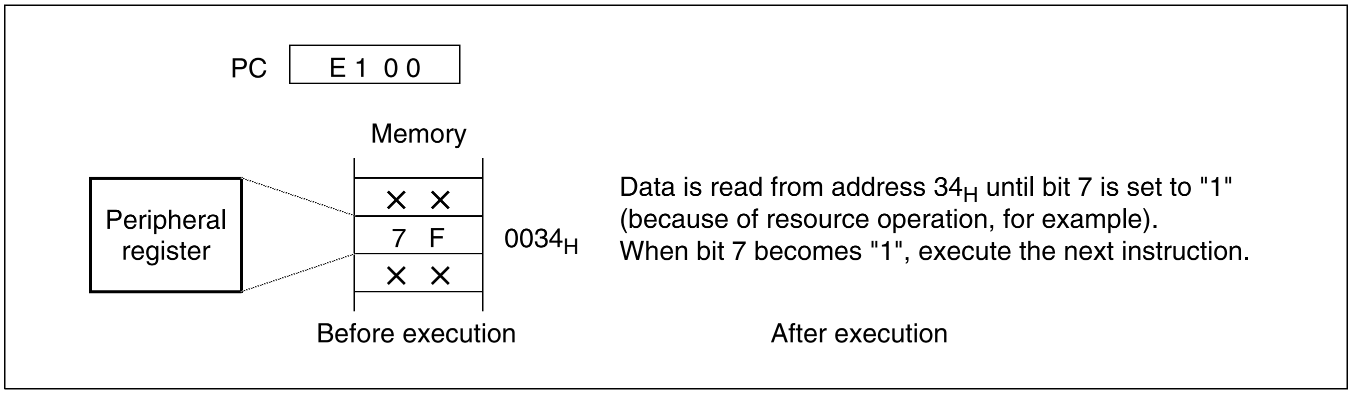
None of the flags is changed.

● Number of bytes, Number of cycles, and Correction value:

Instruction	WBTC	WBTS
Condition	Bit data=0	Bit data=1
Number of bytes	3	3
Number of cycles	Undefined	Undefined
Correction value	Until the condition is satisfied	Until the condition is satisfied

● Example:

WBTS 34H:7


PC
E 1 0 0
Memory
Peripheral register
× ×
7 F
× ×
0034_H
Data is read from address 34_H until bit 7 is set to "1" (because of resource operation, for example).
When bit 7 becomes "1", execute the next instruction.
Before execution
After execution

9.1.128 XCH (Exchange Byte Data of Source to Destination)

Exchange the byte data specified by the first operand with that specified by the second operand.
If the first operand is A, the high-order byte of AL is set to 00_H.

■ XCH (Exchange Byte Data of Source to Destination)

● Assembler format:

XCH A,ear XCH A,eam

XCH Ri,ear XCH Ri,eam

● Operation:

(First operand) ↔ (Second operand) (Byte exchange)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

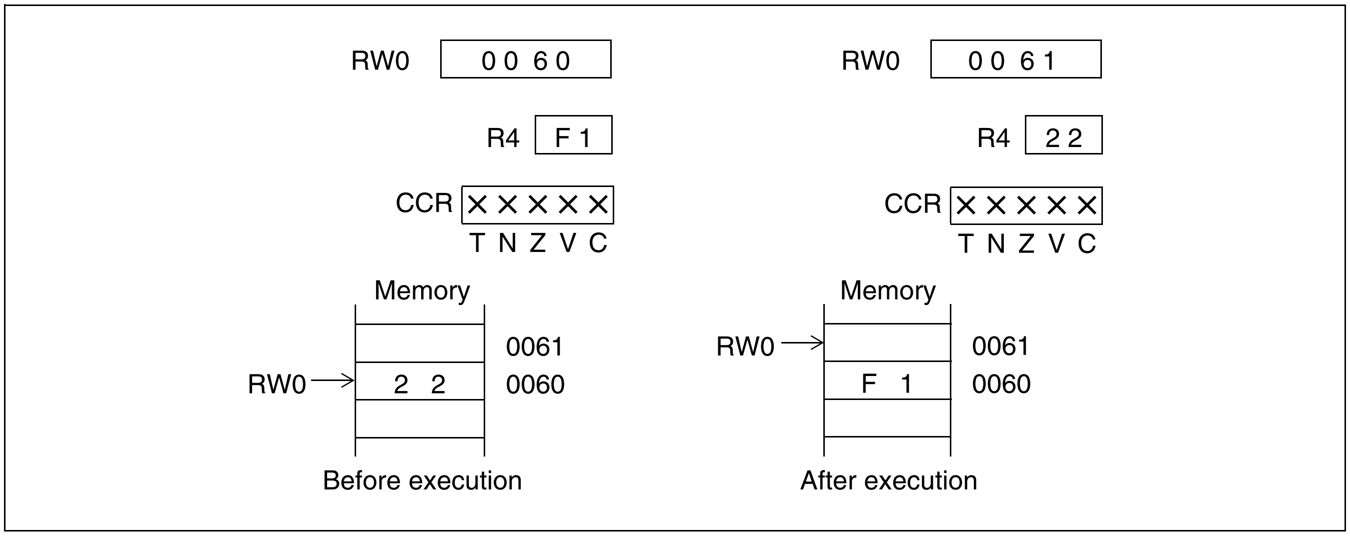
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	Ri	Ri
Second operand	ear	eam	ear	eam
Number of bytes	2	2+	2	2+
Number of cycles	4	5+(a)	7	9+(a)
Correction value	0	2×(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

XCH R4,@RW0+



9.1.129 XCHW (Exchange Word Data of Source to Destination)

Exchange the word data specified by the first operand with that specified by the second operand.

■ XCHW (Exchange Word Data of Source to Destination)

● Assembler format:

XCHW A,ear XCHW A,eam
XCHW RWi,ear XCHW RWi,eam

● Operation:

(First operand) ↔ (Second operand) (Word exchange)

● CCR:

I	S	T	N	Z	V	C
–	–	–	–	–	–	–

None of the flags is changed.

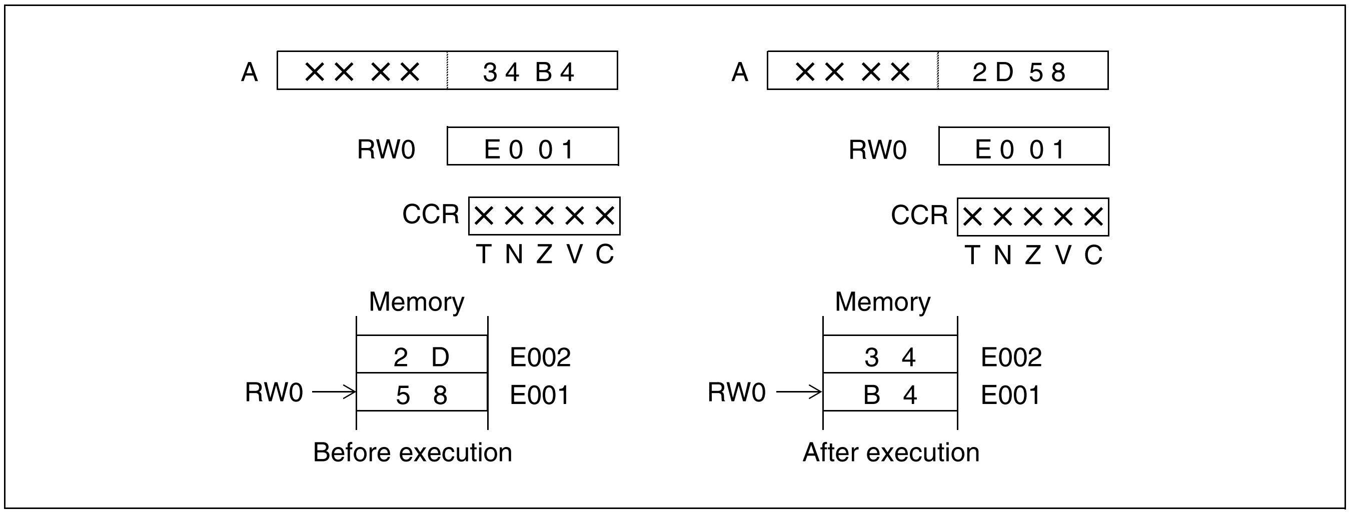
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	RWi	RWi
Second operand	ear	eam	ear	eam
Number of bytes	2	2+	2	2+
Number of cycles	4	5+(a)	7	9+(a)
Correction value	0	2×(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

XCHW A,@RW0



9.1.130 XOR (Exclusive Or Byte Data of Destination and Source to Destination)

Take the logical exclusive OR of the byte data specified by the first operand and the byte data specified by the second operand and restore the result in the first operand.

■ XOR (Exclusive Or Byte Data of Destination and Source to Destination)

- Assembler format:

XOR A,#imm8

XOR A,ear	XOR A,eam
XOR ear,A	XOR eam,A

- Operation:

(First operand) ← (First operand) xor (Second operand) (Byte logical exclusive OR)

- CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

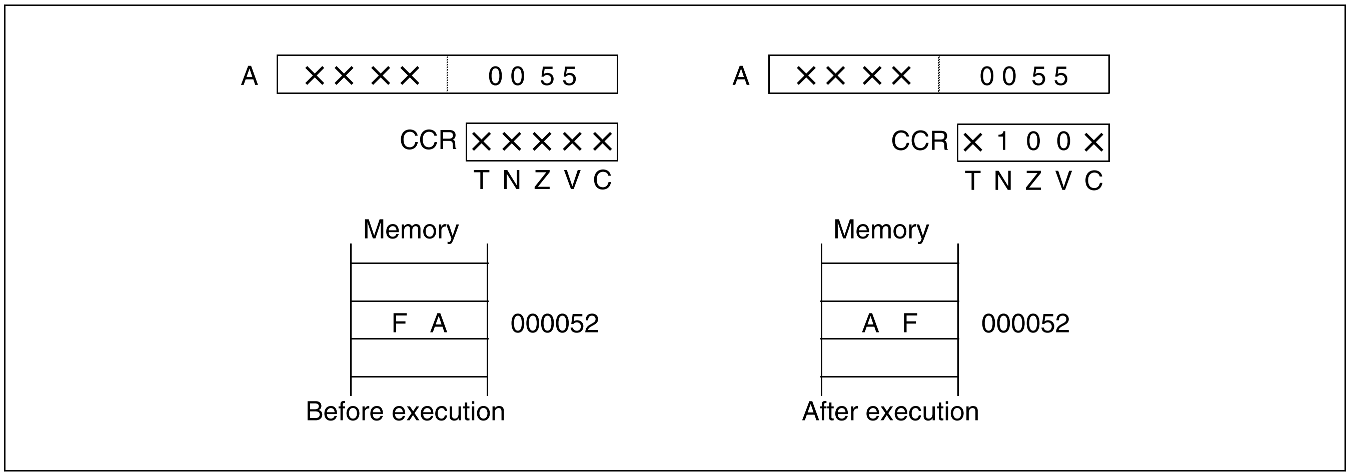
- Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	ear	eam
Second operand	#im8	ear	eam	A	A
Number of bytes	2	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(b)	0	2×(b)

For the explanation of (a) and (b) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

XOR 0052H,A



9.1.131 XORL (Exclusive Or Long Word Data of Destination and Source to Destination)

Take the logical exclusive OR of the long word data for the accumulator (A) and that specified by the second operand and restore the result in A.

■ XORL (Exclusive Or Long Word Data of Destination and Source to Destination)

● Assembler format:

XORL A,ear XORL A,eam

● Operation:

(A) ← (A) xor (Second operand) (Long word logical exclusive OR)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

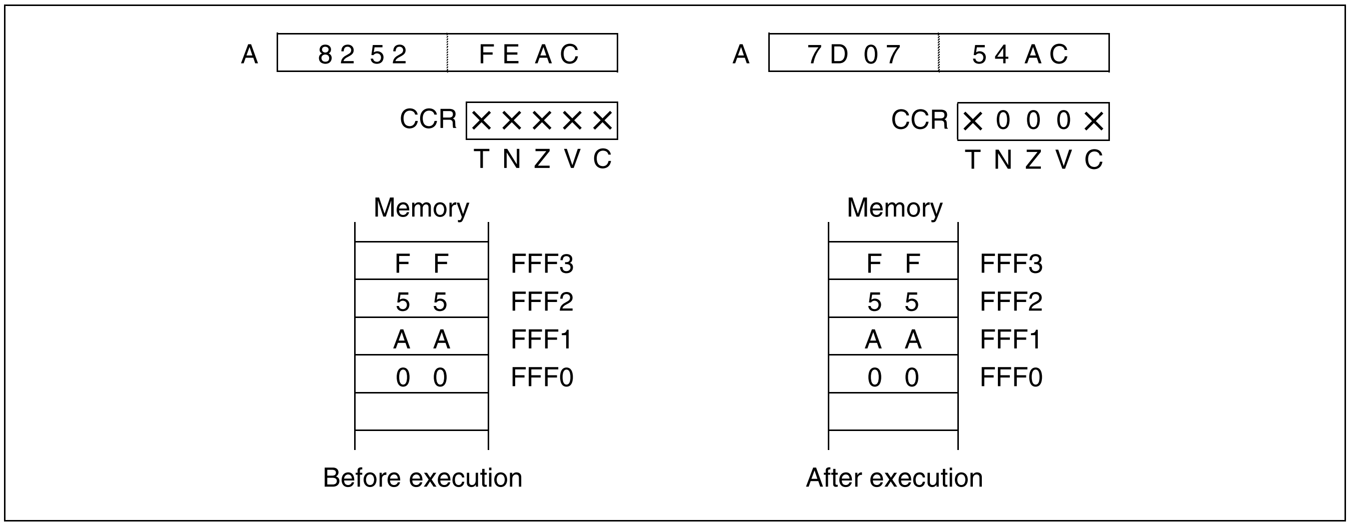
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A
Second operand	ear	eam
Number of bytes	2	2+
Number of cycles	6	7+(a)
Correction value	0	(d)

For the explanation of (a) and (d) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

XORL A,0FFF0H



9.1.132 XORW (Exclusive Or Word Data of AH and AL to AL)

Take the logical exclusive OR for the word data of AH and that of AL and restore the result in AL.

■ XORW (Exclusive Or Word Data of AH and AL to AL)

● Assembler format:

XORW A

● Operation:

(AL) ← (AH) xor (AL) (Word logical exclusive OR)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

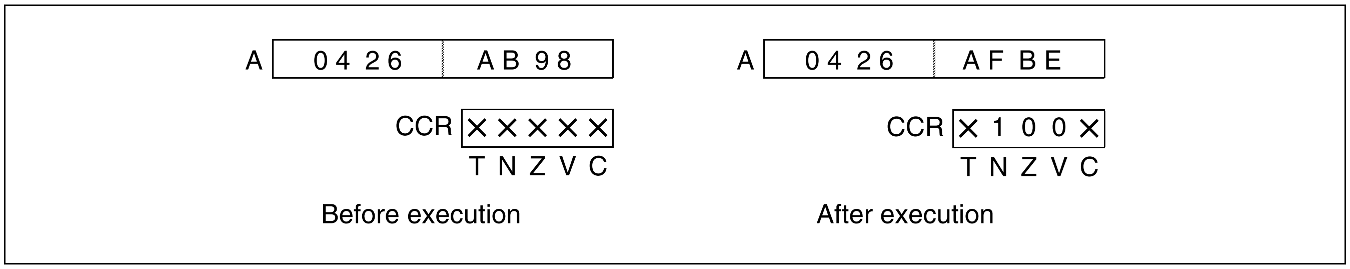
I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

● Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	2
Correction value:	0

● Example:

XORW A



9.1.133 XORW (Exclusive Or Word Data of Destination and Source to Destination)

Take the logical exclusive OR of the word data specified by the first operand and the word data specified by the second operand and restore the result in the first operand.

■ XORW (Exclusive Or Word Data of Destination and Source to Destination)

● Assembler format:

XORW A,#imm16	
XORW A,ear	XORW A,eam
XORW ear,A	XORW eam,A

● Operation:

(First operand) ← (First operand) xor (Second operand) (Word logical exclusive OR)

● CCR:

I	S	T	N	Z	V	C
–	–	–	*	*	R	–

I, S, and T:	Unchanged
N:	Set when the MSB of the operation result is "1", cleared otherwise.
Z:	Set when the operation result is "0", cleared otherwise.
V:	Cleared
C:	Unchanged

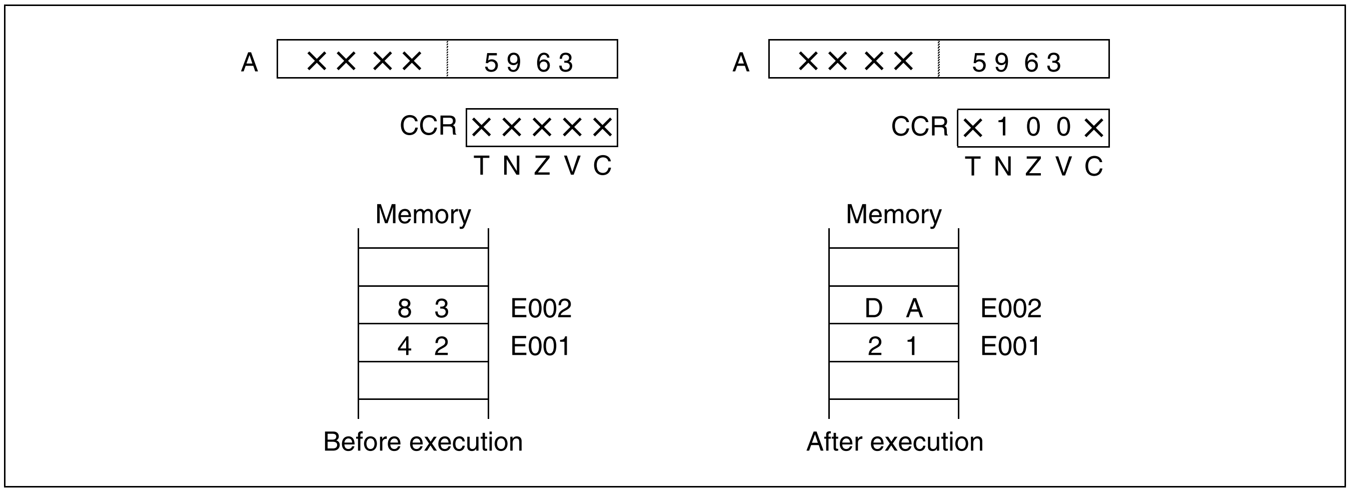
● Number of bytes, Number of cycles, and Correction value:

First operand	A	A	A	ear	eam
Second operand	#i16	ear	eam	A	A
Number of bytes	3	2	2+	2	2+
Number of cycles	2	3	4+(a)	3	5+(a)
Correction value	0	0	(c)	0	2×(c)

For the explanation of (a) and (c) in the table, see Table 8.4-1 and Table 8.4-2 .

● Example:

XORW 0E001H,A



9.1.134 ZEXT (Zero Extend from Byte Data to Word Data)

Transfer "0" to bits 8 to 15 of the accumulator (A).

■ ZEXT (Zero Extend from Byte Data to Word Data)

- Assembler format:

  ZEXT

- Operation:

  Bits 8 to 15 of A ← 00_H

- CCR:

I	S	T	N	Z	V	C
–	–	–	R	*	–	–

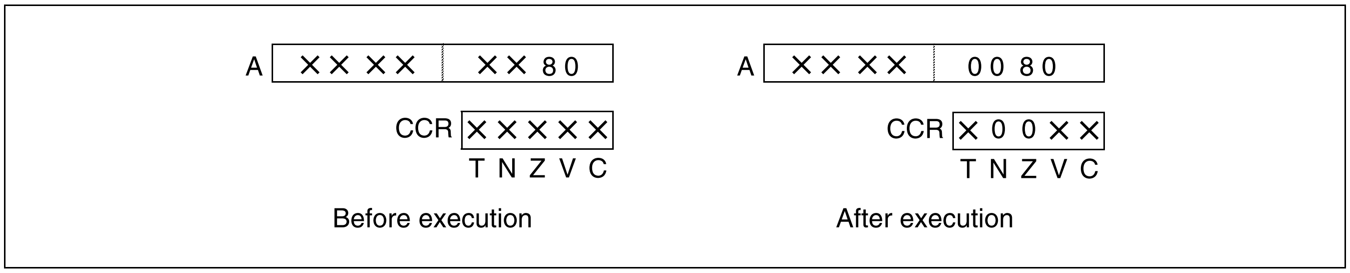
I, S, and T:	Unchanged
N:	Cleared
Z:	Set when the zero-extended data is "0", cleared otherwise.
V and C:	Unchanged

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	1
Correction value:	0

- Example:

  ZEXT



9.1.135 ZEXTW (Zero Extend from Word Data to Long Word Data)

Transfer "0" to bits 16 to 31 of the accumulator (A).

■ ZEXTW (Zero Extend from Word Data to Long Word Data)

- Assembler format:

ZEXTW

- Operation:

Bits 16 to 31 of A ← 0000_H

- CCR:

I	S	T	N	Z	V	C
–	–	–	R	*	–	–

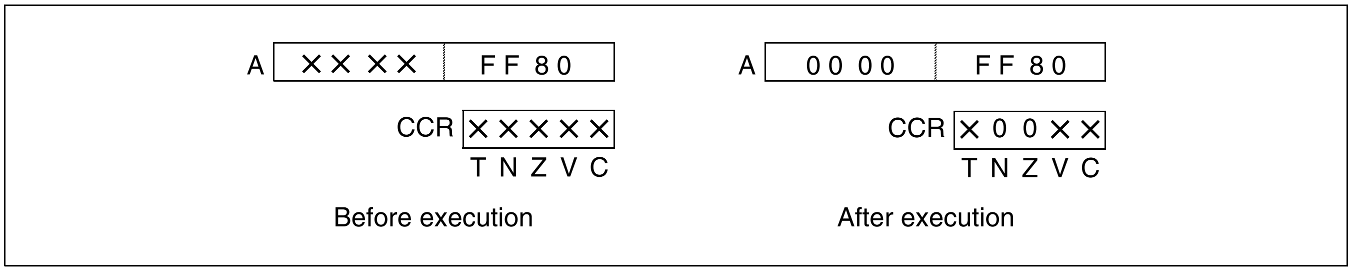
I, S, andT:	Unchanged
N:	Cleared
Z:	Set when the zero-extended data is "0", cleared otherwise.
V and C:	Unchanged

- Number of bytes, Number of cycles, and Correction value:

Number of bytes:	1
Number of cycles:	1
Correction value:	0

- Example:

ZEXTW



APPENDIX

This appendix includes lists and maps of instructions for the F^{2}MC-16LX.

APPENDIX A Explanation of Instruction Lists

APPENDIX B F^{2}MC-16LX Instruction Lists (351 Instructions)

APPENDIX C F^{2}MC-16LX Instruction Maps

APPENDIX A Explanation of Instruction Lists

This section explains items and symbols used in each instruction list included in Appendix B.

A.1 Items Used in Instruction Lists
A.2 Symbols Used in Instruction Lists
A.3 Effective Address Field
A.4 Calculating the Number of Execution Cycles

A.1 Items Used in Instruction Lists

Table A-1 explains the items used in the instruction lists.

■ Explanation of the Items Used in the Instruction Lists

Table A-1 Explanation of the Items Used in the Instruction Lists (1 / 2)

Item	Description
Mnemonic	Upper-case letters and symbols: Described as they appear in assembler. Lower-case letters: Replaced when described in assembler. Numbers after lower-case letters: Indicate the bit width within the instruction.
#	Indicates the number of bytes.
~	Indicates the number of cycles. See Table A-4 for details about meanings of letters in items.
RG	Indicates the register access count during execution of instruction. Used to calculate compensation values for CPU intermittent operation.
B	Indicates the compensation value for calculating the number of actual cycles during execution of instruction. The number of actual cycles during execution of instruction is the compensation value summed with the value in the "~" column.
Operation	Indicates operation of instruction.
LH	Indicates special operations involving bits 15 through 08 of the accumulator. Z: Transfers "0". X: Sign-extended transfer through sign extension. - : Transfers nothing.
AH	Indicates special operations involving the high-order 16 bits in the accumulator. *: Transfers from AL to AH. - : No transfer Z: Transfers 00_H to AH. X: Transfers 00_H or FF_H to AH using sign extension AL.
I	Indicates the status of each of the following flags: I (interrupt enable), S (stack), T (sticky bit), N (negative), Z (zero), V (overflow), and C (carry). *: Changes due to execution of instruction. - : No change S: Set by execution of instruction. R: Reset by execution of instruction.
S	
T	
N	
Z	
V	
C	

Table A-1 Explanation of the Items Used in the Instruction Lists (2 / 2)

Item	Description
RMW	Indicates whether the instruction is a read-modify-write instruction (a single instruction that reads data from memory, etc., processes the data, and then writes the result to memory.). *: Instruction is a read-modify-write instruction. -: Instruction is not a read-modify-write instruction. **Note:** A read-modify-write instruction cannot be used on addresses that have different meanings depending on whether they are read or written.

■ Number of Execution Cycles

The number of cycles required to execute instructions (number of execution cycles) is the summation of the number of cycles of each instruction, the compensation value determined by conditions, and the number of cycles required to fetch programs. If, however, a program stored in memory that is connected to a 16-bit bus such as internal ROMs is to be fetched, program fetch is executed every time instructions being executed exceed the two byte (one word) boundary. Therefore, interference with data access will increase the number of execution cycles. Since program fetch is executed for each byte of instructions being executed when a program stored on the memory connected to an external 8-bit bus is fetched, data access interference will increase the number of execution cycles.

When access is made to general-purpose registers, built-in ROMs, built-in RAMs, built-in I/O units, or external buses during CPU intermittent operation, the CPU clock suspends its operation for the number of cycles that is specified by the CG0/CG1 bit of the low-power-consumption-mode control register. Therefore, to obtain the number of cycles required to execute instructions during CPU intermittent operation, add these compensation values (the number of accesses multiplied by the number of cycled suspended) to the number of normal execution cycles.

A.2 Symbols Used in Instruction Lists

Table A-2 explains the symbols used in the instruction lists.

■ Explanation of the Symbols Used in the Instruction Lists

Table A-2 Explanation of the Symbols Used in the Instruction Lists (1 / 2)

Symbol	Explanation
A	32 bit accumulator The bit length used is different for each instruction. Byte: Lower 8 bits of AL Word: 16 bits of AL Long: 32 bits of AL and AH
AH	Upper 16 bits of A
AL	Lower 16 bits of A
SP	Stack pointer (USP or SSP)
PC	Program counter
PCB	Program bank register
DTB	Data bank register
ADB	Additional data bank register
SSB	System stack bank register
USB	User stack bank register
DPR	Direct page register
brg1	DTB, ADB, SSB, USB, DPR, PCB
brg2	DTB, ADB, SSB, USB, DPR
Ri	R0, R1, R2, R3, R4, R5, R6, R7
Rj	R0, R1, R2, R3
RWi	RW0, RW1, RW2, RW3, RW4, RW5, RW6, RW7
RWj	RW0, RW1, RW2, RW3
RLi	RL0, RL1, RL2, RL3
dir	Abbreviated direct addressing
addr16	Direct addressing
addr24	Physical direct addressing
ad24 0-15 ad24 16-23	Bit0 to bit15 of address 24 Bit16 to bit23 of address 24

Table A-2 Explanation of the Symbols Used in the Instruction Lists (2 / 2)

Symbol	Explanation
io	I/O area (000000_H to $0000FF_H$)
imm4	4-bit immediate data
imm8	8-bit immediate data
imm16	16-bit immediate data
imm32	32-bit immediate data
ext (imm8)	16-bit data signed and extended from 8-bit immediate data
disp8	8-bit displacement
disp16	16-bit displacement
bp	Bit offset value
vct4	Vector number (0 to 15)
vct8	Vector number (0 to 255)
( ) b	Bit address
rel	Branch specification relative to PC
ear	Effective addressing (codes 00 to 07)
eam	Effective addressing (codes 08 to 1F)
rlst	Register list

A.3 Effective Address Field

Table A-3 lists address formats used in the effective address field.

■ Effective Address Field

Table A-3 Effective Address Field

Code	Notation			Address format	Number of bytes of address expansion part*
00	R0	RW0	RL0	Register direct Starting from the left, "ea" corresponds to the byte, word and long-word types.	-
01	R1	RW1	(RL0)		
02	R2	RW2	RL1		
03	R3	RW3	(RL1)		
04	R4	RW4	RL2		
05	R5	RW5	(RL2)		
06	R6	RW6	RL3		
07	R7	RW7	(RL3)		
08	@RW0			Register indirect	0
09	@RW1				
0A	@RW2				
0B	@RW3				
0C	@RW0+			Register indirect with post-incrementing	0
0D	@RW1+				
0E	@RW2+				
0F	@RW3+				
10	@RW0+disp8			Register indirect with 8-bit displacement	1
11	@RW1+disp8				
12	@RW2+disp8				
13	@RW3+disp8				
14	@RW4+disp8				
15	@RW5+disp8				
16	@RW6+disp8				
17	@RW7+disp8				
18	@RW0+disp16			Register indirect with 16-bit displacement	2
19	@RW1+disp16				
1A	@RW2+disp16				
1B	@RW3+disp16				
1C	@RW0+RW7			Register indirect with index	0
1D	@RW1+RW7			Register indirect with index	0
1E	@PC+disp16			PC indirect with 16-bit displacement	2
1F	addr16			Direct address	2

* : The number of bytes of the address expansion part is shown in the "#" (number of bytes) column or the figure before "+" in the expression of bytes in the instruction details.

A.4 Calculating the Number of Execution Cycles

Table A-4 , Table A-5 , and Table A-6 show the method of calculating the number of execution cycles of instructions.

■ Calculating the Number of Execution Cycles

Table A-4 Number of Execution Cycles for Designating Each Effective Address

Code	Operand	(a)* Number of execution cycles for each form of addressing	Number of accesses for each form of addressing
00 to 07	Ri Rwi RLi	Listed in Table of Instructions	Listed in Table of Instructions
08 to 0B	@RWj	2	1
0C to 0F	@RWj+	4	2
10 to 17	@RWi+disp8	2	1
18 to 1B	@RWj+disp16	2	1
1C 1D 1E 1F	@RW0+RW7 @RW1+RW7 @PC+disp16 addr16	4 4 2 1	2 2 0 0

* : (a) is used in "~" (number of cycles), "B" (compensation value) (both in "APPENDIX B F^{2}MC-16LX Instruction Lists (351 Instructions)", and in "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS".

Table A-5 Compensation Values for Calculating the Number of Execution Cycles

Operand	(b) byte *		(c) word *		(d) long *	
	Cycles	Access count	Cycles	Access count	Cycles	Access count
Internal register	+0	1	+0	1	+0	2
Internal register even address	+0	1	+0	1	+0	2
Internal register odd address	+0	1	+2	2	+4	4
Even address on external data bus (16-bits)	+1	1	+1	1	+2	2
Odd address on external data bus (16-bits)	+1	1	+4	2	+8	4
External data bus (8-bit)	+1	1	+4	2	+8	4

* : (b), (c), and (d) are used in "~" (number of cycles), "B" (compensation value) (both in "APPENDIX B F^{2}MC-16LX Instruction Lists (351 Instructions)", and in "CHAPTER 9 DETAILED EXECUTION INSTRUCTIONS".

Note:

If external data buses are used, add the number of cycles that are weighted with ready input and automatic ready.

Table A-6 Compensation Values for Calculating The Number of Program Fetch Cycles

Instruction	Byte boundary	Word boundary
Internal memory	-	+2
External data bus (16-bit)	-	+3
External data bus (8-bit)	+3	-

Notes:

- If external data buses are used, add the number of cycles that are weighted with ready input and automatic ready.
- Since all cases of program fetch do not delay the execution of instructions, use this compensation value to calculate the worst case value.

APPENDIX B F^{2}MC-16LX Instruction Lists (351 Instructions)

This appendix lists the instructions used in assembler.
For items and symbols for each instruction list, see "APPENDIX A Explanation of Instruction Lists".

B.1 Transfer Instructions
B.2 Numeric Data Operation Instructions
B.3 Logical Data Operation Instruction
B.4 Shift Instruction
B.5 Branch Instructions
B.6 Other Instructions

B.1 Transfer Instructions

Table B-1 and Table B-2 lists the transfer instructions of the F²MC-16LX.

- **Transfer instruction (Byte): 41 instructions in Table B-1**
- **Transfer instruction (Word/Long-word): 38 instructions in Table B-2**

■ Transfer Instructions

Table B-1 Transfer Instruction (Byte): 41 Instructions

Mnemonic	#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
MOV A,dir	2	3	0	(b)	byte (A) ← (dir)	Z	*	-	-	-	*	*	-	-	-
MOV A,addr16	3	4	0	(b)	byte (A) ← (addr16)	Z	*	-	-	-	*	*	-	-	-
MOV A,Ri	1	2	1	0	byte (A) ← (Ri)	Z	*	-	-	-	*	*	-	-	-
MOV A,ear	2	2	1	0	byte (A) ← (ear)	Z	*	-	-	-	*	*	-	-	-
MOV A,eam	2+	3 + (a)	0	(b)	byte (A) ← (eam)	Z	*	-	-	-	*	*	-	-	-
MOV A,io	2	3	0	(b)	byte (A) ← (io)	Z	*	-	-	-	*	*	-	-	-
MOV A,#imm8	2	2	0	0	byte (A) ← (imm8)	Z	*	-	-	-	*	*	-	-	-
MOV A,@A	2	3	0	(b)	byte (A) ← ((A))	Z	-	-	-	-	*	*	-	-	-
MOV A,@RLi+disp8	3	10	2	(b)	byte (A) ← ((RLi)+disp8)	Z	*	-	-	-	*	*	-	-	-
MOVN A,#imm4	1	1	0	0	byte (A) ← imm4	Z	*	-	-	-	R	*	-	-	-
MOVX A,dir	2	3	0	(b)	byte (A) ← (dir)	X	*	-	-	-	*	*	-	-	-
MOVX A,addr16	3	4	0	(b)	byte (A) ← (addr16)	X	*	-	-	-	*	*	-	-	-
MOVX A,Ri	2	2	1	0	byte (A) ← (Ri)	X	*	-	-	-	*	*	-	-	-
MOVX A,ear	2	2	1	0	byte (A) ← (ear)	X	*	-	-	-	*	*	-	-	-
MOVX A,eam	2+	3 + (a)	0	(b)	byte (A) ← (eam)	X	*	-	-	-	*	*	-	-	-
MOVX A,io	2	3	0	(b)	byte (A) ← (io)	X	*	-	-	-	*	*	-	-	-
MOVX A,#imm8	2	2	0	0	byte (A) ← imm8	X	*	-	-	-	*	*	-	-	-
MOVX A,@A	2	3	0	(b)	byte (A) ← ((A))	X	-	-	-	-	*	*	-	-	-
MOVX A,@RWi+disp8	2	5	1	(b)	byte (A) ← ((RWi)+disp8)	X	*	-	-	-	*	*	-	-	-
MOVX A,@RLi+disp8	3	10	2	(b)	byte (A) ← ((RLi)+disp8)	X	*	-	-	-	*	*	-	-	-
MOV dir,A	2	3	0	(b)	byte (dir) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV addr16,A	3	4	0	(b)	byte (addr16) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV Ri,A	1	2	1	0	byte (Ri) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV ear,A	2	2	1	0	byte (ear) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV eam,A	2+	3 + (a)	0	(b)	byte (eam) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV io,A	2	3	0	(b)	byte (io) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV @RLi+disp8,A	3	10	2	(b)	byte ((RLi)+disp8) ← (A)	-	-	-	-	-	*	*	-	-	-
MOV Ri,ear	2	3	2	0	byte (Ri) ← (ear)	-	-	-	-	-	*	*	-	-	-
MOV Ri,eam	2+	4 + (a)	1	(b)	byte (Ri) ← (eam)	-	-	-	-	-	*	*	-	-	-
MOV ear,Ri	2	4	2	0	byte (ear) ← (Ri)	-	-	-	-	-	*	*	-	-	-
MOV eam,Ri	2+	5 + (a)	1	(b)	byte (eam) ← (Ri)	-	-	-	-	-	*	*	-	-	-
MOV Ri,#imm8	2	2	1	0	byte (Ri) ← imm8	-	-	-	-	-	*	*	-	-	-
MOV io,#imm8	3	5	0	(b)	byte (io) ← imm8	-	-	-	-	-	-	-	-	-	-
MOV dir,#imm8	3	5	0	(b)	byte (dir) ← imm8	-	-	-	-	-	-	-	-	-	-
MOV ear,#imm8	3	2	1	0	byte (ear) ← imm8	-	-	-	-	-	*	*	-	-	-
MOV eam,#imm8	3+	4 + (a)	0	(b)	byte (eam) ← imm8	-	-	-	-	-	-	-	-	-	-
MOV @AL,AH	2	3	0	(b)	byte ((A)) ← (AH)	-	-	-	-	-	*	*	-	-	-
XCH A,ear	2	4	2	0	byte (A) ↔ (ear)	Z	-	-	-	-	-	-	-	-	-
XCH A,eam	2+	5 + (a)	0	2 × (b)	byte (A) ↔ (eam)	Z	-	-	-	-	-	-	-	-	-
XCH Ri,ear	2	7	4	0	byte (Ri) ↔ (ear)	-	-	-	-	-	-	-	-	-	-
XCH Ri,eam	2+	9 + (a)	2	2 × (b)	byte (Ri) ↔ (eam)	-	-	-	-	-	-	-	-	-	-

Note: See Table A-4 and Table A-5 for information on (a) and (b) in the table.

Table B-2 Transfer Instruction (Word/Long-word): 38 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
MOVW	A,dir	2	3	0	(c)	word (A) ← (dir)	-	*	-	-	-	*	*	-	-	-
MOVW	A,addr16	3	4	0	(c)	word (A) ← (addr16)	-	*	-	-	-	*	*	-	-	-
MOVW	A,SP	1	1	0	0	word (A) ← (SP)	-	*	-	-	-	*	*	-	-	-
MOVW	A,RWi	1	2	1	0	word (A) ← (RWi)	-	*	-	-	-	*	*	-	-	-
MOVW	A,ear	2	2	1	0	word (A) ← (ear)	-	*	-	-	-	*	*	-	-	-
MOVW	A,eam	2+	3 + (a)	0	(c)	word (A) ← (eam)	-	*	-	-	-	*	*	-	-	-
MOVW	A,io	2	3	0	(c)	word (A) ← (io)	-	*	-	-	-	*	*	-	-	-
MOVW	A,@A	2	3	0	(c)	word (A) ← ((A))	-	-	-	-	-	*	*	-	-	-
MOVW	A,#imm16	3	2	2	0	word (A) ← imm16	-	*	-	-	-	*	*	-	-	-
MOVW	A,@RWi+disp8	2	5	1	(c)	word (A) ← ((RWi)+disp8)	-	*	-	-	-	*	*	-	-	-
MOVW	A,@RLi+disp8	3	10	2	(c)	word (A) ← ((RLi)+disp8)	-	*	-	-	-	*	*	-	-	-
MOVW	dir,A	2	3	0	(c)	word (dir) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	addr16,A	3	4	0	(c)	word (addr16) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	SP,A	1	1	0	0	word (SP) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	RWi,A	1	2	1	0	word (RWi) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	ear,A	2	2	1	0	word (ear) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	eam,A	2+	3 + (a)	0	(c)	word (eam) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	io,A	2	3	0	(c)	word (io) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	@RWi+disp8,A	2	5	1	(c)	word ((RWi)+disp8) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	@RLi+disp8,A	3	10	2	(c)	word ((RLi)+disp8) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVW	RWi,ear	2	3	2	0	word (RWi) ← (ear)	-	-	-	-	-	*	*	-	-	-
MOVW	Rwi,eam	2+	4 + (a)	1	(c)	word (RWi) ← (eam)	-	-	-	-	-	*	*	-	-	-
MOVW	ear,Rwi	2	4	2	0	word (ear) ← (RWi)	-	-	-	-	-	*	*	-	-	-
MOVW	eam,Rwi	2+	5 + (a)	1	(c)	word (eam) ← (RWi)	-	-	-	-	-	*	*	-	-	-
MOVW	RWi,#imm16	3	2	1	0	word (RWi) ← imm16	-	-	-	-	-	*	*	-	-	-
MOVW	io,#imm16	4	5	0	(c)	word (io) ← imm16	-	-	-	-	-	-	-	-	-	-
MOVW	ear,#imm16	4	2	1	0	word (ear) ← imm16	-	-	-	-	-	*	*	-	-	-
MOVW	eam,#imm16	4+	4 + (a)	0	(c)	word (eam) ← imm16	-	-	-	-	-	-	-	-	-	-
MOVW	@AL,AH	2	3	0	(c)	word ((A)) ← (AH)	-	-	-	-	-	*	*	-	-	-
XCHW	A,ear	2	4	2	0	word (A) ↔ (ear)	-	-	-	-	-	-	-	-	-	-
XCHW	A,eam	2+	5 + (a)	0	2 × (c)	word (A) ↔ (eam)	-	-	-	-	-	-	-	-	-	-
XCHW	RWi, ear	2	7	4	0	word (RWi) ↔ (ear)	-	-	-	-	-	-	-	-	-	-
XCHW	RWi, eam	2+	9 + (a)	2	2 × (c)	word (RWi) ↔ (eam)	-	-	-	-	-	-	-	-	-	-
MOVL	A,ear	2	4	2	0	long (A) ← (ear)	-	-	-	-	-	*	*	-	-	-
MOVL	A,eam	2+	5 + (a)	0	(d)	long (A) ← (eam)	-	-	-	-	-	*	*	-	-	-
MOVL	A,#imm32	5	3	0	0	long (A) ← imm32	-	-	-	-	-	*	*	-	-	-
MOVL	ear,A	2	4	2	0	long (ear1) ← (A)	-	-	-	-	-	*	*	-	-	-
MOVL	eam,A	2+	5 + (a)	0	(d)	long(eam1) ← (A)	-	-	-	-	-	*	*	-	-	-

Note: See Table A-4 and Table A-5 for information on (a), (c) and (d) in the table.

B.2 Numeric Data Operation Instructions

The numeric data operation instructions of the F^{2}MC-16L are listed in the following five tables:

- **Table B-3 for addition and subtraction (Byte/Word/Long-word) : 42 instructions**
- **Table B-4 for increment and decrement (Byte/Word/Long-word) : 12 instructions**
- **Table B-5 for compare (Byte/Word/Long-word) : 11 instructions**
- **Table B-6 for unsigned multiplication and division : 11 instructions (Word/Long-word)**
- **Table B-7 for signed multiplication and division : 11 instructions (Word/Long-word)**

■ Numeric Data Operation Instructions

Table B-3 Addition and Subtraction (Byte/Word/Long-word): 42 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
ADD	A,#imm8	2	2	0	0	byte (A) ← (A) + imm8	Z	-	-	-	-	*	*	*	*	-
ADD	A,dir	2	5	0	(b)	byte (A) ← (A) + (dir)	Z	-	-	-	-	*	*	*	*	-
ADD	A,ear	2	3	1	0	byte (A) ← (A) + (ear)	Z	-	-	-	-	*	*	*	*	-
ADD	A,eam	2+	4 + (a)	0	(b)	byte (A) ← (A) + (eam)	Z	-	-	-	-	*	*	*	*	-
ADD	ear,A	2	3	2	0	byte (ear) ← (ear) + (A)	-	-	-	-	-	*	*	*	*	-
ADD	eam,A	2+	5 + (a)	0	2 × (b)	byte (eam) ← (eam) + (A)	Z	-	-	-	-	*	*	*	*	*
ADDC	A	1	2	0	0	byte (A) ← (AH) + (AL) + (C)	Z	-	-	-	-	*	*	*	*	-
ADDC	A,ear	2	3	1	0	byte (A) ← (A) + (ear)+ (C)	Z	-	-	-	-	*	*	*	*	-
ADDC	A,eam	2+	4 + (a)	0	(b)	byte (A) ← (A) + (eam)+ (C)	Z	-	-	-	-	*	*	*	*	-
ADDDC	A	1	3	0	0	byte (A) ← (AH) + (AL) + (C) (decimal)	Z	-	-	-	-	*	*	*	*	-
SUB	A,#imm8	2	2	0	0	byte (A) ← (A) - imm8	Z	-	-	-	-	*	*	*	*	-
SUB	A,dir	2	5	0	(b)	byte (A) ← (A) - (dir)	Z	-	-	-	-	*	*	*	*	-
SUB	A,ear	2	3	1	0	byte (A) ← (A) - (ear)	Z	-	-	-	-	*	*	*	*	-
SUB	A,eam	2+	4 + (a)	0	(b)	byte (A) ← (A) - (eam)	Z	-	-	-	-	*	*	*	*	-
SUB	ear,A	2	3	2	0	byte (ear) ← (ear) - (A)	-	-	-	-	-	*	*	*	*	-
SUB	eam,A	2+	5 + (a)	0	2 × (b)	byte (eam) ← (eam) - (A)	-	-	-	-	-	*	*	*	*	*
SUBC	A	1	2	0	0	byte (A) ← (AH) - (AL) - (C)	Z	-	-	-	-	*	*	*	*	-
SUBC	A,ear	2	3	1	0	byte (A) ← (A) - (ear) - (C)	Z	-	-	-	-	*	*	*	*	-
SUBC	A,eam	2+	4 + (a)	0	(b)	byte (A) ← (A) - (eam) - (C)	Z	-	-	-	-	*	*	*	*	-
SUBDC	A	1	3	0	0	byte (A) ← (AH) - (AL) - (C) (decimal)	Z	-	-	-	-	*	*	*	*	-
ADDW	A	1	2	0	0	word (A) ← (AH) + (AL)	-	-	-	-	-	*	*	*	*	-
ADDW	A,ear	2	3	1	0	word (A) ← (A) + (ear)	-	-	-	-	-	*	*	*	*	-
ADDW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) + (eam)	-	-	-	-	-	*	*	*	*	-
ADDW	A,#imm16	3	2	0	0	word (A) ← (A) + imm16	-	-	-	-	-	*	*	*	*	-
ADDW	ear,A	2	3	2	0	word (ear) ← (ear) + (A)	-	-	-	-	-	*	*	*	*	-
ADDW	eam,A	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) + (A)	-	-	-	-	-	*	*	*	*	*
ADDCW	A,ear	2	3	1	0	word (A) ← (A) + (ear) + (C)	-	-	-	-	-	*	*	*	*	-
ADDCW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) + (eam) + (C)	-	-	-	-	-	*	*	*	*	-
SUBW	A	1	2	0	0	word (A) ← (AH) - (AL)	-	-	-	-	-	*	*	*	*	-
SUBW	A,ear	2	3	1	0	word (A) ← (A) - (ear)	-	-	-	-	-	*	*	*	*	-
SUBW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) - (eam)	-	-	-	-	-	*	*	*	*	-
SUBW	A,#imm16	3	2	0	0	word (A) ← (A) - imm16	-	-	-	-	-	*	*	*	*	-
SUBW	ear,A	2	3	2	0	word (ear) ← (ear) - (A)	-	-	-	-	-	*	*	*	*	-
SUBW	eam,A	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) - (A)	-	-	-	-	-	*	*	*	*	*
SUBCW	A,ear	2	3	1	0	word (A) ← (A) - (ear) - (C)	-	-	-	-	-	*	*	*	*	-
SUBCW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) - (eam) - (C)	-	-	-	-	-	*	*	*	*	-
ADDL	A,ear	2	6	2	0	long (A) ← (A) + (ear)	-	-	-	-	-	*	*	*	*	-
ADDL	A,eam	2+	7+(a)	0	(d)	long (A) ← (A) + (eam)	-	-	-	-	-	*	*	*	*	-
ADDL	A,#imm32	5	4	0	0	long (A) ← (A) + imm32	-	-	-	-	-	*	*	*	*	-
SUBL	A,ear	2	6	2	0	long (A) ← (A) - (ear)	-	-	-	-	-	*	*	*	*	-
SUBL	A,eam	2+	7+(a)	0	(d)	long (A) ← (A) - (eam)	-	-	-	-	-	*	*	*	*	-
SUBL	A,#imm32	5	4	0	0	long (A) ← (A) - imm32	-	-	-	-	-	*	*	*	*	-

Note: See Table A-4 and Table A-5 for information on (a) to (d) in the table.

Table B-4 Increment and Decrement (Byte/Word/Long-word): 12 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
INC	ear	2	3	2	0	byte (ear) ← (ear) + 1	-	-	-	-	-	*	*	*	-	-
INC	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← (eam) + 1	-	-	-	-	-	*	*	*	-	*
DEC	ear	2	3	2	0	byte (ear) ← (ear) - 1	-	-	-	-	-	*	*	*	-	-
DEC	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← (eam) - 1	-	-	-	-	-	*	*	*	-	*
INCW	ear	2	3	2	0	word (ear) ← (ear) + 1	-	-	-	-	-	*	*	*	-	-
INCW	eam	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) + 1	-	-	-	-	-	*	*	*	-	*
DECW	ear	2	3	2	0	word (ear) ← (ear) - 1	-	-	-	-	-	*	*	*	-	-
DECW	eam	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) - 1	-	-	-	-	-	*	*	*	-	*
INCL	ear	2	7	4	0	long (ear) ← (ear) + 1	-	-	-	-	-	*	*	*	-	-
INCL	eam	2+	9+(a)	0	2 × (d)	long (eam) ← (eam) + 1	-	-	-	-	-	*	*	*	-	*
DECL	ear	2	7	4	0	long (ear) ← (ear) - 1	-	-	-	-	-	*	*	*	-	-
DECL	eam	2+	9+(a)	0	2 × (d)	long (eam) ← (eam) - 1	-	-	-	-	-	*	*	*	-	*

Note: See Table A-4 and Table A-5 for information on (a) to (d) in the table.

Table B-5 Compare (Byte/Word/Long-word): 11 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
CMP	A	1	1	0	0	byte (AH) - (AL)	-	-	-	-	-	*	*	*	*	-
CMP	A,ear	2	2	1	0	byte (A) - (ear)	-	-	-	-	-	*	*	*	*	-
CMP	A,eam	2+	3+(a)	0	(b)	byte (A) - (eam)	-	-	-	-	-	*	*	*	*	-
CMP	A,#imm8	2	2	0	0	byte (A) - imm8	-	-	-	-	-	*	*	*	*	-
CMPW	A	1	1	0	0	word (AH) - (AL)	-	-	-	-	-	*	*	*	*	-
CMPW	A,ear	2	2	1	0	word (A) - (ear)	-	-	-	-	-	*	*	*	*	-
CMPW	A,eam	2+	3+(a)	0	(c)	word (A) - (eam)	-	-	-	-	-	*	*	*	*	-
CMPW	A,#imm16	3	2	0	0	word (A) - imm16	-	-	-	-	-	*	*	*	*	-
CMPL	A,ear	2	6	2	0	long (A) - (ear)	-	-	-	-	-	*	*	*	*	-
CMPL	A,eam	2+	7+(a)	0	(d)	long (A) - (eam)	-	-	-	-	-	*	*	*	*	-
CMPL	A,#imm32	5	3	0	0	long (A) - imm32	-	-	-	-	-	*	*	*	*	-

Note: See Table A-4 and Table A-5 for information on (a) to (d) in the table.

Table B-6 Unsigned Multiplication and Division: 11 Instructions (Word/Long-word)

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
DIVU	A	1	*1	0	0	word (AH) / byte (AL) quotient → byte (AL) Remainder → byte (AH)	-	-	-	-	-	-	-	*	*	-
DIVU	A,ear	2	*2	1	0	word (A) / byte (ear) quotient → byte (A) Remainder → byte (ear)	-	-	-	-	-	-	-	*	*	-
DIVU	A,eam	2+	*3	0	*6	word (A) / byte (eam) quotient → byte (A) Remainder → byte (eam)	-	-	-	-	-	-	-	*	*	-
DIVUW	A,ear	2	*4	1	0	long (A) / word (ear) quotient → word (A) Remainder → word (ear)	-	-	-	-	-	-	-	*	*	-
DIVUW	A,eam	2+	*5	0	*7	long (A) / word (eam) quotient → word (A) Remainder → word (eam)	-	-	-	-	-	-	-	*	*	-
MULU	A	1	*8	0	0	byte (AH) * byte (AL) → word (A)	-	-	-	-	-	-	-	-	-	-
MULU	A,ear	2	*9	1	0	byte (A) * byte (ear) → word (A)	-	-	-	-	-	-	-	-	-	-
MULU	A,eam	2+	*10	0	(b)	byte (A) * byte (eam) → word (A)	-	-	-	-	-	-	-	-	-	-
MULUW	A	1	*11	0	0	word (AH) * word (AL) → Long (A)	-	-	-	-	-	-	-	-	-	-
MULUW	A,ear	2	*12	1	0	word (A) * word (ear) → Long (A)	-	-	-	-	-	-	-	-	-	-
MULUW	A,eam	2+	*13	0	(c)	word (A) * word (eam) → Long (A)	-	-	-	-	-	-	-	-	-	-

*1: 3 for zero-divide, 7 for overflow, 15 for normal
*2: 4 for zero-divide, 8 for overflow, 16 for normal
*3: 6 + (a) for zero-divide, 9 + (a) for overflow, 19 + (a) for normal
*4: 4 for zero-divide, 7 for overflow, 22 for normal
*5: 6 + (a) for zero-divide, 8 + (a) for overflow, 26 + (a) for normal
*6: (b) for zero-divide or overflow, 2 × (b) for normal
*7: (c) for zero-divide or overflow, 2 × (c) for normal
*8: 3 when byte (AH) is 0, 7 when byte (AH) is not 0
*9: 4 when byte (ear) is 0, 8 when byte (ear) is not 0
*10: 5 + (a) when byte (eam) is 0, 9 + (a) when byte (eam) is not 0
*11: 3 when word (AH) is 0, 11 when word (AH) is not 0
*12: 4 when word (ear) is 0, 12 when word (ear) is not 0
*13: 5 + (a) when word (eam) is 0, 13 + (a) when word (eam) is not 0
Note: See Table A-4 and Table A-5 for information on (a), (b) and (c) in the table.

Table B-7 Signed Multiplication and Division: 11 Instructions (Word/Long-word)

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
DIV	A	2	*1	0	0	word (AH) / byte (AL) quotient → byte (AL) Remainder → byte (AH)	Z	-	-	-	-	-	-	*	*	-
DIV	A,ear	2	*2	1	0	word (A) / byte (ear) quotient → byte (A) Remainder → byte (ear)	Z	-	-	-	-	-	-	*	*	-
DIV	A,eam	2+	*3	0	*6	word (A) / byte (eam) quotient → byte (A) Remainder → byte (eam)	Z	-	-	-	-	-	-	*	*	-
DIVW	A,ear	2	*4	1	0	long (A) / word (ear) quotient → word (A) Remainder → word (ear)	-	-	-	-	-	-	-	*	*	-
DIVW	A,eam	2+	*5	0	*7	long (A) / word (eam) quotient → word (A) Remainder → word (eam)	-	-	-	-	-	-	-	*	*	-
MUL	A	2	*8	0	0	byte (AH) * byte (AL) → word (A)	-	-	-	-	-	-	-	-	-	-
MUL	A,ear	2	*9	1	0	byte (A) * byte (ear) → word (A)	-	-	-	-	-	-	-	-	-	-
MUL	A,eam	2+	*10	0	(b)	byte (A) * byte (eam) → word (A)	-	-	-	-	-	-	-	-	-	-
MULW	A	2	*11	0	0	word (AH) * word (AL) → Long (A)	-	-	-	-	-	-	-	-	-	-
MULW	A,ear	2	*12	1	0	word (A) * word (ear) → Long (A)	-	-	-	-	-	-	-	-	-	-
MULW	A,eam	2+	*13	0	(c)	word (A) * word (eam) → Long (A)	-	-	-	-	-	-	-	-	-	-

*1: 3 for zero-divide, 8 or 18 for overflow, 18 for normal
*2: 4 for zero-divide, 11 or 22 for overflow, 23 for normal
*3: 5 + (a) for zero-divide, 12 + (a) or 23 + (a) for overflow, 24 + (a) for normal
*4: If the dividend is positive: 4 for zero-divide, 12 or 30 for overflow, 31 for normal
If the dividend is negative: 4 for zero-divide, 12 or 31 for overflow, 32 for normal
*5: If the dividend is positive: 5 + (a) for zero-divide, 12 + (a) or 31 + (a) for overflow, 32 + (a) for normal
If the dividend is negative: 5 + (a) for zero-divide, 12 + (a) or 32 + (a) for overflow, 33 + (a) for normal
*6: (b) for zero-divide or overflow, 2 x (b) for normal
*7: (c) for zero-divide or overflow, 2 x (c) for normal
*8: 3 when byte (AH) is 0, 12 when the result is positive, 13 when the result is negative
*9: 4 when byte (ear) is 0, 13 when the result is positive, 14 when the result is negative
*10: 5 + (a) when byte (eam) is 0, 14 + (a) when the result is positive, 15 + (a) when the result is negative
*11: 3 when word (AH) is 0, 16 when the result is positive, 19 when the result is negative
*12: 4 when word (ear) is 0, 17 when the result is positive, 20 when the result is negative
*13: 5 + (a) when word (eam) is 0, 18 + (a) when the result is positive, 21 + (a) when the result is negative

Notes:

- There are two numbers of execution cycles when overflow occurs during execution of the DIV and DIVW instructions, depending on the number of execution cycles is detected before or after operation.
- The content of AL is corrupted when overflow occurs during execution of the DIV and DIVW instructions.
- See Table A-4 and Table A-5 for information on (a), (b) and (c) in the table.

B.3 Logical Data Operation Instruction

The logical data operation instructions of the F2MC-16L are listed in the following four tables:

- **Table B-8 for logic 1 (Byte/Word): 39 instructions**
- **Table B-9 for logic 2 (Long): 6 instructions**
- **Table B-10 for sign inversion (Byte/Word): 6 instructions**
- **Table B-11 for normalize instructions (Long): 1 instruction**

■ Logical Operation Instruction

Table B-8 Logic 1 (Byte/Word): 39 Instructions (1 / 2)

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
AND	A,#imm8	2	2	0	0	byte (A) ← (A) and imm8	-	-	-	-	-	*	*	R	-	-
AND	A,ear	2	3	1	0	byte (A) ← (A) and (ear)	-	-	-	-	-	*	*	R	-	-
AND	A,eam	2+	4+(a)	0	(b)	byte (A) ← (A) and (eam)	-	-	-	-	-	*	*	R	-	-
AND	ear,A	2	3	2	0	byte (ear) ← (ear) and (A)	-	-	-	-	-	*	*	R	-	-
AND	eam,A	2+	5+(a)	0	2 × (b)	byte (eam) ← (eam) and (A)	-	-	-	-	-	*	*	R	-	*
OR	A,#imm8	2	2	0	0	byte (A) ← (A) or imm8	-	-	-	-	-	*	*	R	-	-
OR	A,ear	2	3	1	0	byte (A) ← (A) or (ear)	-	-	-	-	-	*	*	R	-	-
OR	A,eam	2+	4+(a)	0	(b)	byte (A) ← (A) or (eam)	-	-	-	-	-	*	*	R	-	-
OR	ear,A	2	3	2	0	byte (ear) ← (ear) or (A)	-	-	-	-	-	*	*	R	-	-
OR	eam,A	2+	5+(a)	0	2 × (b)	byte (eam) ← (eam) or (A)	-	-	-	-	-	*	*	R	-	*
XOR	A,#imm8	2	2	0	0	byte (A) ← (A) xor imm8	-	-	-	-	-	*	*	R	-	-
XOR	A,ear	2	3	1	0	byte (A) ← (A) xor (ear)	-	-	-	-	-	*	*	R	-	-
XOR	A,eam	2+	4+(a)	0	(b)	byte (A) ← (A) xor (eam)	-	-	-	-	-	*	*	R	-	-
XOR	ear,A	2	3	2	0	byte (ear) ← (ear) xor (A)	-	-	-	-	-	*	*	R	-	-
XOR	eam,A	2+	5+(a)	0	2 × (b)	byte (eam) ← (eam) xor (A)	-	-	-	-	-	*	*	R	-	*
NOT	A	1	2	0	0	byte (A) ← not (A)	-	-	-	-	-	*	*	R	-	-
NOT	ear	2	3	2	0	byte (ear) ← not (ear)	-	-	-	-	-	*	*	R	-	-
NOT	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← not (eam)	-	-	-	-	-	*	*	R	-	*
ANDW	A	1	2	0	0	word (A) ← (AH) and (A)	-	-	-	-	-	*	*	R	-	-
ANDW	A,#imm16	3	2	0	0	word (A) ← (A) and imm16	-	-	-	-	-	*	*	R	-	-
ANDW	A,ear	2	3	1	0	word (A) ← (A) and (ear)	-	-	-	-	-	*	*	R	-	-
ANDW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) and (eam)	-	-	-	-	-	*	*	R	-	-
ANDW	ear,A	2	3	2	0	word (ear) ← (ear) and (A)	-	-	-	-	-	*	*	R	-	-
ANDW	eam,A	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) and (A)	-	-	-	-	-	*	*	R	-	*
ORW	A	1	2	0	0	word (A) ← (AH) or (A)	-	-	-	-	-	*	*	R	-	-
ORW	A,#imm16	3	2	0	0	word (A) ← (A) or imm16	-	-	-	-	-	*	*	R	-	-
ORW	A,ear	2	3	1	0	word (A) ← (A) or (ear)	-	-	-	-	-	*	*	R	-	-
ORW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) or (eam)	-	-	-	-	-	*	*	R	-	-
ORW	ear,A	2	3	2	0	word (ear) ← (ear) or (A)	-	-	-	-	-	*	*	R	-	-
ORW	eam,A	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) or (A)	-	-	-	-	-	*	*	R	-	*

Table B-8 Logic 1 (Byte/Word): 39 Instructions (2 / 2)

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
XORW	A	1	2	0	0	word (A) ← (AH) xor (A)	-	-	-	-	-	*	*	R	-	-
XORW	A,#imm16	3	2	0	0	word (A) ← (A) xor imm16	-	-	-	-	-	*	*	R	-	-
XORW	A,ear	2	3	1	0	word (A) ← (A) xor (ear)	-	-	-	-	-	*	*	R	-	-
XORW	A,eam	2+	4+(a)	0	(c)	word (A) ← (A) xor (eam)	-	-	-	-	-	*	*	R	-	-
XORW	ear,A	2	3	2	0	word (ear) ← (ear) xor (A)	-	-	-	-	-	*	*	R	-	-
XORW	eam,A	2+	5+(a)	0	2 × (c)	word (eam) ← (eam) xor (A)	-	-	-	-	-	*	*	R	-	*
NOTW	A	1	2	0	0	word (A) ← not (A)	-	-	-	-	-	*	*	R	-	-
NOTW	ear	2	3	2	0	word (ear) ← not (ear)	-	-	-	-	-	*	*	R	-	-
NOTW	eam	2+	5+(a)	0	2 × (c)	word (eam) ← not (eam)	-	-	-	-	-	*	*	R	-	*

Note: See Table A-4 and Table A-5 for information on (a), (b) and (c) in the table.

Table B-9 Logic 2 (Long): 6 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
ANDL	A,ear	2	6	2	0	long (A) ← (A) and (ear)	-	-	-	-	-	*	*	R	-	-
ANDL	A,eam	2+	7+(a)	0	(d)	long (A) ← (A) and (eam)	-	-	-	-	-	*	*	R	-	-
ORL	A,ear	2	6	2	0	long (A) ← (A) or (ear)	-	-	-	-	-	*	*	R	-	-
ORL	A,eam	2+	7+(a)	0	(d)	long (A) ← (A) or (eam)	-	-	-	-	-	*	*	R	-	-
XORL	A,ear	2	6	2	0	long (A) ← (A) xor (ear)	-	-	-	-	-	*	*	R	-	-
XORL	A,eam	2+	7+(a)	0	(d)	long (A) ← (A) xor (eam)	-	-	-	-	-	*	*	R	-	-

Note: See Table A-4 and Table A-5 for information on (a) and (d) in the table.

Table B-10 Sign Inversion (Byte/Word): 6 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
NEG	A	1	2	0	0	byte (A) ← 0 - (A)	X	-	-	-	-	*	*	*	*	-
NEG	ear	2	3	2	0	byte (ear) ← 0 - (ear)	-	-	-	-	-	*	*	*	*	-
NEG	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← 0 - (eam)	-	-	-	-	-	*	*	*	*	*
NEGW	A	1	2	0	0	word (A) ← 0 - (A)	-	-	-	-	-	*	*	*	*	-
NEGW	ear	2	3	2	0	word (ear) ← 0 - (ear)	-	-	-	-	-	*	*	*	*	-
NEGW	eam	2+	5+(a)	0	2 × (c)	word (eam) ← 0 - (eam)	-	-	-	-	-	*	*	*	*	*

Note: See Table A-4 and Table A-5 for information on (a), (b) and (c) in the table.

Table B-11 Normalize Instruction (Long): 1 Instruction

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
NRML	A,R0	2	*1	1	0	long (A) ← Shift to the first bit which is set to 1 formerly placed byte (R0) ← Number of shifts at that time	-	-	-	-	-	-	*	-	-	-

*1: 4 when all accumulators have a value of 0; otherwise, 6+(R0)
Note: See Table A-3 and Table A-4 for information on (a), (b) and (c) in the table.

B.4 Shift Instruction

Table B-12 lists 18 shift instructions of F^{2}MC-16L.

■ Shift Instruction

Table B-12 Shift Instructions (Byte/Word/Long-word): 18 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
RORC	A	2	2	0	0	byte (A) ← Right rotate with carry	-	-	-	-	-	*	*	-	*	-
ROLC	A	2	2	0	0	byte (A) ← Left rotate with carry	-	-	-	-	-	*	*	-	*	-
RORC	ear	2	3	2	0	byte (ear) ← Right rotate with carry	-	-	-	-	-	*	*	-	*	-
RORC	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← Right rotate with carry	-	-	-	-	-	*	*	-	*	*
ROLC	ear	2	3	2	0	byte (ear) ← Left rotate with carry	-	-	-	-	-	*	*	-	*	-
ROLC	eam	2+	5+(a)	0	2 × (b)	byte (eam) ← Left rotate with carry	-	-	-	-	-	*	*	-	*	*
ASR	A,R0	2	*1	1	0	byte (A) ← Arithmetic right shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSR	A,R0	2	*1	1	0	byte (A) ← Logical right barrel shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSL	A,R0	2	*1	1	0	byte (A) ← Logical left barrel shift (A, R0)	-	-	-	-	-	*	*	-	*	-
ASRW	A	1	2	0	0	word (A) ← Arithmetic right shift (A, 1 bit)	-	-	-	-	*	*	*	-	*	-
LSRW	A/SHRW A	1	2	0	0	word (A) ← Logical right shift (A, 1 bit)	-	-	-	-	*	R	*	-	*	-
LSLW	A/SHLW A	1	2	0	0	word (A) ← Logical left shift (A, 1 bit)	-	-	-	-	-	*	*	-	*	-
ASRW	A,R0	2	*1	1	0	word (A) ← Arithmetic right barrel shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSRW	A,R0	2	*1	1	0	word (A) ← Logical right barrel shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSLW	A,R0	2	*1	1	0	word (A) ← Logical left barrel shift (A, R0)	-	-	-	-	-	*	*	-	*	-
ASRL	A,R0	2	*2	1	0	long (A) ← Arithmetic right barrel shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSRL	A,R0	2	*2	1	0	long (A) ← Logical right barrel shift (A, R0)	-	-	-	-	*	*	*	-	*	-
LSLL	A,R0	2	*2	1	0	long (A) ← Logical left barrel shift (A, R0)	-	-	-	-	-	*	*	-	*	-

*1: 6 when R0 is 0; otherwise, 5 + (R0)
*2: 6 when R0 is 0; otherwise, 6 + (R0)
Note: See Table A-4 and Table A-5 for information on (a) and (b) in the table.

B.5 Branch Instructions

The branch instructions of the F^2MC-16L are listed in the following two tables:
- **Table B-13 for branch 1: 31 instructions**
- **Table B-14 for branch 2: 19 instructions**

■ Branch Instruction

Table B-13 Branch 1: 31 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
BZ/BEQ	rel	2	*1	0	0	Branch on (Z) = 1	-	-	-	-	-	-	-	-	-	-
BNZ/BNE	rel	2	*1	0	0	Branch on (Z) = 0	-	-	-	-	-	-	-	-	-	-
BC/BLO	rel	2	*1	0	0	Branch on (C) = 1	-	-	-	-	-	-	-	-	-	-
BNC/BHS	rel	2	*1	0	0	Branch on (C) = 0	-	-	-	-	-	-	-	-	-	-
BN	rel	2	*1	0	0	Branch on (N) = 1	-	-	-	-	-	-	-	-	-	-
BP	rel	2	*1	0	0	Branch on (N) = 0	-	-	-	-	-	-	-	-	-	-
BV	rel	2	*1	0	0	Branch on (V) = 1	-	-	-	-	-	-	-	-	-	-
BNV	rel	2	*1	0	0	Branch on (V) = 0	-	-	-	-	-	-	-	-	-	-
BT	rel	2	*1	0	0	Branch on (T) = 1	-	-	-	-	-	-	-	-	-	-
BNT	rel	2	*1	0	0	Branch on (T) = 0	-	-	-	-	-	-	-	-	-	-
BLT	rel	2	*1	0	0	Branch on (V) xor (N) = 1	-	-	-	-	-	-	-	-	-	-
BGE	rel	2	*1	0	0	Branch on (V) xor (N) = 0	-	-	-	-	-	-	-	-	-	-
BLE	rel	2	*1	0	0	Branch on ((V) xor (N)) or (Z) = 1	-	-	-	-	-	-	-	-	-	-
BGT	rel	2	*1	0	0	Branch on ((V) xor (N)) or (Z) = 0	-	-	-	-	-	-	-	-	-	-
BLS	rel	2	*1	0	0	Branch on (C) or (Z) = 1	-	-	-	-	-	-	-	-	-	-
BHI	rel	2	*1	0	0	Branch on (C) or (Z) = 0	-	-	-	-	-	-	-	-	-	-
BRA	rel	2	*1	0	0	Unconditional branch	-	-	-	-	-	-	-	-	-	-
JMP	@A	1	2	0	0	word (PC) ← (A)	-	-	-	-	-	-	-	-	-	-
JMP	addr16	3	3	0	0	word (PC) ← addr16	-	-	-	-	-	-	-	-	-	-
JMP	@ear	2	3	1	0	word (PC) ← (ear)	-	-	-	-	-	-	-	-	-	-
JMP	@eam	2+	4+(a)	0	(c)	word (PC) ← (eam)	-	-	-	-	-	-	-	-	-	-
JMPP	@ear *3	2	5	2	0	word (PC) ← (ear), (PCB) ← (ear+2)	-	-	-	-	-	-	-	-	-	-
JMPP	@eam *3	2+	6+(a)	0	(d)	word (PC) ← (eam), (PCB) ← (eam+2)	-	-	-	-	-	-	-	-	-	-
JMPP	addr24	4	4	0	0	word (PC) ← ad24 0-15, (PCB) ← ad24 16-23	-	-	-	-	-	-	-	-	-	-
CALL	@ear *4	2	6	1	(c)	word (PC) ← (ear)	-	-	-	-	-	-	-	-	-	-
CALL	@eam *4	2+	7+(a)	0	2 × (c)	word (PC) ← (eam)	-	-	-	-	-	-	-	-	-	-
CALL	addr16 *5	3	6	0	(c)	word (PC) ← addr16	-	-	-	-	-	-	-	-	-	-
CALLV	#vct4 *5	1	7	0	2 × (c)	Vector call instruction	-	-	-	-	-	-	-	-	-	-
CALLP	@ear *6	2	10	2	2 × (c)	word (PC) ← (ear)0-15, (PCB) ← (ear)16-23	-	-	-	-	-	-	-	-	-	-
CALLP	@eam *6	2+	11+(a)	0	*2	word (PC) ← (eam)0-15, (PCB) ← (eam)16-23	-	-	-	-	-	-	-	-	-	-
CALLP	addr24 *7	4	10	0	2 × (c)	word (PC) ← addr0-15, (PCB) ← addr16-23	-	-	-	-	-	-	-	-	-	-

*1: 4 when branch is made, 6 when branch is not made
*2: 3 × (c) + (b)
*3: Read the branched address (word)
*4: W: Save to stack (word), R: Read the branched address (word)
*5: Save to stack (word)
*6: W: Save to stack (long), R: Read the branched address (long)
*7: Save to stack (long)
Note: See Table A-4 and for information on (a) to (d) in the table.

Table B-14 Branch 2: 19 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
CBNE	A,#imm8,rel	3	*1	0	0	Branch on byte (A) not equal to imm8	-	-	-	-	-	*	*	*	*	-
CWBNE	A,#imm16,rel	4	*1	0	0	Branch on word (A) not equal to imm16	-	-	-	-	-	*	*	*	*	-
CBNE	ear,#imm8,rel	4	*2	1	0	Branch on byte (ear) not equal to imm8	-	-	-	-	-	*	*	*	*	-
CBNE	eam,#imm8,rel *9	4+	*3	0	(b)	Branch on byte (eam) not equal to imm8	-	-	-	-	-	*	*	*	*	-
CWBNE	ear,#imm16,rel	5	*4	1	0	Branch on word (ear) not equal to imm16	-	-	-	-	-	*	*	*	*	-
CWBNE	eam,#imm16,rel*9	5+	*3	0	(c)	Branch on word (eam) not equal to imm16	-	-	-	-	-	*	*	*	*	-
DBNZ	ear,rel	3	*5	2	0	Branch on byte (ear) = (ear) - 1, (ear) not equal to 0	-	-	-	-	-	*	*	*	-	-
DBNZ	eam,rel	3+	*6	2	2 × (b)	Branch on byte (eam) = (eam) - 1, (eam) not equal to 0	-	-	-	-	-	*	*	*	-	*
DWBNZ	ear,rel	3	*5	2	0	Branch on word (ear) = (ear) - 1, (ear) not equal to 0	-	-	-	-	-	*	*	*	-	-
DWBNZ	eam,rel	3+	*6	2	2 × (c)	Branch on word (eam) = (eam) - 1, (eam) not equal to 0	-	-	-	-	-	*	*	*	-	*
INT	#vct8	2	20	0	8 × (c)	Software interrupt	-	-	R	S	-	-	-	-	-	-
INT	addr16	3	16	0	6 × (c)	Software interrupt	-	-	R	S	-	-	-	-	-	-
INTP	addr24	4	17	0	6 × (c)	Software interrupt	-	-	R	S	-	-	-	-	-	-
INT9		1	20	0	8 × (c)	Software interrupt	-	-	R	S	-	-	-	-	-	-
RETI		1	*8	0	*7	Return from interrupt	-	-	*	*	*	*	*	*	*	-
LINK	#imm8	2	6	0	(c)	At the entrance of function, save old frame pointers im stack, set up new frame pointers, reserve area for local pointers.	-	-	-	-	-	-	-	-	-	-
UNLINK		1	5	0	(c)	At the exit of function, recover the old frame pointers from the stack.	-	-	-	-	-	-	-	-	-	-
RET	*10	1	4	0	(c)	Return from subroutine	-	-	-	-	-	-	-	-	-	-
RETP	*11	1	6	0	(d)	Return from subroutine	-	-	-	-	-	-	-	-	-	-

*1: 5 when branch is made, 4 when branch is not made
*2: 13 when branch is made, 12 when branch is not made
*3: 7 + (a) when branch is made, 6 + (a) when branch is not made
*4: 8 when branch is made, 7 when branch is not made
*5: 7 when branch is made, 6 when branch is not made
*6: 8 + (a) when branch is made, 7 + (a) when branch is not made
*7: 3 × (b) + 2 × (c) when the sequence is branched to the next interrupt request, 6 × (c) when returned from the current interruption
*8: 15 when the sequence is branched to the next interrupt request, 17 when returned from the current interruption
*9: Do not use the RWj + addressing mode for the CBNE/CWBNE instruction.
*10: Return from stack (word)
*11: Return from stack (long)
Note: See Table A-5 and Table A-5 for information on (a) to (d) in the table.

B.6 Other Instructions

Other instructions of the F^{2}MC-16L are listed in the following four tables:

- **Table B-15 for other control systems (Byte/Word/Long-word) : 28 instructions**
- **Table B-16 for bit operation instructions: 21 instructions**
- **Table B-17 for accumulator operation instructions (Byte/Word): 6 instructions**
- **Table B-18 for string instructions: 10 instructions**

■ Other Instructions

Table B-15 Other Control Systems (Byte/Word/Long-word): 28 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
PUSHW	A	1	4	0	(c)	word (SP) ← (SP) - 2, ((SP)) ← (A)	-	-	-	-	-	-	-	-	-	-
PUSHW	AH	1	4	0	(c)	word (SP) ← (SP) - 2, ((SP)) ← (AH)	-	-	-	-	-	-	-	-	-	-
PUSHW	PS	1	4	0	(c)	word (SP) ← (SP) - 2, ((SP)) ← (PS)	-	-	-	-	-	-	-	-	-	-
PUSHW	rlst	2	*3	*5	*4	(SP) ← (SP) - 2n, ((SP)) ← (rlst)	-	-	-	-	-	-	-	-	-	-
POPW	A	1	3	0	(c)	word (A) ← ((SP)), (SP) ← (SP) + 2	-	*	-	-	-	-	-	-	-	-
POPW	AH	1	3	0	(c)	word (AH) ← ((SP)), (SP) ← (SP) + 2	-	-	-	-	-	-	-	-	-	-
POPW	PS	1	4	0	(c)	word (PS) ← ((SP)), (SP) ← (SP) + 2	-	-	*	*	*	*	*	*	*	-
POPW	rlst	2	*2	*5	*4	(rlst) ← ((SP)), (SP) ← (SP)	-	-	-	-	-	-	-	-	-	-
JCTX	@A	1	14	0	6 × (c)	Context switching instruction	-	-	*	*	*	*	*	*	*	-
AND	CCR,#imm8	2	3	0	0	byte (CCR) ← (CCR) and imm8	-	-	*	*	*	*	*	*	*	-
OR	CCR,#imm8	2	3	0	0	byte (CCR) ← (CCR) or imm8	-	-	*	*	*	*	*	*	*	-
MOV	RP,#imm8	2	2	0	0	byte (RP) ← imm8	-	-	-	-	-	-	-	-	-	-
MOV	ILM,#imm8	2	2	0	0	byte (ILM) ← imm8	-	-	-	-	-	-	-	-	-	-
MOVEA	RWi,ear	2	3	1	0	word (RWi) ← ear	-	-	-	-	-	-	-	-	-	-
MOVEA	RWi,eam	2+	2+(a)	1	0	word (RWi) ← eam	-	-	-	-	-	-	-	-	-	-
MOVEA	A,ear	2	1	0	0	word (A) ← ear	-	*	-	-	-	-	-	-	-	-
MOVEA	A,eam	2+	1+(a)	0	0	word (A) ← eam	-	*	-	-	-	-	-	-	-	-
ADDSP	#imm8	2	3	0	0	word (SP) ← ext(imm8)	-	-	-	-	-	-	-	-	-	-
ADDSP	#imm16	3	3	0	0	word (SP) ← imm16	-	-	-	-	-	-	-	-	-	-
MOV	A,brg1	2	*1	0	0	byte (A) ← (brg1)	Z	*	-	-	-	*	*	-	-	-
MOV	brg2,A	2	1	0	0	byte (brg2) ← (A)	-	-	-	-	-	*	*	-	-	-
NOP		1	1	0	0	No operation	-	-	-	-	-	-	-	-	-	-
ADB		1	1	0	0	Prefix code for AD space access	-	-	-	-	-	-	-	-	-	-
DTB		1	1	0	0	Prefix code for DT space access	-	-	-	-	-	-	-	-	-	-
PCB		1	1	0	0	Prefix code for PC space access	-	-	-	-	-	-	-	-	-	-
SPB		1	1	0	0	Prefix code for SP space access	-	-	-	-	-	-	-	-	-	-
NCC		1	1	0	0	Prefix code for flag unchange setting	-	-	-	-	-	-	-	-	-	-
CMR		1	1	0	0	Prefix for common register banks	-	-	-	-	-	-	-	-	-	-

*1: 1 for PCB, ADB, SSB, USB, and SPB 2 for DTB and DPR
*2: 7 + 3 × (number of POP operations) + 2 × (number of the last register that operates POP), 7 when RLST = 0 (no transfer register)
*3: 29 + 3 × (number of PUSH operations) - 3 × (number of the last register that operates PUSH), 8 when RLST = 0 (no transfer register)
*4: (number of POP operations) × c, or (number of PUSH operations) × c
*5: (number of POP operations), or (number of PUSH operations)
Note: See Table A-4 and Table A-5 for information on (a) and (c) in the table.

Table B-16 Bit Operation Instruction: 21 Instructions

Mnemonic		#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
MOVB	A,dir:bp	3	5	0	(b)	byte (A) ← (dir:bp)b	Z	*	-	-	-	*	*	-	-	-
MOVB	A,addr16:bp	4	5	0	(b)	byte (A) ← (addr16:bp)b	Z	*	-	-	-	*	*	-	-	-
MOVB	A,io:bp	3	4	0	(b)	byte (A) ← (io:bp)b	Z	*	-	-	-	*	*	-	-	-
MOVB	dir:bp,A	3	7	0	2 × (b)	bit (dir:bp)b ← (A)	-	-	-	-	-	*	*	-	-	*
MOVB	addr16:bp,A	4	7	0	2 × (b)	bit (addr16:bp)b ← (A)	-	-	-	-	-	*	*	-	-	*
MOVB	io:bp,A	3	6	0	2 × (b)	bit (io:bp)b ← (A)	-	-	-	-	-	*	*	-	-	*
SETB	dir:bp	3	7	0	2 × (b)	bit (dir:bp)b ← 1	-	-	-	-	-	-	-	-	-	*
SETB	addr16:bp	4	7	0	2 × (b)	bit (addr16:bp)b ← 1	-	-	-	-	-	-	-	-	-	*
SETB	io:bp	3	7	0	2 × (b)	bit (io:bp)b ← 1	-	-	-	-	-	-	-	-	-	*
CLRB	dir:bp	3	7	0	2 × (b)	bit (dir:bp)b ← 0	-	-	-	-	-	-	-	-	-	*
CLRB	addr16:bp	4	7	0	2 × (b)	bit (addr16:bp)b ← 0	-	-	-	-	-	-	-	-	-	*
CLRB	io:bp	3	7	0	2 × (b)	bit (io:bp)b ← 0	-	-	-	-	-	-	-	-	-	*
BBC	dir:bp,rel	4	*1	0	(b)	Branch on (dir:bp) b = 0	-	-	-	-	-	-	*	-	-	-
BBC	addr16:bp,rel	5	*1	0	(b)	Branch on (addr16:bp) b = 0	-	-	-	-	-	-	*	-	-	-
BBC	io:bp,rel	4	*2	0	(b)	Branch on (io:bp) b = 0	-	-	-	-	-	-	*	-	-	-
BBS	dir:bp,rel	4	*1	0	(b)	Branch on (dir:bp) b = 1	-	-	-	-	-	-	*	-	-	-
BBS	addr16:bp,rel	5	*1	0	(b)	Branch on (addr16:bp) b = 1	-	-	-	-	-	-	*	-	-	-
BBS	io:bp,rel	4	*2	0	(b)	Branch on (io:bp) b = 1	-	-	-	-	-	-	*	-	-	-
SBBS	addr16:bp,rel	5	*3	0	2 × (b)	Branch on (addr16:bp) b = 1, bit = 1	-	-	-	-	-	-	*	-	-	*
WBTS	io:bp	3	*4	0	*5	Waits until (io:bp) b = 1	-	-	-	-	-	-	-	-	-	-
WBTC	io:bp	3	*4	0	*5	Waits until (io:bp) b = 0	-	-	-	-	-	-	-	-	-	-

*1: 8 when branch is made, 7 when branch is not made
*2: 7 when branch is made, 6 when branch is not made
*3: 10 when the condition is met, 9 when the condition is not met
*4: Undefined count
*5: Until the condition is met
Note: See Table A-5 for information on (b) in the table.

Table B-17 Accumulator Operation Instruction (Byte/Word): 6 Instructions

Mnemonic	#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
SWAP	1	3	0	0	byte (A)0-7 ↔ (A)8-15	-	-	-	-	-	-	-	-	-	-
SWAPW	1	2	0	0	word (AH) ↔ (AL)	-	*	-	-	-	-	-	-	-	-
EXT	1	1	0	0	Byte signed extension	X	-	-	-	-	*	*	-	-	-
EXTW	1	2	0	0	Word signed extension	-	X	-	-	-	*	*	-	-	-
ZEXT	1	1	0	0	Byte zero extension	Z	-	-	-	-	R	*	-	-	-
ZEXTW	1	1	0	0	Word zero extension	-	Z	-	-	-	R	*	-	-	-

Table B-18 String Instruction : 10 Instructions

Mnemonic	#	~	RG	B	Operation	LH	AH	I	S	T	N	Z	V	C	RMW
MOVS / MOVSI	2	*2	*5	*3	byte transfer @AH+ ← @AL+, counter = RW0	-	-	-	-	-	-	-	-	-	-
MOVSD	2	*2	*5	*3	byte transfer @AH- ← @AL-, counter = RW0	-	-	-	-	-	-	-	-	-	-
SCEQ / SCEQI	2	*1	*8	*4	byte search @AH+ ← AL, counter RW0	-	-	-	-	-	*	*	*	*	-
SCEQD	2	*1	*8	*4	byte search @AH- ← AL, counter RW0	-	-	-	-	-	*	*	*	*	-
FILS / FILSI	2	6m+6	*8	*3	byte fill @AH+ ← AL, counter RW0	-	-	-	-	-	*	*	-	-	-
MOVSW / MOVSWI	2	*2	*5	*6	word transfer @AH+ ← @AL+, counter = RW0	-	-	-	-	-	-	-	-	-	-
MOVSWD	2	*2	*5	*6	word transfer @AH- ← @AL-, counter = RW0	-	-	-	-	-	-	-	-	-	-
SCWEQ / SCWEQI	2	*1	*8	*7	word search @AH+ - AL, counter = RW0	-	-	-	-	-	*	*	*	*	-
SCWEQD	2	*1	*8	*7	word search @AH- - AL, counter = RW0	-	-	-	-	-	*	*	*	*	-
FILSW / FILSWI	2	6m+6	*8	*6	word fill @AH+ ← AL, counter = RW0	-	-	-	-	-	*	*	-	-	-

*1: 5 when RW0 is 0, 4 + 7 × (RW0) when count-out is detected, and 7n + 5 when the data in the AL register matches the byte data specified by the AH register in the space that is specified by bank
*2: 5 when RW0 is 0, 4 + 8 × (RW0) for otherwise
*3: (b) × (RW0) + (b) × (RW0):To access different areas for sources and destinations, calculate item (b) separately each other.
*4: (b) × n
*5: 2 × (b) × (RW0)
*6: (c) × (RW0) + (c) × (RW0):To access different areas for sources and destinations, calculate item (c) separately each other.
*7: (c) × n
*8: (b) × (RW0)

Notes: • m: RW0 value (counter value), n: Number of loops
• See Table A-5 for information on (b) and (c) in the table.

APPENDIX C F^{2}MC-16LX Instruction Maps

This appendix describes F^{2}MC-16LX instruction maps.

C.1 Structure of the Instruction Map
C.2 Basic Page Map
C.3 Bit Operation Instruction Map
C.4 Character String Operation Instruction Map
C.5 2-byte Instruction Map
C.6 ea-type Instruction Map
C.7 MOVEA RWi, ea Instruction Map
C.8 MOV Ri, ea Instruction Map
C.9 MOVW RWi, ea Instruction Map
C.10 MOV ea, Ri Instruction Map
C.11 MOVW ea, RWi Instruction Map
C.12 XCH Ri, ea Instruction Map
C.13 XCHW RWi, ea Instruction Map

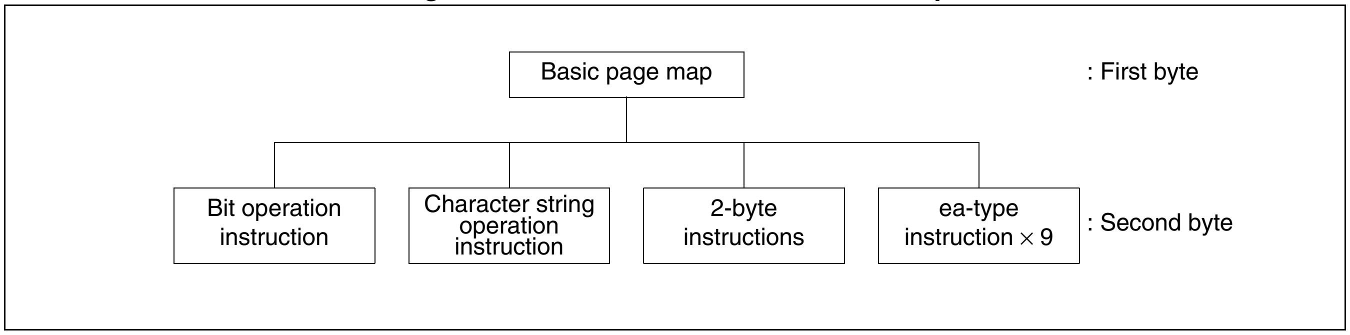
C.1 Structure of the Instruction Map

Since the instruction code of the F^2MC-16LX consists of one- and two-byte instructions, the instruction map consists of more than one page that can be used for one- and two-byte instructions.

■ Structure of the Instruction Map

Figure C-1 shows the structure of the instruction map.

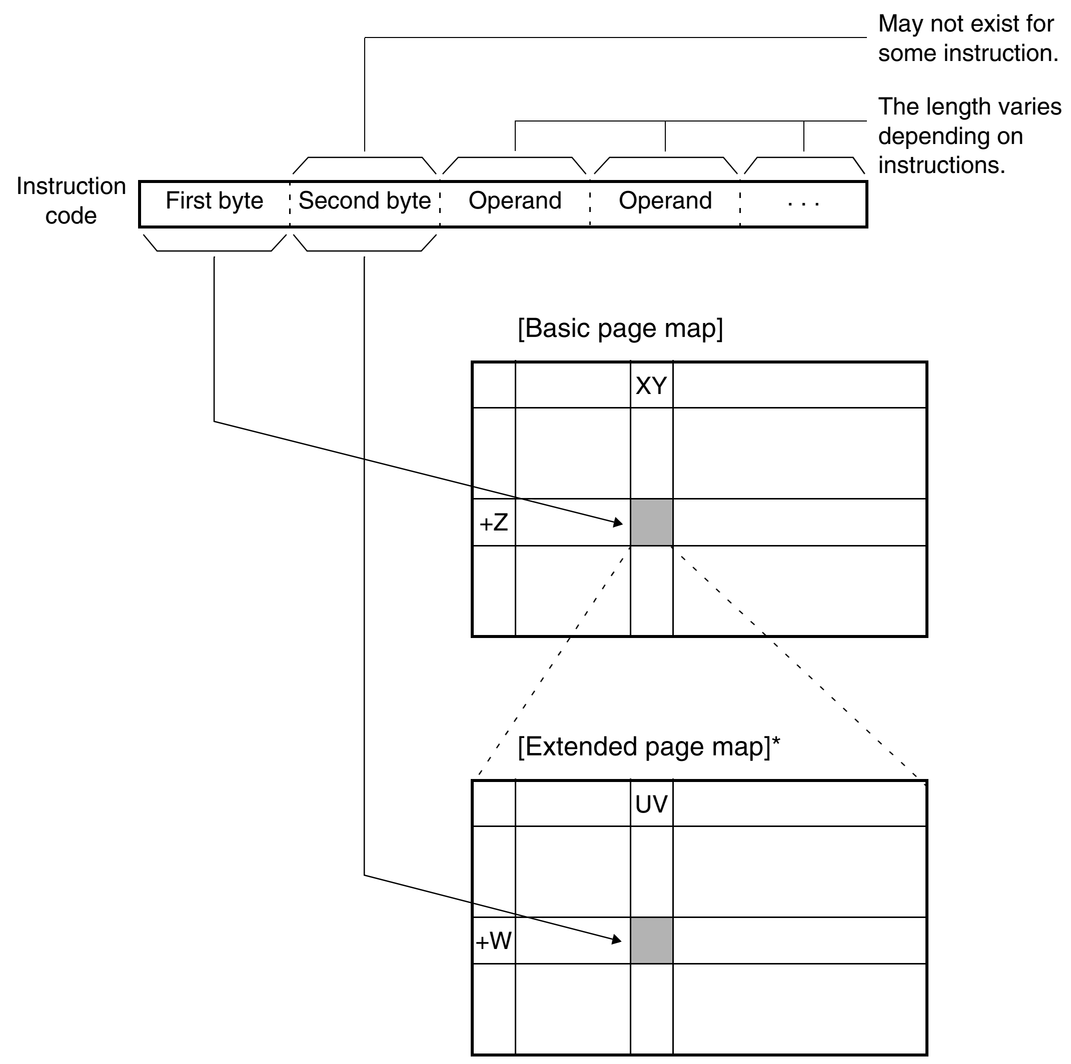
Figure C-1 Structure of the Instruction Map



The instruction code is described on the basic page map for one-byte instructions (such as the NOP instruction). For two-byte instructions (such as the MOVS instruction), see the basic page map to find the name of the map that describes the second byte of the instruction code to be referenced next.

Figure C-2 shows the relationship between actual instruction codes and instruction maps.

Figure C-2 Relationship Between Actual Instruction Codes and Instruction Maps



*: Extended page map is a generic name for bit operation instruction, character string operation instruction, 2-byte instruction, and ea-type instruction. More than one extended page map exists for each type of instruction.

C.2 Basic Page Map

Table C-1 shows the basic page map.

■ Basic Page Map

Table C-1 Basic Page Map

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	NOP	CMR	ADD A, dir	ADD A, #8	MOV A, dir	MOV A, io	BRA rel	ea instruction 1	MOV A, Ri	MOV Ri, A	MOV Ri, #8	MOV A, Ri	MOVX A, @RWi+d8	MOV A, #4	CALL #4	BZ/BEQ rel
+1	INT9	NCC	SUB A, dir	SUB A, #8	MOV dir, A	MOV io, A	JMP @A	ea instruction 2								BNZ/BNE rel
+2	ADDDC A	SUBDC A	ADDC A	SUBC A	MOV A, #8	MOV A, addr16	JMP addr16	ea instruction 3								BC/BL0 rel
+3	NEG A	JCTX @A	CMP A	CMP A, #8	MOVX A, #8	MOV addr16, A	JMPP adde24	ea instruction 4								BNC/BHS rel
+4	PCB	EXT	AND CCR, #8	AND A, #8	MOV dir, #8	MOV io, #8	CALL addr16	ea instruction 5								BN rel
+5	DTB	ZEXT	OR CCR, #8	OR A, #8	MOVX A, dir	MOVX A, io	CALLP addr24	ea instruction 6								BP rel
+6	ADB	SWAP	DIVU A	XOR A, #8	MOVW A, SP	MOVW io, #16	RETP	ea instruction 7								BV rel
+7	SPB	ADDSP #8	MULU A	NOT A	MOVW SP, A	MOVX A, addr16	RET	ea instruction 8								BNV rel
+8	LINK #imm8	ADDL A, #32	ADDW A	ADDW A, #16	MOVW A, dir	MOVW A, io	INT #vct8	ea instruction 9	MOVW A, RWi	MOVW RWi, A	MOVW RWi, #16	MOV A, @RWi+d8	MOVW @RWi+d8, A			BT rel
+9	UNLINK	SUBL A, #32	SUBW A	SUBW A, #16	MOVW dir, A	MOVW io, A	INT addr16	MOVEA RWi, ea								BNT rel
+A	MOV RP, #8	MOV ILM, #8	CBNE A, #8, rel	CWBNE A, #16, rel	MOVW A, #16	MOVW A,addr16	INTP addr24	MOV Ri, ea								BLT rel
+B	NEGW A	CMPL A, #32	CMPW A	CMPW A, #16	MOVL A, #32	MOVW addr16, A	RETI	MOVW RWi, ea								BGE rel
+C	LSLW A	EXTW	ANDW A	ANDW A, #16	PUSHW A	POPW A	Bit operation instruction	MOV ea, Ri								BLE rel
+D		ZEXTW	ORW A	ORW A, #16	PUSHW AH	POPW AH		MOVW ea, RWi								BGT rel
+E	ASRW A	SWAPW	XORW A	XORW A, #16	PUSHW PS	POPW PS	Character string operation instruction	XCH Ri, ea								BLS rel
+F	LSRW A	ADDSP #16	MULUW A	NOTW A	PUSHW rlst	POPW rlst	2-byte instruction	XCHW RWi, ea								BHI rel

Note: For the information about ea-type instruction from (1) to (9), see Table C.6,"ea-type Instruction".

C.3 Bit Operation Instruction Map

Table C-2 shows the bit operation instruction map.

■ Bit Operation Instruction Map

Table C-2 Bit Operation Instruction Map (First Byte = $6C_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOVB A, io:bp		MOVB io:bp, A		CLRB io:bp		SETB io:bp		BBC io;bp, rel		BBS io:bp, rel		WBTS io:bp		WBTC io:bp	
+1	↓		↓		↓		↓		↓		↓		↓		↓	
+2	↓		↓		↓		↓		↓		↓		↓		↓	
+3	↓		↓		↓		↓		↓		↓		↓		↓	
+4	↓		↓		↓		↓		↓		↓		↓		↓	
+5	↓		↓		↓		↓		↓		↓		↓		↓	
+6	↓		↓		↓		↓		↓		↓		↓		↓	
+7	↓		↓		↓		↓		↓		↓		↓		↓	
+8	MOVB A, dir:bp	MOVB A, addr16:bp	MOVB dir:bp, A	MOVB addr16:bp,A	CLRB dir:bp	CLRB addr16:bp	SETB dir:bp	SETB addr16:bp	BBC dir:bp, rel	BBC addr16:bp,rel	BBS dir:bp, rel	BBS addr16:bp,rel				SBBS addr16:bp
+9	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+A	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+B	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+C	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+D	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+E	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓
+F	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓				↓

C.4 Character String Operation Instruction Map

Table C-3 shows the character string operation instruction map.

■ Character String Operation Instruction Map

Table C-3 Character String Operation Instruction Map (First Byte = 6E$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOVSI PCB, PCB	MOVSD →	MOVSWI →	MOVSWD →					SCEQI PCB	SCEQD PCB	SCWEQI PCB	SCWEQD PCB	FILSI PCB		FILSI PCB	
+1	PCB, DTB	↓	↓	↓					DTB	DTB	DTB	DTB	DTB		DTB	
+2	PCB, ADB	↓	↓	↓					ADB	ADB	ADB	ADB	ADB		ADB	
+3	PCB, SPB	↓	↓	↓					SPB	SPB	SPB	SPB	SPB		SPB	
+4	DTB, PCB	↓	↓	↓												
+5	DTB, DTB	↓	↓	↓												
+6	DTB, ADB	↓	↓	↓												
+7	DTB, SPB	↓	↓	↓												
+8	ADB, PCB	↓	↓	↓												
+9	ADB, DTB	↓	↓	↓												
+A	ADB, ADB	↓	↓	↓												
+B	ADB, SPB	↓	↓	↓												
+C	SPB, PCB	↓	↓	↓												
+D	SPB, DTB	↓	↓	↓												
+E	SPB, ADB	↓	↓	↓												
+F	SPB, SPB	↓	↓	↓												

C.5 2-byte Instruction Map

Table C-4 shows the 2-byte instruction map.

■ 2-byte Instruction Map

Table C-4 2-byte Instruction Map (First Byte = 6F$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOV A, DTB	MOV DTB, A	MOVX A, @RL0+d8	MOV @RL0+d8, A	MOV A, @RL0+d8											
+1	MOV A, ADB	MOV ADB, A														
+2	MOV A, SSB	MOV SSB, A	MOVX A, @RL1+d8	MOV @RL1+d8, A	MOV A, @RL1+d8											
+3	MOV A, USB	MOV USB, A														
+4	MOV A, DPR	MOV DPR, A	MOVX A, @RL2+d8	MOV @RL2+d8, A	MOV A, @RL2+d8											
+5	MOV A, @A	MOV @AL, AH														
+6	MOV A, PCB	MOV A, @A	MOVX A, @RL3+d8	MOV @RL3+d8, A	MOV A, @RL3+d8											
+7	ROLC A	ROLC A														
+8				MOVW @RL0+d8, A	MOVW A, @RL0+d8		MUL A									
+9							MULW A									
+A				MOVW @RL1+d8, A	MOVW A, @RL1+d8		DIVU A									
+B																
+C	LSLW A, R0	LSLL A, R0	LSL A, R0	MOVW @RL2+d8, A	MOVW A, @RL2+d8											
+D	MOVW A, @A	MOVW @AL, AH	NRML A, R0													
+E	ASRW A, R0	ASRL A, R0	ASR A, R0	MOVW @RL3+d8, A	MOVW A, @RL3+d8											
+F	LSRW A, R0	LSRL A, R0	LSR A, R0													

C.6 ea-type Instruction Map

ea-type instruction maps (first byte = 70_H to first byte = 78_H) are shown in the following nine tables:

- **Table C-5 for ea-type instruction (1) (first byte = 70_H)**
- **Table C-6 for ea-type instruction (2) (first byte = 71_H)**
- **Table C-7 for ea-type instruction (3) (first byte = 72_H)**
- **Table C-8 for ea-type instruction (4) (first byte = 73_H)**
- **Table C-9 for ea-type instruction (5) (first byte = 74_H)**
- **Table C-10 for ea-type instruction (6) (first byte = 75_H)**
- **Table C-11 for ea-type instruction (7) (first byte = 76_H)**
- **Table C-12 for ea-type instruction (8) (first byte = 77_H)**
- **Table C-13 for ea-type instruction (9) (first byte = 78_H)**

■ ea-type Instruction Map

Table C-5 ea-byte Instruction (1) (First Byte = 70$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
					CWBNE ↓	CWBNE ↓									CBNE ↓	CBNE ↓
+0	ADDL A, RL0	ADDL A, @RW0+d8	SUBL A, RL0	SUBL A, @RW0+d8	RW0, #16, rel	@RW0+d8 #16, rel	CMPL A, RL0	CMPL A, @RW0+d8	ANDL A, RL0	ANDL A, @RW0+d8	ORL A, RL0	ORL A, @RW0+d8	XORL A, RL0	XORL A, @RW0+d8	R0, #8, rel	@RW0+d8, #8, rel
+1	ADDL A, RL0	ADDL A, @RW1+d8	SUBL A, RL0	SUBL A, @RW1+d8	RW1, #16, rel	@RW1+d8 #16, rel	CMPL A, RL0	CMPL A, @RW1+d8	ANDL A, RL0	ANDL A, @RW1+d8	ORL A, RL0	ORL A, @RW1+d8	XORL A, RL0	XORL A, @RW1+d8	R1, #8, rel	@RW1+d8, #8, rel
+2	ADDL A, RL1	ADDL A, @RW2+d8	SUBL A, RL1	SUBL A, @RW2+d8	RW2, #16, rel	@RW2+d8 #16, rel	CMPL A, RL1	CMPL A, @RW2+d8	ANDL A, RL1	ANDL A, @RW2+d8	ORL A, RL1	ORL A, @RW2+d8	XORL A, RL1	XORL A, @RW2+d8	R2, #8, rel	@RW2+d8, #8, rel
+3	ADDL A, RL1	ADDL A, @RW3+d8	SUBL A, RL1	SUBL A, @RW3+d8	RW3, #16, rel	@RW3+d8 #16, rel	CMPL A, RL1	CMPL A, @RW3+d8	ANDL A, RL1	ANDL A, @RW3+d8	ORL A, RL1	ORL A, @RW3+d8	XORL A, RL1	XORL A, @RW3+d8	R3, #8, rel	@RW3+d8, #8, rel
+4	ADDL A, RL2	ADDL A, @RW4+d8	SUBL A, RL2	SUBL A, @RW4+d8	RW4, #16, rel	@RW4+d8 #16, rel	CMPL A, RL2	CMPL A, @RW4+d8	ANDL A, RL2	ANDL A, @RW4+d8	ORL A, RL2	ORL A, @RW4+d8	XORL A, RL2	XORL A, @RW4+d8	R4, #8, rel	@RW4+d8, #8, rel
+5	ADDL A, RL2	ADDL A, @RW5+d8	SUBL A, RL2	SUBL A, @RW5+d8	RW5, #16, rel	@RW5+d8 #16, rel	CMPL A, RL2	CMPL A, @RW5+d8	ANDL A, RL2	ANDL A, @RW5+d8	ORL A, RL2	ORL A, @RW5+d8	XORL A, RL2	XORL A, @RW5+d8	R5, #8, rel	@RW5+d8, #8, rel
+6	ADDL A, RL3	ADDL A, @RW6+d8	SUBL A, RL3	SUBL A, @RW6+d8	RW6, #16, rel	@RW6+d8 #16, rel	CMPL A, RL3	CMPL A, @RW6+d8	ANDL A, RL3	ANDL A, @RW6+d8	ORL A, RL3	ORL A, @RW6+d8	XORL A, RL3	XORL A, @RW6+d8	R6, #8, rel	@RW6+d8, #8, rel
+7	ADDL A, RL3	ADDL A, @RW7+d8	SUBL A, RL3	SUBL A, @RW7+d8	RW7, #16, rel	@RW7+d8 #16, rel	CMPL A, RL3	CMPL A, @RW7+d8	ANDL A, RL3	ANDL A, @RW7+d8	ORL A, RL3	ORL A, @RW7+d8	XORL A, RL3	XORL A, @RW7+d8	R7, #8, rel	@RW7+d8, #8, rel
+8	ADDL A,@RW0	ADDL A, @RW0+d16	SUBL A,@RW0	SUBL A, @RW0+d16	@RW0, #16, rel	@RW0+d16 #16, rel	CMPL A,@RW0	CMPL A, @RW0+d16	ANDL A,@RW0	ANDL A, @RW0+d16	ORL A,@RW0	ORL A, @RW0+d16	XORL A,@RW0	XORL A, @RW0+d16	@RW0, #8, rel	@RW0+d16 ,#8, rel
+9	ADDL A,@RW1	ADDL A, @RW1+d16	SUBL A,@RW1	SUBL A, @RW1+d16	@RW1, #16, rel	@RW1+d16 #16, rel	CMPL A,@RW1	CMPL A, @RW1+d16	ANDL A,@RW1	ANDL A, @RW1+d16	ORL A,@RW1	ORL A, @RW1+d16	XORL A,@RW1	XORL A, @RW1+d16	@RW1, #8, rel	@RW1+d16 ,#8, rel
+A	ADDL A,@RW2	ADDL A, @RW2+d16	SUBL A,@RW2	SUBL A, @RW2+d16	@RW2, #16, rel	@RW2+d16 #16, rel	CMPL A,@RW2	CMPL A, @RW2+d16	ANDL A,@RW2	ANDL A, @RW2+d16	ORL A,@RW2	ORL A, @RW2+d16	XORL A,@RW2	XORL A, @RW2+d16	@RW2, #8, rel	@RW2+d16 ,#8, rel
+B	ADDL A,@RW3	ADDL A, @RW3+d16	SUBL A,@RW3	SUBL A, @RW3+d16	@RW3, #16, rel	@RW3+d16 #16, rel	CMPL A,@RW3	CMPL A, @RW3+d16	ANDL A,@RW3	ANDL A, @RW3+d16	ORL A,@RW3	ORL A, @RW3+d16	XORL A,@RW3	XORL A, @RW3+d16	@RW3, #8, rel	@RW3+d16 ,#8, rel
+C	ADDL A,@RW0+	ADDL A, @RW0+RW7	SUBL A,@RW0+	SUBL A, @RW0+RW7	Use prohibited	@RW0+RW7 #16, rel	CMPL A,@RW0+	CMPL A, @RW0+RW7	ANDL A,@RW0+	ANDL A, @RW0+RW7	ORL A,@RW0+	ORL A, @RW0+RW7	XORL A,@RW0+	XORL A, @RW0+RW7	Use prohibited	@RW0+RW7 ,#8, rel
+D	ADDL A,@RW1+	ADDL A, @RW1+RW7	SUBL A,@RW1+	SUBL A, @RW1+RW7	Use prohibited	@RW1+RW7 #16, rel	CMPL A,@RW1+	CMPL A, @RW1+RW7	ANDL A,@RW1+	ANDL A, @RW1+RW7	ORL A,@RW1+	ORL A, @RW1+RW7	XORL A,@RW1+	XORL A, @RW1+RW7	Use prohibited	@RW1+RW7 ,#8, rel
+E	ADDL A,@RW2+	ADDL A, @PC+d16	SUBL A,@RW2+	SUBL A, @PC+d16	Use prohibited	@PC+d16, #16, rel	CMPL A,@RW2+	CMPL A, @PC+d16	ANDL A,@RW2+	ANDL A, @PC+d16	ORL A,@RW2+	ORL A, @PC+d16	XORL A,@RW2+	XORL A, @PC+d16	Use prohibited	@PC+d16, ,#8, rel
+F	ADDL A,@RW3+	ADDL A, addr16	SUBL A,@RW3+	SUBL A, addr16	Use prohibited	addr16, #16, rel	CMPL A,@RW3+	CMPL A, addr16	ANDL A,@RW3+	ANDL A, addr16	ORL A,@RW3+	ORL A, addr16	XORL A,@RW3+	XORL A, addr16	Use prohibited	addr16, ,#8, rel

Table C-6 ea-type Instruction (2) (First Byte = 71_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	JMPP @RL0	JMPP @@RW0+d8	CALLP @RL0	CALLP @@RW0+d8	INCL RL0	INCL @RW0+d8	DECL RL0	DECL @RW0+d8	MOVL A, RL0	MOVL A, @RW0+d8	MOVL RL0, A	MOVL @RW0+d8,A	MOV R0, #8	MOV @RW0+d8,#8	MOVEA A, RW0	MOVEA A, @RW0+d8
+1	JMPP @RL0	JMPP @@RW1+d8	CALLP @RL0	CALLP @@RW1+d8	INCL RL0	INCL @RW1+d8	DECL RL0	DECL @RW1+d8	MOVL A, RL0	MOVL A, @RW1+d8	MOVL RL0, A	MOVL @RW1+d8,A	MOV R1, #8	MOV @RW1+d8,#8	MOVEA A, RW1	MOVEA A, @RW1+d8
+2	JMPP @RL1	JMPP @@RW2+d8	CALLP @RL1	CALLP @@RW2+d8	INCL RL1	INCL @RW2+d8	DECL RL1	DECL @RW2+d8	MOVL A, RL1	MOVL A, @RW2+d8	MOVL RL1, A	MOVL @RW2+d8,A	MOV R2, #8	MOV @RW2+d8,#8	MOVEA A, RW2	MOVEA A, @RW2+d8
+3	JMPP @RL1	JMPP @@RW3+d8	CALLP @RL1	CALLP @@RW3+d8	INCL RL1	INCL @RW3+d8	DECL RL1	DECL @RW3+d8	MOVL A, RL1	MOVL A, @RW3+d8	MOVL RL1, A	MOVL @RW3+d8,A	MOV R3, #8	MOV @RW3+d8,#8	MOVEA A, RW3	MOVEA A, @RW3+d8
+4	JMPP @RL2	JMPP @@RW4+d8	CALLP @RL2	CALLP @@RW4+d8	INCL RL2	INCL @RW4+d8	DECL RL2	DECL @RW4+d8	MOVL A, RL2	MOVL A, @RW4+d8	MOVL RL2, A	MOVL @RW4+d8,A	MOV R4, #8	MOV @RW4+d8,#8	MOVEA A, RW4	MOVEA A, @RW4+d8
+5	JMPP @RL2	JMPP @@RW5+d8	CALLP @RL2	CALLP @@RW5+d8	INCL RL2	INCL @RW5+d8	DECL RL2	DECL @RW5+d8	MOVL A, RL2	MOVL A, @RW5+d8	MOVL RL2, A	MOVL @RW5+d8,A	MOV R5, #8	MOV @RW5+d8,#8	MOVEA A, RW5	MOVEA A, @RW5+d8
+6	JMPP @RL3	JMPP @@RW6+d8	CALLP @RL3	CALLP @@RW6+d8	INCL RL3	INCL @RW6+d8	DECL RL3	DECL @RW6+d8	MOVL A, RL3	MOVL A, @RW6+d8	MOVL RL3, A	MOVL @RW6+d8,A	MOV R6, #8	MOV @RW6+d8,#8	MOVEA A, RW6	MOVEA A, @RW6+d8
+7	JMPP @RL3	JMPP @@RW7+d8	CALLP @RL3	CALLP @@RW7+d8	INCL RL3	INCL @RW7+d8	DECL RL3	DECL @RW7+d8	MOVL A, RL3	MOVL A, @RW7+d8	MOVL RL3, A	MOVL @RW7+d8,A	MOV R7, #8	MOV @RW7+d8,#8	MOVEA A, RW7	MOVEA A, @RW7+d8
+8	JMPP @@RW0	JMPP @ @RW0+d16	CALLP @@RW0	CALLP @ @RW0+d16	INCL @RW0	INCL @RW0+d16	DECL @RW0	DECL @RW0+d16	MOVL A,@RW0	MOVL A, @RW0+d16	MOVL @RW0,A	MOVL @RW0+d16,A	MOV @RW0, #8	MOV @RW0+d16,#8	MOVEA A,@RW0	MOVEA A, @RW0+d16
+9	JMPP @@RW1	JMPP @ @RW1+d16	CALLP @@RW1	CALLP @ @RW1+d16	INCL @RW1	INCL @RW1+d16	DECL @RW1	DECL @RW1+d16	MOVL A,@RW1	MOVL A, @RW1+d16	MOVL @RW1,A	MOVL @RW1+d16,A	MOV @RW1, #8	MOV @RW1+d16,#8	MOVEA A,@RW1	MOVEA A, @RW1+d16
+A	JMPP @@RW2	JMPP @ @RW2+d16	CALLP @@RW2	CALLP @ @RW2+d16	INCL @RW2	INCL @RW2+d16	DECL @RW2	DECL @RW2+d16	MOVL A,@RW2	MOVL A, @RW2+d16	MOVL @RW2,A	MOVL @RW2+d16,A	MOV @RW2, #8	MOV @RW2+d16,#8	MOVEA A,@RW2	MOVEA A, @RW2+d16
+B	JMPP @@RW3	JMPP @ @RW3+d16	CALLP @@RW3	CALLP @ @RW3+d16	INCL @RW3	INCL @RW3+d16	DECL @RW3	DECL @RW3+d16	MOVL A,@RW3	MOVL A, @RW3+d16	MOVL @RW3,A	MOVL @RW3+d16,A	MOV @RW3, #8	MOV @RW3+d16,#8	MOVEA A,@RW3	MOVEA A, @RW3+d16
+C	JMPP @@RW0+	JMPP @ @RW0+RW7	CALLP @@RW0+	CALLP @ @RW0+RW7	INCL @RW0+	INCL @RW0+RW7	DECL @RW0+	DECL @RW0+RW7	MOVL A,@RW0+	MOVL A, @RW0+RW7	MOVL @RW0+,A	MOVL @RW0+RW7,A	MOV @RW0+, #8	MOV @RW0+RW7,#8	MOVEA A,@RW0+	MOVEA A, @RW0+RW7
+D	JMPP @@RW1+	JMPP @ @RW1+RW7	CALLP @@RW1+	CALLP @ @RW1+RW7	INCL @RW1+	INCL @RW1+RW7	DECL @RW1+	DECL @RW1+RW7	MOVL A,@RW1+	MOVL A, @RW1+RW7	MOVL @RW1+,A	MOVL @RW1+RW7,A	MOV @RW1+, #8	MOV @RW1+RW7,#8	MOVEA A,@RW1+	MOVEA A, @RW1+RW7
+E	JMPP @@RW2+	JMPP @@PC+d16	CALLP @@RW2+	CALLP @@PC+d16	INCL @RW2+	INCL @PC+d16	DECL @RW2+	DECL @PC+d16	MOVL A,@RW2+	MOVL A, @PC+d16	MOVL @RW2+,A	MOVL @PC+d16, A	MOV @RW2+, #8	MOV @PC+d16, #8	MOVEA A,@RW2+	MOVEA A, @PC+d16
+F	JMPP @@RW3+	JMPP @addr16	CALLP @@RW3+	CALLP @addr16	INCL @RW3+	INCL addr16	DECL @RW3+	DECL addr16	MOVL A,@RW3+	MOVL A, addr16	MOVL @RW3+,A	MOVL addr16, A	MOV @RW3+, #8	MOV addr16, #8	MOVEA A,@RW3+	MOVEA A, addr16

Table C-7 ea-type Instruction (3) (First Byte = 72_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	ROLC R0	ROLC @RW0+d8	RORC R0	RORC @RW0+d8	INC R0	INC @RW0+d8	DEC R0	DEC @RW0+d8	MOV A, R0	MOV A, @RW0+d8	MOV R0, A	MOV @RW0+d8,A	MOVX A, R0	MOVX A, @RW0+d8	XCH A, R0	XCH A, @RW0+d8
+1	ROLC R1	ROLC @RW1+d8	RORC R1	RORC @RW1+d8	INC R1	INC @RW1+d8	DEC R1	DEC @RW1+d8	MOV A, R1	MOV A, @RW1+d8	MOV R1, A	MOV @RW1+d8,A	MOVX A, R1	MOVX A, @RW1+d8	XCH A, R1	XCH A, @RW1+d8
+2	ROLC R2	ROLC @RW2+d8	RORC R2	RORC @RW2+d8	INC R2	INC @RW2+d8	DEC R2	DEC @RW2+d8	MOV A, R2	MOV A, @RW2+d8	MOV R2, A	MOV @RW2+d8,A	MOVX A, R2	MOVX A, @RW2+d8	XCH A, R2	XCH A, @RW2+d8
+3	ROLC R3	ROLC @RW3+d8	RORC R3	RORC @RW3+d8	INC R3	INC @RW3+d8	DEC R3	DEC @RW3+d8	MOV A, R3	MOV A, @RW3+d8	MOV R3, A	MOV @RW3+d8,A	MOVX A, R3	MOVX A, @RW3+d8	XCH A, R3	XCH A, @RW3+d8
+4	ROLC R4	ROLC @RW4+d8	RORC R4	RORC @RW4+d8	INC R4	INC @RW4+d8	DEC R4	DEC @RW4+d8	MOV A, R4	MOV A, @RW4+d8	MOV R4, A	MOV @RW4+d8,A	MOVX A, R4	MOVX A, @RW4+d8	XCH A, R4	XCH A, @RW4+d8
+5	ROLC R5	ROLC @RW5+d8	RORC R5	RORC @RW5+d8	INC R5	INC @RW5+d8	DEC R5	DEC @RW5+d8	MOV A, R5	MOV A, @RW5+d8	MOV R5, A	MOV @RW5+d8,A	MOVX A, R5	MOVX A, @RW5+d8	XCH A, R5	XCH A, @RW5+d8
+6	ROLC R6	ROLC @RW6+d8	RORC R6	RORC @RW6+d8	INC R6	INC @RW6+d8	DEC R6	DEC @RW6+d8	MOV A, R6	MOV A, @RW6+d8	MOV R6, A	MOV @RW6+d8,A	MOVX A, R6	MOVX A, @RW6+d8	XCH A, R6	XCH A, @RW6+d8
+7	ROLC R7	ROLC @RW7+d8	RORC R7	RORC @RW7+d8	INC R7	INC @RW7+d8	DEC R7	DEC @RW7+d8	MOV A, R7	MOV A, @RW7+d8	MOV R7, A	MOV @RW7+d8,A	MOVX A, R7	MOVX A, @RW7+d8	XCH A, R7	XCH A, @RW7+d8
+8	ROLC @RW0	ROLC @RW0+d16	RORC @RW0	RORC @RW0+d16	INC @RW0	INC @RW0+d16	DEC @RW0	DEC @RW0+d16	MOV A,@RW0	MOV A, @RW0+d16	MOV @RW0, A	MOV @RW0+d16,A	MOVX A,@RW0	MOVX A, @RW0+d16	XCH A,@RW0	XCH A, @RW0+d16
+9	ROLC @RW1	ROLC @RW1+d16	RORC @RW1	RORC @RW1+d16	INC @RW1	INC @RW1+d16	DEC @RW1	DEC @RW1+d16	MOV A,@RW1	MOV A, @RW1+d16	MOV @RW1, A	MOV @RW1+d16,A	MOVX A,@RW1	MOVX A, @RW1+d16	XCH A,@RW1	XCH A, @RW1+d16
+A	ROLC @RW2	ROLC @RW2+d16	RORC @RW2	RORC @RW2+d16	INC @RW2	INC @RW2+d16	DEC @RW2	DEC @RW2+d16	MOV A,@RW2	MOV A, @RW2+d16	MOV @RW2, A	MOV @RW2+d16,A	MOVX A,@RW2	MOVX A, @RW2+d16	XCH A,@RW2	XCH A, @RW2+d16
+B	ROLC @RW3	ROLC @RW3+d16	RORC @RW3	RORC @RW3+d16	INC @RW3	INC @RW3+d16	DEC @RW3	DEC @RW3+d16	MOV A,@RW3	MOV A, @RW3+d16	MOV @RW3, A	MOV @RW3+d16,A	MOVX A,@RW3	MOVX A, @RW3+d16	XCH A,@RW3	XCH A, @RW3+d16
+C	ROLC @RW0+	ROLC @RW0+RW7	RORC @RW0+	RORC @RW0+RW7	INC @RW0+	INC @RW0+RW7	DEC @RW0+	DEC @RW0+RW7	MOV A,@RW0+	MOV A, @RW0+RW7	MOV @RW0+, A	MOV @RW0+RW7,A	MOVX A,@RW0+	MOVX A, @RW0+RW7	XCH A,@RW0+	XCH A, @RW0+RW7
+D	ROLC @RW1+	ROLC @RW1+RW7	RORC @RW1+	RORC @RW1+RW7	INC @RW1+	INC @RW1+RW7	DEC @RW1+	DEC @RW1+RW7	MOV A,@RW1+	MOV A, @RW1+RW7	MOV @RW1+, A	MOV @RW1+RW7,A	MOVX A,@RW1+	MOVX A, @RW1+RW7	XCH A,@RW1+	XCH A, @RW1+RW7
+E	ROLC @RW2+	ROLC @PC+d16	RORC @RW2+	RORC @PC+d16	INC @RW2+	INC @PC+d16	DEC @RW2+	DEC @PC+d16	MOV A,@RW2+	MOV A, @PC+d16	MOV @RW2+, A	MOV @PC+d16, A	MOVX A,@RW2+	MOVX A, @PC+d16	XCH A,@RW2+	XCH A, @PC+d16
+F	ROLC @RW3+	ROLC addr16	RORC @RW3+	RORC addr16	INC @RW3+	INC addr16	DEC @RW3+	DEC addr16	MOV A,@RW3+	MOV A, addr16	MOV @RW3+, A	MOV addr16, A	MOVX A,@RW3+	MOVX A, addr16	XCH A,@RW3+	XCH A, addr16

Table C-8 ea-type Instruction (4) (First Byte = 73_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	JMP @RW0	JMP @@RW0+d8	CALL RW0	CALL @@RW0+d8	INCW RW0	INCW @RW0+d8	DECW RW0	DECW @RW0+d8	MOVW A, RW0	MOVW A, @RW0+d8	MOVW RW0, A	MOVW @RW0+d8,A	MOVW RW0, #16	MOVW @RW0+d8,#16	XCHW A, RW0	XCHW A, @RW0+d8
+1	JMP @RW1	JMP @@RW1+d8	CALL RW1	CALL @@RW1+d8	INCW RW1	INCW @RW1+d8	DECW RW1	DECW @RW1+d8	MOVW A, RW1	MOVW A, @RW1+d8	MOVW RW1, A	MOVW @RW1+d8,A	MOVW RW1, #16	MOVW @RW1+d8,#16	XCHW A, RW1	XCHW A, @RW1+d8
+2	JMP @RW2	JMP @@RW2+d8	CALL RW2	CALL @@RW2+d8	INCW RW2	INCW @RW2+d8	DECW RW2	DECW @RW2+d8	MOVW A, RW2	MOVW A, @RW2+d8	MOVW RW2, A	MOVW @RW2+d8,A	MOVW RW2, #16	MOVW @RW2+d8,#16	XCHW A, RW2	XCHW A, @RW2+d8
+3	JMP @RW3	JMP @@RW3+d8	CALL RW3	CALL @@RW3+d8	INCW RW3	INCW @RW3+d8	DECW RW3	DECW @RW3+d8	MOVW A, RW3	MOVW A, @RW3+d8	MOVW RW3, A	MOVW @RW3+d8,A	MOVW RW3, #16	MOVW @RW3+d8,#16	XCHW A, RW3	XCHW A, @RW3+d8
+4	JMP @RW4	JMP @@RW4+d8	CALL RW4	CALL @@RW4+d8	INCW RW4	INCW @RW4+d8	DECW RW4	DECW @RW4+d8	MOVW A, RW4	MOVW A, @RW4+d8	MOVW RW4, A	MOVW @RW4+d8,A	MOVW RW4, #16	MOVW @RW4+d8,#16	XCHW A, RW4	XCHW A, @RW4+d8
+5	JMP @RW5	JMP @@RW5+d8	CALL RW5	CALL @@RW5+d8	INCW RW5	INCW @RW5+d8	DECW RW5	DECW @RW5+d8	MOVW A, RW5	MOVW A, @RW5+d8	MOVW RW5, A	MOVW @RW5+d8,A	MOVW RW5, #16	MOVW @RW5+d8,#16	XCHW A, RW5	XCHW A, @RW5+d8
+6	JMP @RW6	JMP @@RW6+d8	CALL RW6	CALL @@RW6+d8	INCW RW6	INCW @RW6+d8	DECW RW6	DECW @RW6+d8	MOVW A, RW6	MOVW A, @RW6+d8	MOVW RW6, A	MOVW @RW6+d8,A	MOVW RW6, #16	MOVW @RW6+d8,#16	XCHW A, RW6	XCHW A, @RW6+d8
+7	JMP @RW7	JMP @@RW7+d8	CALL RW7	CALL @@RW7+d8	INCW RW7	INCW @RW7+d8	DECW RW7	DECW @RW7+d8	MOVW A, RW7	MOVW A, @RW7+d8	MOVW RW7, A	MOVW @RW7+d8,A	MOVW RW7, #16	MOVW @RW7+d8,#16	XCHW A, RW7	XCHW A, @RW7+d8
+8	JMP @@RW0	JMP @ @RW0+d16	CALL @@RW0	CALL @ @RW0+d16	INCW @RW0	INCW @ @RW0+d16	DECW @RW0	DECW @RW0+d16	MOVW A,@RW0	MOVW A, @RW0+d16	MOVW @RW0,A	MOVW @RW0+d16,A	MOVW @RW0, #16	MOVW @RW0+d16,#16	XCHW A,@RW0	XCHW A, @RW0+d16
+9	JMP @@RW1	JMP @ @RW1+d16	CALL @@RW1	CALL @ @RW1+d16	INCW @RW1	INCW @ @RW1+d16	DECW @RW1	DECW @RW1+d16	MOVW A,@RW1	MOVW A, @RW1+d16	MOVW @RW1, A	MOVW @RW1+d16,A	MOVW @RW1, #16	MOVW @RW1+d16,#16	XCHW A,@RW1	XCHW A, @RW1+d16
+A	JMP @@RW2	JMP @ @RW2+d16	CALL @@RW2	CALL @ @RW2+d16	INCW @RW2	INCW @ @RW2+d16	DECW @RW2	DECW @RW2+d16	MOVW A,@RW2	MOVW A, @RW2+d16	MOVW @RW2, A	MOVW @RW2+d16,A	MOVW @RW2, #16	MOVW @RW2+d16,#16	XCHW A,@RW2	XCHW A, @RW2+d16
+B	JMP @@RW3	JMP @ @RW3+d16	CALL @@RW3	CALL @ @RW3+d16	INCW @RW3	INCW @ @RW3+d16	DECW @RW3	DECW @RW3+d16	MOVW A,@RW3	MOVW A, @RW3+d16	MOVW @RW3, A	MOVW @RW3+d16,A	MOVW @RW3, #16	MOVW @RW3+d16,#16	XCHW A,@RW3	XCHW A, @RW3+d16
+C	JMP @@RW0+	JMP @ @RW0+RW7	CALL @@RW0+	CALL @ @RW0+RW7	INCW @RW0+	INCW @ @RW0+RW7	DECW @RW0+	DECW @RW0+RW7	MOVW A,@RW0+	MOVW A, @RW0+RW7	MOVW @RW0+, A	MOVW @RW0+RW7,A	MOVW @RW0+, #16	MOVW @RW0+RW7,#16	XCHW A,@RW0+	XCHW A, @RW0+RW7
+D	JMP @@RW1+	JMP @ @RW1+RW7	CALL @@RW1+	CALL @ @RW1+RW7	INCW @RW1+	INCW @ @RW1+RW7	DECW @RW1+	DECW @RW1+RW7	MOVW A,@RW1+	MOVW A, @RW1+RW7	MOVW @RW1+, A	MOVW @RW1+RW7,A	MOVW @RW1+, #16	MOVW @RW1+RW7,#16	XCHW A,@RW1+	XCHW A, @RW1+RW7
+E	JMP @@RW2+	JMP @@PC+d16	CALL @@RW2+	CALL @@PC+d16	INCW @RW2+	INCW @@PC+d16	DECW @RW2+	DECW @PC+d16	MOVW A,@RW2+	MOVW A, @PC+d16	MOVW @RW2+, A	MOVW @PC+d16, A	MOVW @RW2+, #16	MOVW @PC+d16, #16	XCHW A,@RW2+	XCHW A, @PC+d16
+F	JMP @@RW3+	JMP @addr16	CALL @@RW3+	CALL @addr16	INCW @RW3+	INCW addr16	DECW @RW3+	DECW addr16	MOVW A,@RW3+	MOVW A, addr16	MOVW @RW3+, A	MOVW addr16, A	MOVW @RW3+, #16	MOVW addr16, #16	XCHW A,@RW3+	XCHW A, addr16

Table C-9 ea-type Instruction (5) (First Byte = 74_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	ADD A, R0	ADD A, @RW0+d8	SUB A, R0	SUB A, @RW0+d8	ADDC A, R0	ADDC A, @RW0+d8	CMP A, R0	CMP A, @RW0+d8	AND A, R0	AND A, @RW0+d8	OR A, R0	OR A, @RW0+d8	XOR A, R0	XOR A, @RW0+d8	DBNZ R0, r	DBNZ @RW0+d8, r
+1	ADD A, R1	ADD A, @RW1+d8	SUB A, R1	SUB A, @RW1+d8	ADDC A, R1	ADDC A, @RW1+d8	CMP A, R1	CMP A, @RW1+d8	AND A, R1	AND A, @RW1+d8	OR A, R1	OR A, @RW1+d8	XOR A, R1	XOR A, @RW1+d8	DBNZ R1, r	DBNZ @RW1+d8, r
+2	ADD A, R2	ADD A, @RW2+d8	SUB A, R2	SUB A, @RW2+d8	ADDC A, R2	ADDC A, @RW2+d8	CMP A, R2	CMP A, @RW2+d8	AND A, R2	AND A, @RW2+d8	OR A, R2	OR A, @RW2+d8	XOR A, R2	XOR A, @RW2+d8	DBNZ R2, r	DBNZ @RW2+d8, r
+3	ADD A, R3	ADD A, @RW3+d8	SUB A, R3	SUB A, @RW3+d8	ADDC A, R3	ADDC A, @RW3+d8	CMP A, R3	CMP A, @RW3+d8	AND A, R3	AND A, @RW3+d8	OR A, R3	OR A, @RW3+d8	XOR A, R3	XOR A, @RW3+d8	DBNZ R3, r	DBNZ @RW3+d8, r
+4	ADD A, R4	ADD A, @RW4+d8	SUB A, R4	SUB A, @RW4+d8	ADDC A, R4	ADDC A, @RW4+d8	CMP A, R4	CMP A, @RW4+d8	AND A, R4	AND A, @RW4+d8	OR A, R4	OR A, @RW4+d8	XOR A, R4	XOR A, @RW4+d8	DBNZ R4, r	DBNZ @RW4+d8, r
+5	ADD A, R5	ADD A, @RW5+d8	SUB A, R5	SUB A, @RW5+d8	ADDC A, R5	ADDC A, @RW5+d8	CMP A, R5	CMP A, @RW5+d8	AND A, R5	AND A, @RW5+d8	OR A, R5	OR A, @RW5+d8	XOR A, R5	XOR A, @RW5+d8	DBNZ R5, r	DBNZ @RW5+d8, r
+6	ADD A, R6	ADD A, @RW6+d8	SUB A, R6	SUB A, @RW6+d8	ADDC A, R6	ADDC A, @RW6+d8	CMP A, R6	CMP A, @RW6+d8	AND A, R6	AND A, @RW6+d8	OR A, R6	OR A, @RW6+d8	XOR A, R6	XOR A, @RW6+d8	DBNZ R6, r	DBNZ @RW6+d8, r
+7	ADD A, R7	ADD A, @RW7+d8	SUB A, R7	SUB A, @RW7+d8	ADDC A, R7	ADDC A, @RW7+d8	CMP A, R7	CMP A, @RW7+d8	AND A, R7	AND A, @RW7+d8	OR A, R7	OR A, @RW7+d8	XOR A, R7	XOR A, @RW7+d8	DBNZ R7, r	DBNZ @RW7+d8, r
+8	ADD A,@RW0	ADD A, @RW0+d16	SUB A,@RW0	SUB A, @RW0+d16	ADDC A,@RW0	ADDC A, @RW0+d16	CMP A,@RW0	CMP A, @RW0+d16	AND A,@RW0	AND A, @RW0+d16	OR A,@RW0	OR A, @RW0+d16	XOR A,@RW0	XOR A, @RW0+d16	DBNZ @RW0, r	DBNZ @RW0+d16, r
+9	ADD A,@RW1	ADD A, @RW1+d16	SUB A,@RW1	SUB A, @RW1+d16	ADDC A,@RW1	ADDC A, @RW1+d16	CMP A,@RW1	CMP A, @RW1+d16	AND A,@RW1	AND A, @RW1+d16	OR A,@RW1	OR A, @RW1+d16	XOR A,@RW1	XOR A, @RW1+d16	DBNZ @RW1, r	DBNZ @RW1+d16, r
+A	ADD A,@RW2	ADD A, @RW2+d16	SUB A,@RW2	SUB A, @RW2+d16	ADDC A,@RW2	ADDC A, @RW2+d16	CMP A,@RW2	CMP A, @RW2+d16	AND A,@RW2	AND A, @RW2+d16	OR A,@RW2	OR A, @RW2+d16	XOR A,@RW2	XOR A, @RW2+d16	DBNZ @RW2, r	DBNZ @RW2+d16, r
+B	ADD A,@RW3	ADD A, @RW3+d16	SUB A,@RW3	SUB A, @RW3+d16	ADDC A,@RW3	ADDC A, @RW3+d16	CMP A,@RW3	CMP A, @RW3+d16	AND A,@RW3	AND A, @RW3+d16	OR A,@RW3	OR A, @RW3+d16	XOR A,@RW3	XOR A, @RW3+d16	DBNZ @RW3, r	DBNZ @RW3+d16, r
+C	ADD A,@RW0+	ADD A, @RW0+RW7	SUB A,@RW0+	SUB A, @RW0+RW7	ADDC A,@RW0+	ADDC A, @RW0+RW7	CMP A,@RW0+	CMP A, @RW0+RW7	AND A,@RW0+	AND A, @RW0+RW7	OR A,@RW0+	OR A, @RW0+RW7	XOR A,@RW0+	XOR A, @RW0+RW7	DBNZ @RW0+, r	DBNZ @RW0+RW7, r
+D	ADD A,@RW1+	ADD A, @RW1+RW7	SUB A,@RW1+	SUB A, @RW1+RW7	ADDC A,@RW1+	ADDC A, @RW1+RW7	CMP A,@RW1+	CMP A, @RW1+RW7	AND A,@RW1+	AND A, @RW1+RW7	OR A,@RW1+	OR A, @RW1+RW7	XOR A,@RW1+	XOR A, @RW1+RW7	DBNZ @RW1+, r	DBNZ @RW1+RW7, r
+E	ADD A,@RW2+	ADD A, @PC+d16	SUB A,@RW2+	SUB A, @PC+d16	ADDC A,@RW2+	ADDC A, @PC+d16	CMP A,@RW2+	CMP A, @PC+d16	AND A,@RW2+	AND A, @PC+d16	OR A,@RW2+	OR A, @PC+d16,	XOR A,@RW2+	XOR A, @PC+d16	DBNZ @RW2+, r	DBNZ @PC+d16, r
+F	ADD A,@RW3+	ADD A, addr16	SUB A,@RW3+	SUB A, addr16	ADDC A,@RW3+	ADDC A, addr16	CMP A,@RW3+	CMP A, addr16	AND A,@RW3+	AND A, addr16	OR A,@RW3+	OR A, addr16	XOR A,@RW3+	XOR A, addr16	DBNZ @RW3+, r	DBNZ addr16, r

Table C-10 ea-type Instruction (6) (First Byte = 75_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	ADD R0, A	ADD @RW0+d8, A	SUB R0, A	SUB @RW0+d8, A	SUBC A, R0	SUBC A, @RW0+d8	NEG R0	NEG A, @RW0+d8	AND R0, A	AND @RW0+d8, A	OR R0, A	OR @RW0+d8, A	XOR R0, A	XOR @RW0+d8, A	NOT R0	NOT @RW0+d8
+1	ADD R1, A	ADD @RW1+d8, A	SUB R1, A	SUB @RW1+d8, A	SUBC A, R1	SUBC A, @RW1+d8	NEG R1	NEG A, @RW1+d8	AND R1, A	AND @RW1+d8, A	OR R1, A	OR @RW1+d8, A	XOR R1, A	XOR @RW1+d8, A	NOT R1	NOT @RW1+d8
+2	ADD R2, A	ADD @RW2+d8,A	SUB R2, A	SUB @RW2+d8,A	SUBC A, R2	SUBC A, @RW2+d8	NEG R2	NEG A, @RW2+d8	AND R2, A	AND @RW2+d8,A	OR R2, A	OR @RW2+d8,A	XOR R2, A	XOR @RW2+d8,A	NOT R2	NOT @RW2+d8
+3	ADD R3, A	ADD @RW3+d8, A	SUB R3, A	SUB @RW3+d8, A	SUBC A, R3	SUBC A, @RW3+d8	NEG R3	NEG A, @RW3+d8	AND R3, A	AND @RW3+d8, A	OR R3, A	OR @RW3+d8, A	XOR R3, A	XOR @RW3+d8, A	NOT R3	NOT @RW3+d8
+4	ADD R4, A	ADD @RW4+d8, A	SUB R4, A	SUB @RW4+d8, A	SUBC A, R4	SUBC A, @RW4+d8	NEG R4	NEG A, @RW4+d8	AND R4, A	AND @RW4+d8, A	OR R4, A	OR @RW4+d8, A	XOR R4, A	XOR @RW4+d8, A	NOT R4	NOT @RW4+d8
+5	ADD R5, A	ADD @RW5+d8, A	SUB R5, A	SUB @RW5+d8, A	SUBC A, R5	SUBC A, @RW5+d8	NEG R5	NEG A, @RW5+d8	AND R5, A	AND @RW5+d8, A	OR R5, A	OR @RW5+d8, A	XOR R5, A	XOR @RW5+d8, A	NOT R5	NOT @RW5+d8
+6	ADD R6, A	ADD @RW6+d8, A	SUB R6, A	SUB @RW6+d8, A	SUBC A, R6	SUBC A, @RW6+d8	NEG R6	NEG A, @RW6+d8	AND R6, A	AND @RW6+d8, A	OR R6, A	OR @RW6+d8, A	XOR R6, A	XOR @RW6+d8, A	NOT R6	NOT @RW6+d8
+7	ADD R7, A	ADD @RW7+d8, A	SUB R7, A	SUB @RW7+d8, A	SUBC A, R7	SUBC A, @RW7+d8	NEG R7	NEG A, @RW7+d8	AND R7, A	AND @RW7+d8, A	OR R7, A	OR @RW7+d8, A	XOR R7, A	XOR @RW7+d8, A	NOT R7	NOT @RW7+d8
+8	ADD @RW0, A	ADD @RW0+d16,A	SUB @RW0, A	SUB @RW0+d16,A	SUBC A, @RW0	SUBC A, @RW0+d16	NEG @RW0	NEG A, @RW0+d16	AND @RW0, A	AND @RW0+d16,A	OR @RW0, A	OR @RW0+d16,A	XOR @RW0, A	XOR @RW0+d16,A	NOT @RW0	NOT @RW0+d16
+9	ADD @RW1, A	ADD @R @RW1+d16,A	SUB @RW1, A	SUB @RW1+d16,A	SUBC A, @RW1	SUBC A, @RW1+d16	NEG @RW1	NEG A, @RW1+d16	AND @RW1, A	AND @RW1+d16,A	OR @RW1, A	OR @RW1+d16,A	XOR @RW1, A	XOR @RW1+d16,A	NOT @RW1	NOT @RW1+d16
+A	ADD @RW2, A	ADD @R @RW2+d16,A	SUB @RW2, A	SUB @RW2+d16,A	SUBC A, @RW2	SUBC A, @RW2+d16	NEG @RW2	NEG A, @RW2+d16	AND @RW2, A	AND @RW2+d16,A	OR @RW2, A	OR @RW2+d16,A	XOR @RW2, A	XOR @RW2+d16,A	NOT @RW2	NOT @RW2+d16
+B	ADD @RW3, A	ADD @R @RW3+d16,A	SUB @RW3, A	SUB @RW3+d16,A	SUBC A, @RW3	SUBC A, @RW3+d16	NEG @RW3	NEG A, @RW3+d16	AND @RW3, A	AND @RW3+d16,A	OR @RW3, A	OR @RW3+d16,A	XOR @RW3, A	XOR @RW3+d16,A	NOT @RW3	NOT @RW3+d16
+C	ADD @RW0+, A	ADD @R @RW0+RW7,A	SUB @RW0+, A	SUB @RW0+RW7,A	SUBC A,@RW0+	SUBC A, @RW0+RW7	NEG @RW0+	NEG A, @RW0+RW7	AND @RW0+, A	AND @RW0+RW7,A	OR @RW0+, A	OR @RW0+RW7,A	XOR @RW0+, A	XOR @RW0+RW7,A	NOT @RW0+	NOT @RW0+RW7
+D	ADD @RW1+, A	ADD @R @RW1+RW7,A	SUB @RW1+, A	SUB @RW1+RW7,A	SUBC A,@RW1+	SUBC A, @RW1+RW7	NEG @RW1+	NEG A, @RW1+RW7	AND @RW1+, A	AND @RW1+RW7,A	OR @RW1+, A	OR @RW1+RW7,A	XOR @RW1+, A	XOR @RW1+RW7,A	NOT @RW1+	NOT @RW1+RW7
+E	ADD @RW2+, A	ADD @PC+d16,A	SUB @RW2+, A	SUB @PC+d16,A	SUBC A,@RW2+	SUBC A, @PC+d16	NEG @RW2+	NEG A, @PC+d16	AND @RW2+, A	AND @PC+d16,A	OR @RW2+, A	OR @PC+d16,A	XOR @RW2+, A	XOR @PC+d16,A	NOT @RW2+	NOT @PC+d16
+F	ADD @RW3+, A	ADD addr16, A	SUB @RW3+, A	SUB addr16, A	SUBC A,@RW3+	SUBC A, addr16	NEG @RW3+	NEG A, addr16	AND @RW3+, A	AND addr16, A	OR @RW3+, A	OR addr16, A	XOR @RW3+, A	XOR addr16, A	NOT @RW3+	NOT addr16

Table C-11 ea-type Instruction (7) (First Byte = 76_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	ADDW A, RW0	ADDW A, @RW0+d8	SUBW A, RW0	SUBW A, @RW0+d8	ADDCW A, RW0	ADDCW A, @RW0+d8	CMPW A, RW0	CMPW A, @RW0+d8	ANDW A, RW0	ANDW A, @RW0+d8	ORW A, RW0	ORW A, @RW0+d8	XORW A, RW0	XORW A, @RW0+d8	DWBNZ RW0, r	DWBNZ @RW0+d8,r
+1	ADDW A, RW1	ADDW A, @RW1+d8	SUBW A, RW1	SUBW A, @RW1+d8	ADDCW A, RW1	ADDCW A, @RW1+d8	CMPW A, RW1	CMPW A, @RW1+d8	ANDW A, RW1	ANDW A, @RW1+d8	ORW A, RW1	ORW A, @RW1+d8	XORW A, RW1	XORW A, @RW1+d8	DWBNZ RW1, r	DWBNZ @RW1+d8,r
+2	ADDW A, RW2	ADDW A, @RW2+d8	SUBW A, RW2	SUBW A, @RW2+d8	ADDCW A, RW2	ADDCW A, @RW2+d8	CMPW A, RW2	CMPW A, @RW2+d8	ANDW A, RW2	ANDW A, @RW2+d8	ORW A, RW2	ORW A, @RW2+d8	XORW A, RW2	XORW A, @RW2+d8	DWBNZ RW2, r	DWBNZ @RW2+d8,r
+3	ADDW A, RW3	ADDW A, @RW3+d8	SUBW A, RW3	SUBW A, @RW3+d8	ADDCW A, RW3	ADDCW A, @RW3+d8	CMPW A, RW3	CMPW A, @RW3+d8	ANDW A, RW3	ANDW A, @RW3+d8	ORW A, RW3	ORW A, @RW3+d8	XORW A, RW3	XORW A, @RW3+d8	DWBNZ RW3, r	DWBNZ @RW3+d8,r
+4	ADDW A, RW4	ADDW A, @RW4+d8	SUBW A, RW4	SUBW A, @RW4+d8	ADDCW A, RW4	ADDCW A, @RW4+d8	CMPW A, RW4	CMPW A, @RW4+d8	ANDW A, RW4	ANDW A, @RW4+d8	ORW A, RW4	ORW A, @RW4+d8	XORW A, RW4	XORW A, @RW4+d8	DWBNZ RW4, r	DWBNZ @RW4+d8,r
+5	ADDW A, RW5	ADDW A, @RW5+d8	SUBW A, RW5	SUBW A, @RW5+d8	ADDCW A, RW5	ADDCW A, @RW5+d8	CMPW A, RW5	CMPW A, @RW5+d8	ANDW A, RW5	ANDW A, @RW5+d8	ORW A, RW5	ORW A, @RW5+d8	XORW A, RW5	XORW A, @RW5+d8	DWBNZ RW5, r	DWBNZ @RW5+d8,r
+6	ADDW A, RW6	ADDW A, @RW6+d8	SUBW A, RW6	SUBW A, @RW6+d8	ADDCW A, RW6	ADDCW A, @RW6+d8	CMPW A, RW6	CMPW A, @RW6+d8	ANDW A, RW6	ANDW A, @RW6+d8	ORW A, RW6	ORW A, @RW6+d8	XORW A, RW6	XORW A, @RW6+d8	DWBNZ RW6, r	DWBNZ @RW6+d8,r
+7	ADDW A, RW7	ADDW A, @RW7+d8	SUBW A, RW7	SUBW A, @RW7+d8	ADDCW A, RW7	ADDCW A, @RW7+d8	CMPW A, RW7	CMPW A, @RW7+d8	ANDW A, RW7	ANDW A, @RW7+d8	ORW A, RW7	ORW A, @RW7+d8	XORW A, RW7	XORW A, @RW7+d8	DWBNZ RW7, r	DWBNZ @RW7+d8,r
+8	ADDW A,@RW0	ADDW A, @RW0+d16	SUBW A,@RW0	SUBW A, @RW0+d16	ADDCW A,@RW0	ADDCW A, @RW0+d16	CMPW A,@RW0	CMPW A, @RW0+d16	ANDW A,@RW0	ANDW A, @RW0+d16	ORW A,@RW0	ORW A, @RW0+d16	XORW A,@RW0	XORW A, @RW0+d16	DWBNZ @RW0, r	DWBNZ @RW0+d16,r
+9	ADDW A,@RW1	ADDW A, @RW1+d16	SUBW A,@RW1	SUBW A, @RW1+d16	ADDCW A,@RW1	ADDCW A, @RW1+d16	CMPW A,@RW1	CMPW A, @RW1+d16	ANDW A,@RW1	ANDW A, @RW1+d16	ORW A,@RW1	ORW A, @RW1+d16	XORW A,@RW1	XORW A, @RW1+d16	DWBNZ @RW1, r	DWBNZ @RW1+d16,r
+A	ADDW A,@RW2	ADDW A, @RW2+d16	SUBW A,@RW2	SUBW A, @RW2+d16	ADDCW A,@RW2	ADDCW A, @RW2+d16	CMPW A,@RW2	CMPW A, @RW2+d16	ANDW A,@RW2	ANDW A, @RW2+d16	ORW A,@RW2	ORW A, @RW2+d16	XORW A,@RW2	XORW A, @RW2+d16	DWBNZ @RW2, r	DWBNZ @RW2+d16,r
+B	ADDW A,@RW3	ADDW A, @RW3+d16	SUBW A,@RW3	SUBW A, @RW3+d16	ADDCW A,@RW3	ADDCW A, @RW3+d16	CMPW A,@RW3	CMPW A, @RW3+d16	ANDW A,@RW3	ANDW A, @RW3+d16	ORW A,@RW3	ORW A, @RW3+d16	XORW A,@RW3	XORW A, @RW3+d16	DWBNZ @RW3, r	DWBNZ @RW3+d16,r
+C	ADDW A,@RW0+	ADDW A, @RW0+RW7	SUBW A,@RW0+	SUBW A, @RW0+RW7	ADDCW A,@RW0+	ADDCW A, @RW0+RW7	CMPW A,@RW0+	CMPW A, @RW0+RW7	ANDW A,@RW0+	ANDW A, @RW0+RW7	ORW A,@RW0+	ORW A, @RW0+RW7	XORW A,@RW0+	XORW A, @RW0+RW7	DWBNZ @RW0+, r	DWBNZ @RW0+RW7,r
+D	ADDW A,@RW1+	ADDW A, @RW1+RW7	SUBW A,@RW1+	SUBW A, @RW1+RW7	ADDCW A,@RW1+	ADDCW A, @RW1+RW7	CMPW A,@RW1+	CMPW A, @RW1+RW7	ANDW A,@RW1+	ANDW A, @RW1+RW7	ORW A,@RW1+	ORW A, @RW1+RW7	XORW A,@RW1+	XORW A, @RW1+RW7	DWBNZ @RW1+, r	DWBNZ @RW1+RW7,r
+E	ADDW A,@RW2+	ADDW A, @PC+d16	SUBW A,@RW2+	SUBW A, @PC+d16	ADDCW A,@RW2+	ADDCW A, @PC+d16	CMPW A,@RW2+	CMPW A, @PC+d16	ANDW A,@RW2+	ANDW A, @PC+d16	ORW A,@RW2+	ORW A, @PC+d16,	XORW A,@RW2+	XORW A, @PC+d16	DWBNZ @RW2+, r	DWBNZ @PC+d16,r
+F	ADDW A,@RW3+	ADDW A, addr 16	SUBW A,@RW3+	SUBW A, addr 16	ADDCW A,@RW3+	ADDCW A, addr 16	CMPW A,@RW3+	CMPW A, addr 16	ANDW A,@RW3+	ANDW A, addr 16	ORW A,@RW3+	ORW A, addr r16	XORW A,@RW3+	XORW A, addr 16	DWBNZ @RW3+, r	DWBNZ addr r16, r

Table C-12 ea-type Instruction (8) (First Byte = 77_H)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	ADDW RW0, A	ADDW @RW0+d8, A	SUBW RW0, A	SUBW @RW0+d8, A	SUBCW A, RW0	SUBCW A, @RW0+d8	NEGW RW0	NEGW @RW0+d8	ANDW RW0, A	ANDW @RW0+d8, A	ORW RW0, A	ORW @RW0+d8, A	XORW RW0, A	XORW @RW0+d8, A	NOTW RW0	NOTW @RW0+d8
+1	ADDW RW1, A	ADDW @RW1+d8, A	SUBW RW1, A	SUBW @RW1+d8, A	SUBCW A, RW1	SUBCW A, @RW1+d8	NEGW RW1	NEGW @RW1+d8	ANDW RW1, A	ANDW @RW1+d8, A	ORW RW1, A	ORW @RW1+d8, A	XORW RW1, A	XORW @RW1+d8, A	NOTW RW1	NOTW @RW1+d8
+2	ADDW RW2, A	ADDW @RW2+d8,A	SUBW RW2, A	SUBW @RW2+d8,A	SUBCW A, RW2	SUBCW A, @RW2+d8	NEGW RW2	NEGW @RW2+d8	ANDW RW2, A	ANDW @RW2+d8,A	ORW RW2, A	ORW @RW2+d8,A	XORW RW2, A	XORW @RW2+d8,A	NOTW RW2	NOTW @RW2+d8
+3	ADDW RW3, A	ADDW @RW3+d8, A	SUBW RW3, A	SUBW @RW3+d8, A	SUBCW A, RW3	SUBCW A, @RW3+d8	NEGW RW3	NEGW @RW3+d8	ANDW RW3, A	ANDW @RW3+d8, A	ORW RW3, A	ORW @RW3+d8, A	XORW RW3, A	XORW @RW3+d8, A	NOTW RW3	NOTW @RW3+d8
+4	ADDW RW4, A	ADDW @RW4+d8, A	SUBW RW4, A	SUBW @RW4+d8, A	SUBCW A, RW4	SUBCW A, @RW4+d8	NEGW RW4	NEGW @RW4+d8	ANDW RW4, A	ANDW @RW4+d8, A	ORW RW4, A	ORW @RW4+d8, A	XORW RW4, A	XORW @RW4+d8, A	NOTW RW4	NOTW @RW4+d8
+5	ADDW RW5, A	ADDW @RW5+d8, A	SUBW RW5, A	SUBW @RW5+d8, A	SUBCW A, RW5	SUBCW A, @RW5+d8	NEGW RW5	NEGW @RW5+d8	ANDW RW5, A	ANDW @RW5+d8, A	ORW RW5, A	ORW @RW5+d8, A	XORW RW5, A	XORW @RW5+d8, A	NOTW RW5	NOTW @RW5+d8
+6	ADDW RW6, A	ADDW @RW6+d8, A	SUBW RW6, A	SUBW @RW6+d8, A	SUBCW A, RW6	SUBCW A, @RW6+d8	NEGW RW6	NEGW @RW6+d8	ANDW RW6, A	ANDW @RW6+d8, A	ORW RW6, A	ORW @RW6+d8, A	XORW RW6, A	XORW @RW6+d8, A	NOTW RW6	NOTW @RW6+d8
+7	ADDW RW7, A	ADDW @RW7+d8, A	SUBW RW7, A	SUBW @RW7+d8, A	SUBCW A, RW7	SUBCW A, @RW7+d8	NEGW RW7	NEGW @RW7+d8	ANDW RW7, A	ANDW @RW7+d8, A	ORW RW7, A	ORW @RW7+d8, A	XORW RW7, A	XORW @RW7+d8, A	NOTW RW7	NOTW @RW7+d8
+8	ADDW @RW0, A	ADDW @RW0+d16,A	SUBW @RW0, A	SUBW @RW0+d16,A	SUBCW A, @RW0	SUBCW A, @RW0+d16	NEGW @RW0	NEGW @RW0+d16	ANDW @RW0, A	ANDW @RW0+d16,A	ORW @RW0, A	ORW @RW0+d16,A	XORW @RW0, A	XORW @RW0+d16,A	NOTW @RW0	NOTW @RW0+d16
+9	ADDW @RW1, A	ADDW @RW1+d16,A	SUBW @RW1, A	SUBW @RW1+d16,A	SUBCW A, @RW1	SUBCW A, @RW1+d16	NEGW @RW1	NEGW @RW1+d16	ANDW @RW1, A	ANDW @RW1+d16,A	ORW @RW1, A	ORW @RW1+d16,A	XORW @RW1, A	XORW @RW1+d16,A	NOTW @RW1	NOTW @RW1+d16
+A	ADDW @RW2, A	ADDW @RW2+d16,A	SUBW @RW2, A	SUBW @RW2+d16,A	SUBCW A, @RW2	SUBCW A, @RW2+d16	NEGW @RW2	NEGW @RW2+d16	ANDW @RW2, A	ANDW @RW2+d16,A	ORW @RW2, A	ORW @RW2+d16,A	XORW @RW2, A	XORW @RW2+d16,A	NOTW @RW2	NOTW @RW2+d16
+B	ADDW @RW3, A	ADDW @RW3+d16,A	SUBW @RW3, A	SUBW @RW3+d16,A	SUBCW A, @RW3	SUBCW A, @RW3+d16	NEGW @RW3	NEGW @RW3+d16	ANDW @RW3, A	ANDW @RW3+d16,A	ORW @RW3, A	ORW @RW3+d16,A	XORW @RW3, A	XORW @RW3+d16,A	NOTW @RW3	NOTW @RW3+d16
+C	ADDW @RW0+, A	ADDW @RW0+RW7,A	SUBW @RW0+, A	SUBW @RW0+RW7,A	SUBCW A,@RW0+	SUBCW A, @RW0+RW7	NEGW @RW0+	NEGW @RW0+RW7	ANDW @RW0+, A	ANDW @RW0+RW7,A	ORW @RW0+, A	ORW @RW0+RW7,A	XORW @RW0+, A	XORW @RW0+RW7,A	NOTW @RW0+	NOTW @RW0+RW7
+D	ADDW @RW1+, A	ADDW @RW1+RW7,A	SUBW @RW1+, A	SUBW @RW1+RW7,A	SUBCW A,@RW1+	SUBCW A, @RW1+RW7	NEGW @RW1+	NEGW @RW1+RW7	ANDW @RW1+, A	ANDW @RW1+RW7,A	ORW @RW1+, A	ORW @RW1+RW7,A	XORW @RW1+, A	XORW @RW1+RW7,A	NOTW @RW1+	NOTW @RW1+RW7
+E	ADDW @RW2+, A	ADDW @PC+d16,A	SUBW @RW2+, A	SUBW @PC+d16,A	SUBCW A,@RW2+	SUBCW A, @PC+d16	NEGW @RW2+	NEGW @PC+d16	ANDW @RW2+, A	ANDW @PC+d16,A	ORW @RW2+, A	ORW @PC+d16,A	XORW @RW2+, A	XORW @PC+d16,A	NOTW @RW2+	NOTW @PC+d16
+F	ADDW @RW3+, A	ADDW addr16, A	SUBW @RW3+, A	SUBW addr16, A	SUBCW A,@RW3+	SUBCW A, addr16	NEGW @RW3+	NEGW addr16	ANDW @RW3+, A	ANDW addr16, A	ORW @RW3+, A	ORW addr16, A	XORW @RW3+, A	XORW addr16, A	NOTW @RW3+	NOTW addr16

Table C-13 ea-type Instruction (9) (First Byte = 78$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MULU A, R0	MULU A, @RW0+d8	MULUW A, RW0	MULUW A, @RW0+d8	MUL A, R0	MUL A, @RW0+d8	MULW A, RW0	MULW A, @RW0+d8	DIVU A, R0	DIVU A, @RW0+d8	DIVUW A, RW0	DIVUW A, @RW0+d8	DIV A, R0	DIV A, @RW0+d8	DIVW A, RW0	DIVW A, @RW0+d8
+1	MULU A, R1	MULU A, @RW1+d8	MULUW A, RW1	MULUW A, @RW1+d8	MUL A, R1	MUL A, @RW1+d8	MULW A, RW1	MULW A, @RW1+d8	DIVU A, R1	DIVU A, @RW1+d8	DIVUW A, RW1	DIVUW A, @RW1+d8	DIV A, R1	DIV A, @RW1+d8	DIVW A, RW1	DIVW A, @RW1+d8
+2	MULU A, R2	MULU A, @RW2+d8	MULUW A, RW2	MULUW A, @RW2+d8	MUL A, R2	MUL A, @RW2+d8	MULW A, RW2	MULW A, @RW2+d8	DIVU A, R2	DIVU A, @RW2+d8	DIVUW A, RW2	DIVUW A, @RW2+d8	DIV A, R2	DIV A, @RW2+d8	DIVW A, RW2	DIVW A, @RW2+d8
+3	MULU A, R3	MULU A, @RW3+d8	MULUW A, RW3	MULUW A, @RW3+d8	MUL A, R3	MUL A, @RW3+d8	MULW A, RW3	MULW A, @RW3+d8	DIVU A, R3	DIVU A, @RW3+d8	DIVUW A, RW3	DIVUW A, @RW3+d8	DIV A, R3	DIV A, @RW3+d8	DIVW A, RW3	DIVW A, @RW3+d8
+4	MULU A, R4	MULU A, @RW4+d8	MULUW A, RW4	MULUW A, @RW4+d8	MUL A, R4	MUL A, @RW4+d8	MULW A, RW4	MULW A, @RW4+d8	DIVU A, R4	DIVU A, @RW4+d8	DIVUW A, RW4	DIVUW A, @RW4+d8	DIV A, R4	DIV A, @RW4+d8	DIVW A, RW4	DIVW A, @RW4+d8
+5	MULU A, R5	MULU A, @RW5+d8	MULUW A, RW5	MULUW A, @RW5+d8	MUL A, R5	MUL A, @RW5+d8	MULW A, RW5	MULW A, @RW5+d8	DIVU A, R5	DIVU A, @RW5+d8	DIVUW A, RW5	DIVUW A, @RW5+d8	DIV A, R5	DIV A, @RW5+d8	DIVW A, RW5	DIVW A, @RW5+d8
+6	MULU A, R6	MULU A, @RW6+d8	MULUW A, RW6	MULUW A, @RW6+d8	MUL A, R6	MUL A, @RW6+d8	MULW A, RW6	MULW A, @RW6+d8	DIVU A, R6	DIVU A, @RW6+d8	DIVUW A, RW6	DIVUW A, @RW6+d8	DIV A, R6	DIV A, @RW6+d8	DIVW A, RW6	DIVW A, @RW6+d8
+7	MULU A, R7	MULU A, @RW7+d8	MULUW A, RW7	MULUW A, @RW7+d8	MUL A, R7	MUL A, @RW7+d8	MULW A, RW7	MULW A, @RW7+d8	DIVU A, R7	DIVU A, @RW7+d8	DIVUW A, RW7	DIVUW A, @RW7+d8	DIV A, R7	DIV A, @RW7+d8	DIVW A, RW7	DIVW A, @RW7+d8
+8	MULU A,@RW0	MULU A, @RW0+d16	MULUW A,@RW0	MULUW A, @RW0+d16	MUL A, @RW0	MUL A, @RW0+d16	MULW A,@RW0	MULW A, @RW0+d16	DIVU A,@RW0	DIVU A, @RW0+d16	DIVUW A,@RW0	DIVUW A, @RW0+d16	DIV A,@RW0	DIV A, @RW0+d16	DIVW A,@RW0	DIVW A, @RW0+d16
+9	MULU A,@RW1	MULU A, @RW1+d16	MULUW A,@RW1	MULUW A, @RW1+d16	MUL A, @RW1	MUL A, @RW1+d16	MULW A,@RW1	MULW A, @RW1+d16	DIVU A,@RW1	DIVU A, @RW1+d16	DIVUW A,@RW1	DIVUW A, @RW1+d16	DIV A,@RW1	DIV A, @RW1+d16	DIVW A,@RW1	DIVW A, @RW1+d16
+A	MULU A,@RW2	MULU A, @RW2+d16	MULUW A,@RW2	MULUW A, @RW2+d16	MUL A, @RW2	MUL A, @RW2+d16	MULW A,@RW2	MULW A, @RW2+d16	DIVU A,@RW2	DIVU A, @RW2+d16	DIVUW A,@RW2	DIVUW A, @RW2+d16	DIV A,@RW2	DIV A, @RW2+d16	DIVW A,@RW2	DIVW A, @RW2+d16
+B	MULU A,@RW3	MULU A, @RW3+d16	MULUW A,@RW3	MULUW A, @RW3+d16	MUL A, @RW3	MUL A, @RW3+d16	MULW A,@RW3	MULW A, @RW3+d16	DIVU A,@RW3	DIVU A, @RW3+d16	DIVUW A,@RW3	DIVUW A, @RW3+d16	DIV A,@RW3	DIV A, @RW3+d16	DIVW A,@RW3	DIVW A, @RW3+d16
+C	MULU A,@RW0+	MULU A, @RW0+RW7	MULUW A,@RW0+	MULUW A, @RW0+RW7	MUL A,@RW0+	MUL A, @RW0+RW7	MULW A,@RW0+	MULW A, @RW0+RW7	DIVU A,@RW0+	DIVU A, @RW0+RW7	DIVUW A,@RW0+	DIVUW A, @RW0+RW7	DIV A,@RW0+	DIV A, @RW0+RW7	DIVW A,@RW0+	DIVW A, @RW0+RW7
+D	MULU A,@RW1+	MULU A, @RW1+RW7	MULUW A,@RW1+	MULUW A, @RW1+RW7	MUL A,@RW1+	MUL A, @RW1+RW7	MULW A,@RW1+	MULW A, @RW1+RW7	DIVU A,@RW1+	DIVU A, @RW1+RW7	DIVUW A,@RW1+	DIVUW A, @RW1+RW7	DIV A,@RW1+	DIV A, @RW1+RW7	DIVW A,@RW1+	DIVW A, @RW1+RW7
+E	MULU A,@RW2+	MULU A, @PC+d16	MULUW A,@RW2+	MULUW A, @PC+d16	MUL A,@RW2+	MUL A, @PC+d16	MULW A,@RW2+	MULW A, @PC+d16	DIVU A,@RW2+	DIVU A, @PC+d16	DIVUW A,@RW2+	DIVUW A, @PC+d16	DIV A,@RW2+	DIV A, @PC+d16	DIVW A,@RW2+	DIVW A, @PC+d16
+F	MULU A, @RW3+	MULU A, addr16	MULUW A,@RW3+	MULUW A, addr16	MUL A,@RW3+	MUL A, addr16	MULW A,@RW3+	MULW A, addr16	DIVU A,@RW3+	DIVU A, addr16	DIVUW A,@RW3+	DIVUW A, addr16	DIV A,@RW3+	DIV A, addr16	DIVW A,@RW3+	DIVW A, addr16

C.7 MOVEA RWi, ea Instruction Map

Table C-14 lists MOVEA RWi, ea instruction map.

■ MOVEA RWi, ea Instruction Map

Table C-14 MOVEA RWi, ea Instruction (First Byte = 79$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOVEA RW0,RW0	MOVEA RW0 ,@RW0+d8	MOVEA RW1,RW0	MOVEA RW1 ,@RW0+d8	MOVEA RW2,RW0	MOVEA RW2 ,@RW0+d8	MOVEA RW3,RW0	MOVEA RW3 ,@RW0+d8	MOVEA RW4,RW0	MOVEA RW4 ,@RW0+d8	MOVEA RW5,RW0	MOVEA RW5 ,@RW0+d8	MOVEA RW6,RW0	MOVEA RW6 ,@RW0+d8	MOVEA RW7,RW0	MOVEA RW7 ,@RW0+d8
+1	MOVEA RW0,RW1	MOVEA RW0 ,@RW1+d8	MOVEA RW1,RW1	MOVEA RW1 ,@RW1+d8	MOVEA RW2,RW1	MOVEA RW2 ,@RW1+d8	MOVEA RW3,RW1	MOVEA RW3 ,@RW1+d8	MOVEA RW4,RW1	MOVEA RW4 ,@RW1+d8	MOVEA RW5,RW1	MOVEA RW5 ,@RW1+d8	MOVEA RW6,RW1	MOVEA RW6 ,@RW1+d8	MOVEA RW7,RW1	MOVEA RW7 ,@RW1+d8
+2	MOVEA RW0,RW2	MOVEA RW0 ,@RW2+d8	MOVEA RW1,RW2	MOVEA RW1 ,@RW2+d8	MOVEA RW2,RW2	MOVEA RW2 ,@RW2+d8	MOVEA RW3,RW2	MOVEA RW3 ,@RW2+d8	MOVEA RW4,RW2	MOVEA RW4 ,@RW2+d8	MOVEA RW5,RW2	MOVEA RW5 ,@RW2+d8	MOVEA RW6,RW2	MOVEA RW6 ,@RW2+d8	MOVEA RW7,RW2	MOVEA RW7 ,@RW2+d8
+3	MOVEA RW0,RW3	MOVEA RW0 ,@RW3+d8	MOVEA RW1,RW3	MOVEA RW1 ,@RW3+d8	MOVEA RW2,RW3	MOVEA RW2 ,@RW3+d8	MOVEA RW3,RW3	MOVEA RW3 ,@RW3+d8	MOVEA RW4,RW3	MOVEA RW4 ,@RW3+d8	MOVEA RW5,RW3	MOVEA RW5 ,@RW3+d8	MOVEA RW6,RW3	MOVEA RW6 ,@RW3+d8	MOVEA RW7,RW3	MOVEA RW7 ,@RW3+d8
+4	MOVEA RW0,RW4	MOVEA RW0 ,@RW4+d8	MOVEA RW1,RW4	MOVEA RW1 ,@RW4+d8	MOVEA RW2,RW4	MOVEA RW2 ,@RW4+d8	MOVEA RW3,RW4	MOVEA RW3 ,@RW4+d8	MOVEA RW4,RW4	MOVEA RW4 ,@RW4+d8	MOVEA RW5,RW4	MOVEA RW5 ,@RW4+d8	MOVEA RW6,RW4	MOVEA RW6 ,@RW4+d8	MOVEA RW7,RW4	MOVEA RW7 ,@RW4+d8
+5	MOVEA RW0,RW5	MOVEA RW0 ,@RW5+d8	MOVEA RW1,RW5	MOVEA RW1 ,@RW5+d8	MOVEA RW2,RW5	MOVEA RW2 ,@RW5+d8	MOVEA RW3,RW5	MOVEA RW3 ,@RW5+d8	MOVEA RW4,RW5	MOVEA RW4 ,@RW5+d8	MOVEA RW5,RW5	MOVEA RW5 ,@RW5+d8	MOVEA RW6,RW5	MOVEA RW6 ,@RW5+d8	MOVEA RW7,RW5	MOVEA RW7 ,@RW5+d8
+6	MOVEA RW0,RW6	MOVEA RW0 ,@RW6+d8	MOVEA RW1,RW6	MOVEA RW1 ,@RW6+d8	MOVEA RW2,RW6	MOVEA RW2 ,@RW6+d8	MOVEA RW3,RW6	MOVEA RW3 ,@RW6+d8	MOVEA RW4,RW6	MOVEA RW4 ,@RW6+d8	MOVEA RW5,RW6	MOVEA RW5 ,@RW6+d8	MOVEA RW6,RW6	MOVEA RW6 ,@RW6+d8	MOVEA RW7,RW6	MOVEA RW7 ,@RW6+d8
+7	MOVEA RW0,RW7	MOVEA RW0 ,@RW7+d8	MOVEA RW1,RW7	MOVEA RW1 ,@RW7+d8	MOVEA RW2,RW7	MOVEA RW2 ,@RW7+d8	MOVEA RW3,RW7	MOVEA RW3 ,@RW7+d8	MOVEA RW4,RW7	MOVEA RW4 ,@RW7+d8	MOVEA RW5,RW7	MOVEA RW5 ,@RW7+d8	MOVEA RW6,RW7	MOVEA RW6 ,@RW7+d8	MOVEA RW7,RW7	MOVEA RW7 ,@RW7+d8
+8	MOVEA RW0,@RW0	MOVEA RW0 ,@RW0+d16	MOVEA RW1,@RW0	MOVEA RW1 ,@RW0+d16	MOVEA RW2,@RW0	MOVEA RW2 ,@RW0+d16	MOVEA RW3,@RW0	MOVEA RW3 ,@RW0+d16	MOVEA RW4,@RW0	MOVEA RW4 ,@RW0+d16	MOVEA RW5,@RW0	MOVEA RW5 ,@RW0+d16	MOVEA RW6,@RW0	MOVEA RW6 ,@RW0+d16	MOVEA RW7,@RW0	MOVEA RW7 ,@RW0+d16
+9	MOVEA RW0,@RW1	MOVEA RW0 ,@RW1+d16	MOVEA RW1,@RW1	MOVEA RW1 ,@RW1+d16	MOVEA RW2,@RW1	MOVEA RW2 ,@RW1+d16	MOVEA RW3,@RW1	MOVEA RW3 ,@RW1+d16	MOVEA RW4,@RW1	MOVEA RW4 ,@RW1+d16	MOVEA RW5,@RW1	MOVEA RW5 ,@RW1+d16	MOVEA RW6,@RW1	MOVEA RW6 ,@RW1+d16	MOVEA RW7,@RW1	MOVEA RW7 ,@RW1+d16
+A	MOVEA RW0,@RW2	MOVEA RW0 ,@RW2+d16	MOVEA RW1,@RW2	MOVEA RW1 ,@RW2+d16	MOVEA RW2,@RW2	MOVEA RW2 ,@RW2+d16	MOVEA RW3,@RW2	MOVEA RW3 ,@RW2+d16	MOVEA RW4,@RW2	MOVEA RW4 ,@RW2+d16	MOVEA RW5,@RW2	MOVEA RW5 ,@RW2+d16	MOVEA RW6,@RW2	MOVEA RW6 ,@RW2+d16	MOVEA RW7,@RW2	MOVEA RW7 ,@RW2+d16
+B	MOVEA RW0,@RW3	MOVEA RW0 ,@RW3+d16	MOVEA RW1,@RW3	MOVEA RW1 ,@RW3+d16	MOVEA RW2,@RW3	MOVEA RW2 ,@RW3+d16	MOVEA RW3,@RW3	MOVEA RW3 ,@RW3+d16	MOVEA RW4,@RW3	MOVEA RW4 ,@RW3+d16	MOVEA RW5,@RW3	MOVEA RW5 ,@RW3+d16	MOVEA RW6,@RW3	MOVEA RW6 ,@RW3+d16	MOVEA RW7,@RW3	MOVEA RW7 ,@RW3+d16
+C	MOVEA RW0,@RW0+	MOVEA RW0 ,@RW0+RW7	MOVEA RW1,@RW0+	MOVEA RW1 ,@RW0+RW7	MOVEA RW2,@RW0+	MOVEA RW2 ,@RW0+RW7	MOVEA RW3,@RW0+	MOVEA RW3 ,@RW0+RW7	MOVEA RW4,@RW0+	MOVEA RW4 ,@RW0+RW7	MOVEA RW5,@RW0+	MOVEA RW5 ,@RW0+RW7	MOVEA RW6,@RW0+	MOVEA RW6 ,@RW0+RW7	MOVEA RW7,@RW0+	MOVEA RW7 ,@RW0+RW7
+D	MOVEA RW0,@RW1+	MOVEA RW0 ,@RW1+RW7	MOVEA RW1,@RW1+	MOVEA RW1 ,@RW1+RW7	MOVEA RW2,@RW1+	MOVEA RW2 ,@RW1+RW7	MOVEA RW3,@RW1+	MOVEA RW3 ,@RW1+RW7	MOVEA RW4,@RW1+	MOVEA RW4 ,@RW1+RW7	MOVEA RW5,@RW1+	MOVEA RW5 ,@RW1+RW7	MOVEA RW6,@RW1+	MOVEA RW6 ,@RW1+RW7	MOVEA RW7,@RW1+	MOVEA RW7 ,@RW1+RW7
+E	MOVEA RW0,@RW2+	MOVEA RW0 ,@PC+d16	MOVEA RW1,@RW2+	MOVEA RW1 ,@PC+d16	MOVEA RW2,@RW2+	MOVEA RW2 ,@PC+d16	MOVEA RW3,@RW2+	MOVEA RW3 ,@PC+d16	MOVEA RW4,@RW2+	MOVEA RW4 ,@PC+d16	MOVEA RW5,@RW2+	MOVEA RW5 ,@PC+d16	MOVEA RW6,@RW2+	MOVEA RW6 ,@PC+d16	MOVEA RW7,@RW2+	MOVEA RW7 ,@PC+d16
+F	MOVEA RW0,@RW3+	MOVEA RW0, addr16	MOVEA RW1,@RW3+	MOVEA RW1, addr16	MOVEA RW2,@RW3+	MOVEA RW2, addr16	MOVEA RW3,@RW3+	MOVEA RW3, addr16	MOVEA RW4,@RW3+	MOVEA RW4, addr16	MOVEA RW5,@RW3+	MOVEA RW5, addr16	MOVEA RW6,@RW3+	MOVEA RW6, addr16	MOVEA RW7@RW3+	MOVEA RW7, addr16

C.8 MOV Ri, ea Instruction Map

Table C-15 lists MOV Ri, ea instruction map.

■ MOV Ri, ea Instruction Map

Table C-15 MOV Ri, ea Instruction (First Byte = 7A$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOV R0, R0	MOV R0, @RW0+d8	MOV R1, R0	MOV R1, @RW0+d8	MOV R2, R0	MOV R2, @RW0+d8	MOV R3, R0	MOV R3, @RW0+d8	MOV R4, R0	MOV R4, @RW0+d8	MOV R5, R0	MOV R5, @RW0+d8	MOV R6, R0	MOV R6, @RW0+d8	MOV R7, R0	MOV R7, @RW0+d8
+1	MOV R0, R1	MOV R0, @RW1+d8	MOV R1, R1	MOV R1, @RW1+d8	MOV R2, R1	MOV R2, @RW1+d8	MOV R3, R1	MOV R3, @RW1+d8	MOV R4, R1	MOV R4, @RW1+d8	MOV R5, R1	MOV R5, @RW1+d8	MOV R6, R1	MOV R6, @RW1+d8	MOV R7, R1	MOV R7, @RW1+d8
+2	MOV R0, R2	MOV R0, @RW2+d8	MOV R1, R2	MOV R1, @RW2+d8	MOV R2, R2	MOV R2, @RW2+d8	MOV R3, R2	MOV R3, @RW2+d8	MOV R4, R2	MOV R4, @RW2+d8	MOV R5, R2	MOV R5, @RW2+d8	MOV R6, R2	MOV R6, @RW2+d8	MOV R7, R2	MOV R7, @RW2+d8
+3	MOV R0, R3	MOV R0, @RW3+d8	MOV R1, R3	MOV R1, @RW3+d8	MOV R2, R3	MOV R2, @RW3+d8	MOV R3, R3	MOV R3, @RW3+d8	MOV R4, R3	MOV R4, @RW3+d8	MOV R5, R3	MOV R5, @RW3+d8	MOV R6, R3	MOV R6, @RW3+d8	MOV R7, R3	MOV R7, @RW3+d8
+4	MOV R0, R4	MOV R0, @RW4+d8	MOV R1, R4	MOV R1, @RW4+d8	MOV R2, R4	MOV R2, @RW4+d8	MOV R3, R4	MOV R3, @RW4+d8	MOV R4, R4	MOV R4, @RW4+d8	MOV R5, R4	MOV R5, @RW4+d8	MOV R6, R4	MOV R6, @RW4+d8	MOV R7, R4	MOV R7, @RW4+d8
+5	MOV R0, R5	MOV R0, @RW5+d8	MOV R1, R5	MOV R1, @RW5+d8	MOV R2, R5	MOV R2, @RW5+d8	MOV R3, R5	MOV R3, @RW5+d8	MOV R4, R5	MOV R4, @RW5+d8	MOV R5, R5	MOV R5, @RW5+d8	MOV R6, R5	MOV R6, @RW5+d8	MOV R7, R5	MOV R7, @RW5+d8
+6	MOV R0, R6	MOV R0, @RW6+d8	MOV R1, R6	MOV R1, @RW6+d8	MOV R2, R6	MOV R2, @RW6+d8	MOV R3, R6	MOV R3, @RW6+d8	MOV R4, R6	MOV R4, @RW6+d8	MOV R5, R6	MOV R5, @RW6+d8	MOV R6, R6	MOV R6, @RW6+d8	MOV R7, R6	MOV R7, @RW6+d8
+7	MOV R0, R7	MOV R0, @RW7+d8	MOV R1, R7	MOV R1, @RW7+d8	MOV R2, R7	MOV R2, @RW7+d8	MOV R3, R7	MOV R3, @RW7+d8	MOV R4, R7	MOV R4, @RW7+d8	MOV R5, R7	MOV R5, @RW7+d8	MOV R6, R7	MOV R6, @RW7+d8	MOV R7, R7	MOV R7, @RW7+d8
+8	MOV R0,@RW0	MOV R0, @RW0+d16	MOV R1,@RW0	MOV R1, @RW0+d16	MOV R2,@RW0	MOV R2, @RW0+d16	MOV R3,@RW0	MOV R3, @RW0+d16	MOV R4,@RW0	MOV R4, @RW0+d16	MOV R5,@RW0	MOV R5, @RW0+d16	MOV R6,@RW0	MOV R6, @RW0+d16	MOV R7,@RW0	MOV R7, @RW0+d16
+9	MOV R0,@RW1	MOV R0, @RW1+d16	MOV R1,@RW1	MOV R1, @RW1+d16	MOV R2,@RW1	MOV R2, @RW1+d16	MOV R3,@RW1	MOV R3, @RW1+d16	MOV R4,@RW1	MOV R4, @RW1+d16	MOV R5,@RW1	MOV R5, @RW1+d16	MOV R6,@RW1	MOV R6, @RW1+d16	MOV R7,@RW1	MOV R7, @RW1+d16
+A	MOV R0,@RW2	MOV R0, @RW2+d16	MOV R1,@RW2	MOV R1, @RW2+d16	MOV R2,@RW2	MOV R2, @RW2+d16	MOV R3,@RW2	MOV R3, @RW2+d16	MOV R4,@RW2	MOV R4, @RW2+d16	MOV R5,@RW2	MOV R5, @RW2+d16	MOV R6,@RW2	MOV R6, @RW2+d16	MOV R7,@RW2	MOV R7, @RW2+d16
+B	MOV R0,@RW3	MOV R0, @RW3+d16	MOV R1,@RW3	MOV R1, @RW3+d16	MOV R2,@RW3	MOV R2, @RW3+d16	MOV R3,@RW3	MOV R3, @RW3+d16	MOV R4,@RW3	MOV R4, @RW3+d16	MOV R5,@RW3	MOV R5, @RW3+d16	MOV R6,@RW3	MOV R6, @RW3+d16	MOV R7,@RW3	MOV R7, @RW3+d16
+C	MOV R0, @RW0+	MOV R0, @RW0+RW7	MOV R1, @RW0+	MOV R1, @RW0+RW7	MOV R2, @RW0+	MOV R2, @RW0+RW7	MOV R3, @RW0+	MOV R3, @RW0+RW7	MOV R4, @RW0+	MOV R4, @RW0+RW7	MOV R5, @RW0+	MOV R5, @RW0+RW7	MOV R6, @RW0+	MOV R6, @RW0+RW7	MOV R7, @RW0+	MOV R7, @RW0+RW7
+D	MOV R0, @RW1+	MOV R0, @RW1+RW7	MOV R1, @RW1+	MOV R1, @RW1+RW7	MOV R2, @RW1+	MOV R2, @RW1+RW7	MOV R3, @RW1+	MOV R3, @RW1+RW7	MOV R4, @RW1+	MOV R4, @RW1+RW7	MOV R5, @RW1+	MOV R5, @RW1+RW7	MOV R6, @RW1+	MOV R6, @RW1+RW7	MOV R7, @RW1+	MOV R7, @RW1+RW7
+E	MOV R0, @RW2+	MOV R0, @PC+d16	MOV R1, @RW2+	MOV R1, @PC+d16	MOV R2, @RW2+	MOV R2, @PC+d16	MOV R3, @RW2+	MOV R3, @PC+d16	MOV R4, @RW2+	MOV R4, @PC+d16	MOV R5, @RW2+	MOV R5, @PC+d16	MOV R6, @RW2+	MOV R6, @PC+d16	MOV R7, @RW2+	MOV R7, @PC+d16
+F	MOV R0, @RW3+	MOV R0, addr16	MOV R1, @RW3+	MOV R1, addr16	MOV R2, @RW3+	MOV R2, addr16	MOV R3, @RW3+	MOV R3, addr16	MOV R4, @RW3+	MOV R4, addr16	MOV R5, @RW3+	MOV R5, addr16	MOV R6, @RW3+	MOV R6, addr16	MOV R7, @RW3+	MOV R7, addr16

C.9 MOVW RWi, ea Instruction Map

Table C-16 lists MOVW RWi, ea instruction map.

■ MOVW RWi, ea Instruction Map

Table C-16 MOVW RWi, ea Instruction (First Byte = $7B_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOVW RW0, RW0	MOVW @RW0+d8	MOVW RW1, RW0	MOVW RW1, @RW0+d8	MOVW RW2, RW0	MOVW RW2, @RW0+d8	MOVW RW3, RW0	MOVW RW3, @RW0+d8	MOVW RW4, RW0	MOVW RW4, @RW0+d8	MOVW RW5, RW0	MOVW RW5, @RW0+d8	MOVW RW6, RW0	MOVW RW6, @RW0+d8	MOVW RW7, RW0	MOVW RW7, @RW0+d8
+1	MOVW RW0, RW1	MOVW @RW1+d8	MOVW RW1, RW1	MOVW RW1, @RW1+d8	MOVW RW2, RW1	MOVW RW2, @RW1+d8	MOVW RW3, RW1	MOVW RW3, @RW1+d8	MOVW RW4, RW1	MOVW RW4, @RW1+d8	MOVW RW5, RW1	MOVW RW5, @RW1+d8	MOVW RW6, RW1	MOVW RW6, @RW1+d8	MOVW RW7, RW1	MOVW RW7, @RW1+d8
+2	MOVW RW0, RW2	MOVW @RW2+d8	MOVW RW1, RW2	MOVW RW1, @RW2+d8	MOVW RW2, RW2	MOVW RW2, @RW2+d8	MOVW RW3, RW2	MOVW RW3, @RW2+d8	MOVW RW4, RW2	MOVW RW4, @RW2+d8	MOVW RW5, RW2	MOVW RW5, @RW2+d8	MOVW RW6, RW2	MOVW RW6, @RW2+d8	MOVW RW7, RW2	MOVW RW7, @RW2+d8
+3	MOVW RW0, RW3	MOVW @RW3+d8	MOVW RW1, RW3	MOVW RW1, @RW3+d8	MOVW RW2, RW3	MOVW RW2, @RW3+d8	MOVW RW3, RW3	MOVW RW3, @RW3+d8	MOVW RW4, RW3	MOVW RW4, @RW3+d8	MOVW RW5, RW3	MOVW RW5, @RW3+d8	MOVW RW6, RW3	MOVW RW6, @RW3+d8	MOVW RW7, RW3	MOVW RW7, @RW3+d8
+4	MOVW RW0, RW4	MOVW @RW4+d8	MOVW RW1, RW4	MOVW RW1, @RW4+d8	MOVW RW2, RW4	MOVW RW2, @RW4+d8	MOVW RW3, RW4	MOVW RW3, @RW4+d8	MOVW RW4, RW4	MOVW RW4, @RW4+d8	MOVW RW5, RW4	MOVW RW5, @RW4+d8	MOVW RW6, RW4	MOVW RW6, @RW4+d8	MOVW RW7, RW4	MOVW RW7, @RW4+d8
+5	MOVW RW0, RW5	MOVW @RW5+d8	MOVW RW1, RW5	MOVW RW1, @RW5+d8	MOVW RW2, RW5	MOVW RW2, @RW5+d8	MOVW RW3, RW5	MOVW RW3, @RW5+d8	MOVW RW4, RW5	MOVW RW4, @RW5+d8	MOVW RW5, RW5	MOVW RW5, @RW5+d8	MOVW RW6, RW5	MOVW RW6, @RW5+d8	MOVW RW7, RW5	MOVW RW7, @RW5+d8
+6	MOVW RW0, RW6	MOVW @RW6+d8	MOVW RW1, RW6	MOVW RW1, @RW6+d8	MOVW RW2, RW6	MOVW RW2, @RW6+d8	MOVW RW3, RW6	MOVW RW3, @RW6+d8	MOVW RW4, RW6	MOVW RW4, @RW6+d8	MOVW RW5, RW6	MOVW RW5, @RW6+d8	MOVW RW6, RW6	MOVW RW6, @RW6+d8	MOVW RW7, RW6	MOVW RW7, @RW6+d8
+7	MOVW RW0, RW7	MOVW @RW7+d8	MOVW RW1, RW7	MOVW RW1, @RW7+d8	MOVW RW2, RW7	MOVW RW2, @RW7+d8	MOVW RW3, RW7	MOVW RW3, @RW7+d8	MOVW RW4, RW7	MOVW RW4, @RW7+d8	MOVW RW5, RW7	MOVW RW5, @RW7+d8	MOVW RW6, RW7	MOVW RW6, @RW7+d8	MOVW RW7, RW7	MOVW RW7, @RW7+d8
+8	MOVW RW0,@RW0	MOVW @RW0+d16	MOVW RW1,@RW0	MOVW RW1, @RW0+d16	MOVW RW2,@RW0	MOVW RW2, @RW0+d16	MOVW RW3,@RW0	MOVW RW3, @RW0+d16	MOVW RW4,@RW0	MOVW RW4, @RW0+d16	MOVW RW5,@RW0	MOVW RW5, @RW0+d16	MOVW RW6,@RW0	MOVW RW6, @RW0+d16	MOVW RW7,@RW0	MOVW RW7, @RW0+d16
+9	MOVW RW0,@RW1	MOVW @RW1+d16	MOVW RW1,@RW1	MOVW RW1, @RW1+d16	MOVW RW2,@RW1	MOVW RW2, @RW1+d16	MOVW RW3,@RW1	MOVW RW3, @RW1+d16	MOVW RW4,@RW1	MOVW RW4, @RW1+d16	MOVW RW5,@RW1	MOVW RW5, @RW1+d16	MOVW RW6,@RW1	MOVW RW6, @RW1+d16	MOVW RW7,@RW1	MOVW RW7, @RW1+d16
+A	MOVW RW0,@RW2	MOVW @RW2+d16	MOVW RW1,@RW2	MOVW RW1, @RW2+d16	MOVW RW2,@RW2	MOVW RW2, @RW2+d16	MOVW RW3,@RW2	MOVW RW3, @RW2+d16	MOVW RW4,@RW2	MOVW RW4, @RW2+d16	MOVW RW5,@RW2	MOVW RW5, @RW2+d16	MOVW RW6,@RW2	MOVW RW6, @RW2+d16	MOVW RW7,@RW2	MOVW RW7, @RW2+d16
+B	MOVW RW0,@RW3	MOVW @RW3+d16	MOVW RW1,@RW3	MOVW RW1, @RW3+d16	MOVW RW2,@RW3	MOVW RW2, @RW3+d16	MOVW RW3,@RW3	MOVW RW3, @RW3+d16	MOVW RW4,@RW3	MOVW RW4, @RW3+d16	MOVW RW5,@RW3	MOVW RW5, @RW3+d16	MOVW RW6,@RW3	MOVW RW6, @RW3+d16	MOVW RW7,@RW3	MOVW RW7, @RW3+d16
+C	MOVW RW0,@RW0+	MOVW @RW0+RW7	MOVW RW1,@RW0+	MOVW RW1, @RW0+RW7	MOVW RW2,@RW0+	MOVW RW2, @RW0+RW7	MOVW RW3,@RW0+	MOVW RW3, @RW0+RW7	MOVW RW4,@RW0+	MOVW RW4, @RW0+RW7	MOVW RW5,@RW0+	MOVW RW5, @RW0+RW7	MOVW RW6,@RW0+	MOVW RW6, @RW0+RW7	MOVW RW7,@RW0+	MOVW RW7, @RW0+RW7
+D	MOVW RW0, @RW1+	MOVW @RW1+RW7	MOVW RW1, @RW1+	MOVW RW1, @RW1+RW7	MOVW RW2, @RW1+	MOVW RW2, @RW1+RW7	MOVW RW3, @RW1+	MOVW RW3, @RW1+RW7	MOVW RW4, @RW1+	MOVW RW4, @RW1+RW7	MOVW RW5, @RW1+	MOVW RW5, @RW1+RW7	MOVW RW6, @RW1+	MOVW RW6, @RW1+RW7	MOVW RW7, @RW1+	MOVW RW7, @RW1+RW7
+E	MOVW RW0, @RW2+	MOVW @PC+d16	MOVW RW1, @RW2+	MOVW RW1, @PC+d16	MOVW RW2, @RW2+	MOVW RW2, @PC+d16	MOVW RW3, @RW2+	MOVW RW3, @PC+d16	MOVW RW4, @RW2+	MOVW RW4, @PC+d16	MOVW RW5, @RW2+	MOVW @PC+d16	MOVW RW6, @RW2+	MOVW RW6, @PC+d16	MOVW RW7, @RW2+	MOVW RW7, @PC+d16
+F	MOVW RW0, @RW3+	MOVW RW0, addr16	MOVW RW1, @RW3+	MOVW RW1, addr16	MOVW RW2, @RW3+	MOVW RW2, addr16	MOVW RW3, @RW3+	MOVW RW3, addr16	MOVW RW4, @RW3+	MOVW RW4, addr16	MOVW RW5, @RW3+	MOVW RW5, addr16	MOVW RW6, @RW3+	MOVW RW6, addr16	MOVW RW7, @RW3+	MOVW RW7, addr16

C.10 MOV ea, Ri Instruction Map

Table C-17 lists MOV ea, Ri instruction map.

■ MOV ea, Ri Instruction Map

Table C-17 MOV ea, Ri Instruction (First Byte = 7C$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOV R0, R0	MOV @RW0+d8, R0	MOV R0, R1	MOV @RW0+d8, R1	MOV R0, R2	MOV @RW0+d8, R2	MOV R0, R3	MOV @RW0+d8, R3	MOV R0, R4	MOV @RW0+d8, R4	MOV R0, R5	MOV @RW0+d8, R5	MOV R0, R6	MOV @RW0+d8, R6	MOV R0, R7	MOV @RW0+d8, R7
+1	MOV R1, R0	MOV @RW1+d8, R0	MOV R1, R1	MOV @RW1+d8, R1	MOV R1, R2	MOV @RW1+d8, R2	MOV R1, R3	MOV @RW1+d8, R3	MOV R1, R4	MOV @RW1+d8, R4	MOV R1, R5	MOV @RW1+d8, R5	MOV R1, R6	MOV @RW1+d8, R6	MOV R1, R7	MOV @RW1+d8, R7
+2	MOV R2, R0	MOV @RW2+d8, R0	MOV R2, R1	MOV @RW2+d8, R1	MOV R2, R2	MOV @RW2+d8, R2	MOV R2, R3	MOV @RW2+d8, R3	MOV R2, R4	MOV @RW2+d8, R4	MOV R2, R5	MOV @RW2+d8, R5	MOV R2, R6	MOV @RW2+d8, R6	MOV R2, R7	MOV @RW2+d8, R7
+3	MOV R3, R0	MOV @RW3+d8, R0	MOV R3, R1	MOV @RW3+d8, R1	MOV R3, R2	MOV @RW3+d8, R2	MOV R3, R3	MOV @RW3+d8, R3	MOV R3, R4	MOV @RW3+d8, R4	MOV R3, R5	MOV @RW3+d8, R5	MOV R3, R6	MOV @RW3+d8, R6	MOV R3, R7	MOV @RW3+d8, R7
+4	MOV R4, R0	MOV @RW4+d8, R0	MOV R4, R1	MOV @RW4+d8, R1	MOV R4, R2	MOV @RW4+d8, R2	MOV R4, R3	MOV @RW4+d8, R3	MOV R4, R4	MOV @RW4+d8, R4	MOV R4, R5	MOV @RW4+d8, R5	MOV R4, R6	MOV @RW4+d8, R6	MOV R4, R7	MOV @RW4+d8, R7
+5	MOV R5, R0	MOV @RW5+d8, R0	MOV R5, R1	MOV @RW5+d8, R1	MOV R5, R2	MOV @RW5+d8, R2	MOV R5, R3	MOV @RW5+d8, R3	MOV R5, R4	MOV @RW5+d8, R4	MOV R5, R5	MOV @RW5+d8, R5	MOV R5, R6	MOV @RW5+d8, R6	MOV R5, R7	MOV @RW5+d8, R7
+6	MOV R6, R0	MOV @RW6+d8, R0	MOV R6, R1	MOV @RW6+d8, R1	MOV R6, R2	MOV @RW6+d8, R2	MOV R6, R3	MOV @RW6+d8, R3	MOV R6, R4	MOV @RW6+d8, R4	MOV R6, R5	MOV @RW6+d8, R5	MOV R6, R6	MOV @RW6+d8, R6	MOV R6, R7	MOV @RW6+d8, R7
+7	MOV R7, R0	MOV @RW7+d8, R0	MOV R7, R1	MOV @RW7+d8, R1	MOV R7, R2	MOV @RW7+d8, R2	MOV R7, R3	MOV @RW7+d8, R3	MOV R7, R4	MOV @RW7+d8, R4	MOV R7, R5	MOV @RW7+d8, R5	MOV R7, R6	MOV @RW7+d8, R6	MOV R7, R7	MOV @RW7+d8, R7
+8	MOV @RW0, R0	MOV @RW0+d16, R0	MOV @RW0, R1	MOV @RW0+d16, R1	MOV @RW0, R2	MOV @RW0+d16, R2	MOV @RW0, R3	MOV @RW0+d16, R3	MOV @RW0, R4	MOV @RW0+d16, R4	MOV @RW0, R5	MOV @RW0+d16, R5	MOV @RW0, R6	MOV @RW0+d16, R6	MOV @RW0, R7	MOV @RW0+d16, R7
+9	MOV @RW1, R0	MOV @RW1+d16, R0	MOV @RW1, R1	MOV @RW1+d16, R1	MOV @RW1, R2	MOV @RW1+d16, R2	MOV @RW1, R3	MOV @RW1+d16, R3	MOV @RW1, R4	MOV @RW1+d16, R4	MOV @RW1, R5	MOV @RW1+d16, R5	MOV @RW1, R6	MOV @RW1+d16, R6	MOV @RW1, R7	MOV @RW1+d16, R7
+A	MOV @RW2, R0	MOV @RW2+d16, R0	MOV @RW2, R1	MOV @RW2+d16, R1	MOV @RW2, R2	MOV @RW2+d16, R2	MOV @RW2, R3	MOV @RW2+d16, R3	MOV @RW2, R4	MOV @RW2+d16, R4	MOV @RW2, R5	MOV @RW2+d16, R5	MOV @RW2, R6	MOV @RW2+d16, R6	MOV @RW2, R7	MOV @RW2+d16, R7
+B	MOV @RW3, R0	MOV @RW3+d16, R0	MOV @RW3, R1	MOV @RW3+d16, R1	MOV @RW3, R2	MOV @RW3+d16, R2	MOV @RW3, R3	MOV @RW3+d16, R3	MOV @RW3, R4	MOV @RW3+d16, R4	MOV @RW3, R5	MOV @RW3+d16, R5	MOV @RW3, R6	MOV @RW3+d16, R6	MOV @RW3, R7	MOV @RW3+d16, R7
+C	MOV @RW0+, R0	MOV @RW0+RW7, R0	MOV @RW0+, R1	MOV @RW0+RW7, R1	MOV @RW0+, R2	MOV @RW0+RW7, R2	MOV @RW0+, R3	MOV @RW0+RW7, R3	MOV @RW0+, R4	MOV @RW0+RW7, R4	MOV @RW0+, R5	MOV @RW0+RW7, R5	MOV @RW0+, R6	MOV @RW0+RW7, R6	MOV @RW0+, R7	MOV @RW0+RW7, R7
+D	MOV @RW1+, R0	MOV @RW1+RW7, R0	MOV @RW1+, R1	MOV @RW1+RW7, R1	MOV @RW1+, R2	MOV @RW1+RW7, R2	MOV @RW1+, R3	MOV @RW1+RW7, R3	MOV @RW1+, R4	MOV @RW1+RW7, R4	MOV @RW1+, R5	MOV @RW1+RW7, R5	MOV @RW1+, R6	MOV @RW1+RW7, R6	MOV @RW1+, R7	MOV @RW1+RW7, R7
+E	MOV @RW2+, R0	MOV @PC+d16, R0	MOV @RW2+, R1	MOV @PC+d16, R1	MOV @RW2+, R2	MOV @PC+d16, R2	MOV @RW2+, R3	MOV @PC+d16, R3	MOV @RW2+, R4	MOV @PC+d16, R4	MOV @RW2+, R5	MOV @PC+d16, R5	MOV @RW2+, R6	MOV @PC+d16, R6	MOV @RW2+, R7	MOV @PC+d16, R7
+F	MOV @RW3+, R0	MOV addr16, R0	MOV @RW3+, R1	MOV addr16, R1	MOV @RW3+, R2	MOV addr16, R2	MOV @RW3+, R3	MOV addr16, R3	MOV @RW3+, R4	MOV addr16, R4	MOV @RW3+, R5	MOV addr16, R5	MOV @RW3+, R6	MOV addr16, R6	MOV @RW3+, R7	MOV addr16, R7

C.11 MOVW ea, RWi Instruction Map

Table C-18 lists MOVW ea, RWi instruction map.

■ MOVW ea, RWi Instruction Map

Table C-18 MOVW ea, RWi Instruction (First Byte = 7D$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	MOVW RW0, RW0	MOVW @RW0+d8, RW0	MOVW RW0, RW1	MOVW @RW0+d8, RW1	MOVW RW0, RW2	MOVW @RW0+d8, RW2	MOVW RW0, RW3	MOVW @RW0+d8, RW3	MOVW RW0, RW4	MOVW @RW0+d8, RW4	MOVW RW0, RW5	MOVW @RW0+d8, RW5	MOVW RW0, RW6	MOVW @RW0+d8, RW6	MOVW RW0, RW7	MOVW @RW0+d8, RW7
+1	MOVW RW1, RW0	MOVW @RW1+d8, RW0	MOVW RW1, RW1	MOVW @RW1+d8, RW1	MOVW RW1, RW2	MOVW @RW1+d8, RW2	MOVW RW1, RW3	MOVW @RW1+d8, RW3	MOVW RW1, RW4	MOVW @RW1+d8, RW4	MOVW RW1, RW5	MOVW @RW1+d8, RW5	MOVW RW1, RW6	MOVW @RW1+d8, RW6	MOVW RW1, RW7	MOVW @RW1+d8, RW7
+2	MOVW RW2, RW0	MOVW @RW2+d8, RW0	MOVW RW2, RW1	MOVW @RW2+d8, RW1	MOVW RW2, RW2	MOVW @RW2+d8, RW2	MOVW RW2, RW3	MOVW @RW2+d8, RW3	MOVW RW2, RW4	MOVW @RW2+d8, RW4	MOVW RW2, RW5	MOVW @RW2+d8, RW5	MOVW RW2, RW6	MOVW @RW2+d8, RW6	MOVW RW2, RW7	MOVW @RW2+d8, RW7
+3	MOVW RW3, RW0	MOVW @RW3+d8, RW0	MOVW RW3, RW1	MOVW @RW3+d8, RW1	MOVW RW3, RW2	MOVW @RW3+d8, RW2	MOVW RW3, RW3	MOVW @RW3+d8, RW3	MOVW RW3, RW4	MOVW @RW3+d8, RW4	MOVW RW3, RW5	MOVW @RW3+d8, RW5	MOVW RW3, RW6	MOVW @RW3+d8, RW6	MOVW RW3, RW7	MOVW @RW3+d8, RW7
+4	MOVW RW4, RW0	MOVW @RW4+d8, RW0	MOVW RW4, RW1	MOVW @RW4+d8, RW1	MOVW RW4, RW2	MOVW @RW4+d8, RW2	MOVW RW4, RW3	MOVW @RW4+d8, RW3	MOVW RW4, RW4	MOVW @RW4+d8, RW4	MOVW RW4, RW5	MOVW @RW4+d8, RW5	MOVW RW4, RW6	MOVW @RW4+d8, RW6	MOVW RW4, RW7	MOVW @RW4+d8, RW7
+5	MOVW RW5, RW0	MOVW @RW5+d8, RW0	MOVW RW5, RW1	MOVW @RW5+d8, RW1	MOVW RW5, RW2	MOVW @RW5+d8, RW2	MOVW RW5, RW3	MOVW @RW5+d8, RW3	MOVW RW5, RW4	MOVW @RW5+d8, RW4	MOVW RW5, RW5	MOVW @RW5+d8, RW5	MOVW RW5, RW6	MOVW @RW5+d8, RW6	MOVW RW5, RW7	MOVW @RW5+d8, RW7
+6	MOVW RW6, RW0	MOVW @RW6+d8, RW0	MOVW RW6, RW1	MOVW @RW6+d8, RW1	MOVW RW6, RW2	MOVW @RW6+d8, RW2	MOVW RW6, RW3	MOVW @RW6+d8, RW3	MOVW RW6, RW4	MOVW @RW6+d8, RW4	MOVW RW6, RW5	MOVW @RW6+d8, RW5	MOVW RW6, RW6	MOVW @RW6+d8, RW6	MOVW RW6, RW7	MOVW @RW6+d8, RW7
+7	MOVW RW7, RW0	MOVW @RW7+d8, RW0	MOVW RW7, RW1	MOVW @RW7+d8, RW1	MOVW RW7, RW2	MOVW @RW7+d8, RW2	MOVW RW7, RW3	MOVW @RW7+d8, RW3	MOVW RW7, RW4	MOVW @RW7+d8, RW4	MOVW RW7, RW5	MOVW @RW7+d8, RW5	MOVW RW7, RW6	MOVW @RW7+d8, RW6	MOVW RW7, RW7	MOVW @RW7+d8, RW7
+8	MOVW @RW0, RW0	MOVW@RW0 +d16, RW0	MOVW @RW0, RW1	MOVW@RW0 +d16, RW1	MOVW @RW0, RW2	MOVW@RW0 +d16, RW2	MOVW @RW0, RW3	MOVW@RW0 +d16, RW3	MOVW @RW0, RW4	MOVW@RW0 +d16, RW4	MOVW @RW0, RW5	MOVW@RW0 +d16, RW5	MOVW @RW0, RW6	MOVW@RW0 +d16, RW6	MOVW @RW0, RW7	MOVW@RW0 +d16, RW7
+9	MOVW @RW1, RW0	MOVW@RW1 +d16, RW0	MOVW @RW1, RW1	MOVW@RW1 +d16, RW1	MOVW @RW1, RW2	MOVW@RW1 +d16, RW2	MOVW @RW1, RW3	MOVW@RW1 +d16, RW3	MOVW @RW1, RW4	MOVW@RW1 +d16, RW4	MOVW @RW1, RW5	MOVW@RW1 +d16, RW5	MOVW @RW1, RW6	MOVW@RW1 +d16, RW6	MOVW @RW1, RW7	MOVW@RW1 +d16, RW7
+A	MOVW @RW2, RW0	MOVW@RW2 +d16, RW0	MOVW @RW2, RW1	MOVW@RW2 +d16, RW1	MOVW @RW2, RW2	MOVW@RW2 +d16, RW2	MOVW @RW2, RW3	MOVW@RW2 +d16, RW3	MOVW @RW2, RW4	MOVW@RW2 +d16, RW4	MOVW @RW2, RW5	MOVW@RW2 +d16, RW5	MOVW @RW2, RW6	MOVW@RW2 +d16, RW6	MOVW @RW2, RW7	MOVW@RW2 +d16, RW7
+B	MOVW @RW3, RW0	MOVW@RW3 +d16, RW0	MOVW @RW3, RW1	MOVW@RW3 +d16, RW1	MOVW @RW3, RW2	MOVW@RW3 +d16, RW2	MOVW @RW3, RW3	MOVW@RW3 +d16, RW3	MOVW @RW3, RW4	MOVW@RW3 +d16, RW4	MOVW @RW3, RW5	MOVW@RW3 +d16, RW5	MOVW @RW3, RW6	MOVW@RW3 +d16, RW6	MOVW @RW3, RW7	MOVW@RW3 +d16, RW7
+C	MOVW @RW0+,RW0	MOVW@RW0 +RW7,RW0	MOVW @RW0+, RW1	MOVW@RW0 +RW7,RW1	MOVW @RW0+,RW2	MOVW@RW0 +RW7,RW2	MOVW @RW0+,RW3	MOVW@RW0 +RW7,RW3	MOVW @RW0+,RW4	MOVW@RW0 +RW7,RW4	MOVW @RW0+,RW5	MOVW@RW0 +RW7,RW5	MOVW @RW0+,RW6	MOVW@RW0 +RW7,RW6	MOVW @RW0+,RW7	MOVW@RW0 +RW7,RW7
+D	MOVW @RW1+,RW0	MOVW@RW1 +RW7,RW0	MOVW @RW1+,RW1	MOVW@RW1 +RW7,RW1	MOVW @RW1+,RW2	MOVW@RW1 +RW7,RW2	MOVW @RW1+,RW3	MOVW@RW1 -+RW7,RW3	MOVW @RW1+,RW4	MOVW@RW1 +RW7,RW4	MOVW @RW1+,RW5	MOVW@RW1 +RW7,RW5	MOVW @RW1+,RW6	MOVW@RW1 +RW7,RW6	MOVW @RW1+,RW7	MOVW@RW1 +RW7,RW7
+E	MOVW @RW2+,RW0	MOVW@PC +d16, RW0	MOVW @RW2+,RW1	MOVW@PC +d16, RW1	MOVW @RW2+,RW2	MOVW@PC +d16, RW2	MOVW @RW2+,RW3	MOVW@PC +d16, RW3	MOVW @RW2+,RW4	MOVW@PC +d16, RW4	MOVW @RW2+,RW5	MOVW@PC +d16, RW5	MOVW @RW2+,RW6	MOVW @PC +d16, RW6	MOVW @RW2+,RW7	MOVW@PC +d16, RW7
+F	MOVW @RW3+,RW0	MOVW addr16, RW0	MOVW @RW3+,RW1	MOVW addr16, RW1	MOVW @RW3+,RW2	MOVW addr16, RW2	MOVW @RW3+,RW3	MOVW addr16, RW3	MOVW @RW3+,RW4	MOVW addr16, RW4	MOVW @RW3+,RW5	MOVW addr16, RW5	MOVW @RW3+,RW6	MOVW addr16, RW6	MOVW @RW3+,RW7	MOVW addr16, RW7

C.12 XCH Ri, ea Instruction Map

Table C-19 lists XCH Ri, ea instruction map.

■ XCH Ri, ea Instruction Map

Table C-19 XCH Ri, ea Instruction (First Byte = 7E$_H$)

	00	10	20	30	40	50	60	70	80	90	A	B0	C0	D0	E0	F0
+0	XCH R0, R0	XCH R0, @RW0+d8	XCH R1, R0	XCH R1, @RW0+d8	XCH R2, R0	XCH R2, @RW0+d8	XCH R3, R0	XCH R3, @RW0+d8	XCH R4, R0	XCH R4, @RW0+d8	XCH R5, R0	XCH R5, @RW0+d8	XCH R6, R0	XCH R6, @RW0+d8	XCH R7, R0	XCH R7, @RW0+d8
+1	XCH R0, R1	XCH R0, @RW1+d8	XCH R1, R1	XCH R1, @RW1+d8	XCH R2, R1	XCH R2, @RW1+d8	XCH R3, R1	XCH R3, @RW1+d8	XCH R4, R1	XCH R4, @RW1+d8	XCH R5, R1	XCH R5, @RW1+d8	XCH R6, R1	XCH R6, @RW1+d8	XCH R7, R1	XCH R7, @RW1+d8
+2	XCH R0, R2	XCH R0, @RW2+d8	XCH R1, R2	XCH R1, @RW2+d8	XCH R2, R2	XCH R2, @RW2+d8	XCH R3, R2	XCH R3, @RW2+d8	XCH R4, R2	XCH R4, @RW2+d8	XCH R5, R2	XCH R5, @RW2+d8	XCH R6, R2	XCH R6, @RW2+d8	XCH R7, R2	XCH R7, @RW2+d8
+3	XCH R0, R3	XCH R0, @RW3+d8	XCH R1, R3	XCH R1, @RW3+d8	XCH R2, R3	XCH R2, @RW3+d8	XCH R3, R3	XCH R3, @RW3+d8	XCH R4, R3	XCH R4, @RW3+d8	XCH R5, R3	XCH R5, @RW3+d8	XCH R6, R3	XCH R6, @RW3+d8	XCH R7, R3	XCH R7, @RW3+d8
+4	XCH R0, R4	XCH R0, @RW4+d8	XCH R1, R4	XCH R1, @RW4+d8	XCH R2, R4	XCH R2, @RW4+d8	XCH R3, R4	XCH R3, @RW4+d8	XCH R4, R4	XCH R4, @RW4+d8	XCH R5, R4	XCH R5, @RW4+d8	XCH R6, R4	XCH R6, @RW4+d8	XCH R7, R4	XCH R7, @RW4+d8
+5	XCH R0, R5	XCH R0, @RW5+d8	XCH R1, R5	XCH R1, @RW5+d8	XCH R2, R5	XCH R2, @RW5+d8	XCH R3, R5	XCH R3, @RW5+d8	XCH R4, R5	XCH R4, @RW5+d8	XCH R5, R5	XCH R5, @RW5+d8	XCH R6, R5	XCH R6, @RW5+d8	XCH R7, R5	XCH R7, @RW5+d8
+6	XCH R0, R6	XCH R0, @RW6+d8	XCH R1, R6	XCH R1, @RW6+d8	XCH R2, R6	XCH R2, @RW6+d8	XCH R3, R6	XCH R3, @RW6+d8	XCH R4, R6	XCH R4, @RW6+d8	XCH R5, R6	XCH R5, @RW6+d8	XCH R6, R6	XCH R6, @RW6+d8	XCH R7, R6	XCH R7, @RW6+d8
+7	XCH R0, R7	XCH R0, @RW7+d8	XCH R1, R7	XCH R1, @RW7+d8	XCH R2, R7	XCH R2, @RW7+d8	XCH R3, R7	XCH R3, @RW7+d8	XCH R4, R7	XCH R4, @RW7+d8	XCH R5, R7	XCH R5, @RW7+d8	XCH R6, R7	XCH R6, @RW7+d8	XCH R7, R7	XCH R7, @RW7+d8
+8	XCH R0,@RW0	XCH R0, @RW0+d16	XCH R1,@RW0	XCH R1, @RW0+d16	XCH R2,@RW0	XCH R2, @RW0+d16	XCH R3,@RW0	XCH R3, @RW0+d16	XCH R4,@RW0	XCH R4, @RW0+d16	XCH R5,@RW0	XCH R5, @RW0+d16	XCH R6,@RW0	XCH R6, @RW0+d16	XCH R7,@RW0	XCH R7, @RW0+d16
+9	XCH R0,@RW1	XCH R0, @RW1+d16	XCH R1,@RW1	XCH R1, @RW1+d16	XCH R2,@RW1	XCH R2, @RW1+d16	XCH R3,@RW1	XCH R3, @RW1+d16	XCH R4,@RW1	XCH R4, @RW1+d16	XCH R5,@RW1	XCH R5, @RW1+d16	XCH R6,@RW1	XCH R6, @RW1+d16	XCH R7,@RW1	XCH R7, @RW1+d16
+A	XCH R0,@RW2	XCH R0, W2+d16, A	XCH R1,@RW2	XCH R1, W2+d16, A	XCH R2,@RW2	XCH R2, W2+d16, A	XCH R3,@RW2	XCH R3, W2+d16, A	XCH R4,@RW2	XCH R4, W2+d16, A	XCH R5,@RW2	XCH R5, W2+d16, A	XCH R6,@RW2	XCH R6, W2+d16, A	XCH R7,@RW2	XCH R7, W2+d16, A
+B	XCH R0,@RW3	XCH R0, @RW3+d16	XCH R1,@RW3	XCH R1, @RW3+d16	XCH R2,@RW3	XCH R2, @RW3+d16	XCH R3,@RW3	XCH R3, @RW3+d16	XCH R4,@RW3	XCH R4, @RW3+d16	XCH R5,@RW3	XCH R5, @RW3+d16	XCH R6,@RW3	XCH R6, @RW3+d16	XCH R7,@RW3	XCH R7, @RW3+d16
+C	XCH R0,@RW0+	XCH R0, @RW0+RW7	XCH R1,@RW0+	XCH R1, @RW0+RW7	XCH R2,@RW0+	XCH R2, @RW0+RW7	XCH R3,@RW0+	XCH R3, @RW0+RW7	XCH R4,@RW0+	XCH R4, @RW0+RW7	XCH R5,@RW0+	XCH R5, @RW0+RW7	XCH R6,@RW0+	XCH R6, @RW0+RW7	XCH R7,@RW0+	XCH R7, @RW0+RW7
+D	XCH R0,@RW1+	XCH R0, @RW1+RW7	XCH R1,@RW1+	XCH R1, @RW1+RW7	XCH R2,@RW1+	XCH R2, @RW1+RW7	XCH R3,@RW1+	XCH R3, @RW1+RW7	XCH R4,@RW1+	XCH R4, @RW1+RW7	XCH R5,@RW1+	XCH R5, @RW1+RW7	XCH R6,@RW1+	XCH R6, @RW1+RW7	XCH R7,@RW1+	XCH R7, @RW1+RW7
+E	XCH R0,@RW2+	XCH R0, @PC+d16	XCH R1,@RW2+	XCH R1, @PC+d16	XCH R2,@RW2+	XCH R2, @PC+d16	XCH R3,@RW2+	XCH R3, @PC+d16	XCH R4,@RW2+	XCH R4, @PC+d16	XCH R5,@RW2+	XCH R5, @PC+d16	XCH R6,@RW2+	XCH R6, @PC+d16	XCH R7,@RW2+	XCH R7, @PC+d16
+F	XCH R0,@RW3+	XCH R0, addr16	XCH R1,@RW3+	XCH R1, addr16	XCH R2,@RW3+	XCH R2, addr16	XCH R3,@RW3+	XCH R3, addr16	XCH R4,@RW3+	XCH R4, addr16	XCH R5,@RW3+	XCH R5, addr16	XCH R6,@RW3+	XCH R6, addr16	XCH R7,@RW3+	XCH R7, addr16

C.13 XCHW RWi, ea Instruction Map

Table C-20 lists XCHW RWi, ea instruction map.

■ XCHW RWi, ea Instruction Map

Table C-20 XCHW RWi, ea Instruction (First Byte = 7F$_H$)

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
+0	XCHW RW0, RW0	XCHW RW0, @RW0+d8	XCHW RW1, RW0	XCHW RW1, @RW0+d8	XCHW RW2, RW0	XCHW RW2, @RW0+d8	XCHW RW3, RW0	XCHW RW3, @RW0+d8	XCHW RW4, RW0	XCHW RW4, @RW0+d8	XCHW RW5, RW0	XCHW RW5, @RW0+d8	XCHW RW6, RW0	XCHW RW6, @RW0+d8	XCHW RW7, RW0	XCHW RW7, @RW0+d8
+1	XCHW RW0, RW1	XCHW RW0, @RW1+d8	XCHW RW1, RW1	XCHW RW1, @RW1+d8	XCHW RW2, RW1	XCHW RW2, @RW1+d8	XCHW RW3, RW1	XCHW RW3, @RW1+d8	XCHW RW4, RW1	XCHW RW4, @RW1+d8	XCHW RW5, RW1	XCHW RW5, @RW1+d8	XCHW RW6, RW1	XCHW RW6, @RW1+d8	XCHW RW7, RW1	XCHW RW7, @RW1+d8
+2	XCHW RW0, RW2	XCHW RW0, @RW2+d8	XCHW RW1, RW2	XCHW RW1, @RW2+d8	XCHW RW2, RW2	XCHW RW2, @RW2+d8	XCHW RW3, RW2	XCHW RW3, @RW2+d8	XCHW RW4, RW2	XCHW RW4, @RW2+d8	XCHW RW5, RW2	XCHW RW5, @RW2+d8	XCHW RW6, RW2	XCHW RW6, @RW2+d8	XCHW RW7, RW2	XCHW RW7, @RW2+d8
+3	XCHW RW0, RW3	XCHW RW0, @RW3+d8	XCHW RW1, RW3	XCHW RW1, @RW3+d8	XCHW RW2, RW3	XCHW RW2, @RW3+d8	XCHW RW3, RW3	XCHW RW3, @RW3+d8	XCHW RW4, RW3	XCHW RW4, @RW3+d8	XCHW RW5, RW3	XCHW RW5, @RW3+d8	XCHW RW6, RW3	XCHW RW6, @RW3+d8	XCHW RW7, RW3	XCHW RW7, @RW3+d8
+4	XCHW RW0, RW4	XCHW RW0, @RW4+d8	XCHW RW1, RW4	XCHW RW1, @RW4+d8	XCHW RW2, RW4	XCHW RW2, @RW4+d8	XCHW RW3, RW4	XCHW RW3, @RW4+d8	XCHW RW4, RW4	XCHW RW4, @RW4+d8	XCHW RW5, RW4	XCHW RW5, @RW4+d8	XCHW RW6, RW4	XCHW RW6, @RW4+d8	XCHW RW7, RW4	XCHW RW7, @RW4+d8
+5	XCHW RW0, RW5	XCHW RW0, @RW5+d8	XCHW RW1, RW5	XCHW RW1, @RW5+d8	XCHW RW2, RW5	XCHW RW2, @RW5+d8	XCHW RW3, RW5	XCHW RW3, @RW5+d8	XCHW RW4, RW5	XCHW RW4, @RW5+d8	XCHW RW5, RW5	XCHW RW5, @RW5+d8	XCHW RW6, RW5	XCHW RW6, @RW5+d8	XCHW RW7, RW5	XCHW RW7, @RW5+d8
+6	XCHW RW0, RW6	XCHW RW0, @RW6+d8	XCHW RW1, RW6	XCHW RW1, @RW6+d8	XCHW RW2, RW6	XCHW RW2, @RW6+d8	XCHW RW3, RW6	XCHW RW3, @RW6+d8	XCHW RW4, RW6	XCHW RW4, @RW6+d8	XCHW RW5, RW6	XCHW RW5, @RW6+d8	XCHW RW6, RW6	XCHW RW6, @RW6+d8	XCHW RW7, RW6	XCHW RW7, @RW6+d8
+7	XCHW RW0, RW7	XCHW RW0, @RW7+d8	XCHW RW1, RW7	XCHW RW1, @RW7+d8	XCHW RW2, RW7	XCHW RW2, @RW7+d8	XCHW RW3, RW7	XCHW RW3, @RW7+d8	XCHW RW4, RW7	XCHW RW4, @RW7+d8	XCHW RW5, RW7	XCHW RW5, @RW7+d8	XCHW RW6, RW7	XCHW RW6, @RW7+d8	XCHW RW7, RW7	XCHW RW7, @RW7+d8
+8	XCHW RW0,@RW0	XCHW RW0, @RW0+d16	XCHW RW1,@RW0	XCHW RW1, @RW0+d16	XCHW RW2,@RW0	XCHW RW2, @RW0+d16	XCHW RW3,@RW0	XCHW RW3, @RW0+d16	XCHW RW4,@RW0	XCHW RW4, @RW0+d16	XCHW RW5,@RW0	XCHW RW5, @RW0+d16	XCHW RW6,@RW0	XCHW RW6, @RW0+d16	XCHW RW7,@RW0	XCHW RW7, @RW0+d16
+9	XCHW RW0,@RW1	XCHW RW0, @RW1+d16	XCHW RW1,@RW1	XCHW RW1, @RW1+d16	XCHW RW2,@RW1	XCHW RW2, @RW1+d16	XCHW RW3,@RW1	XCHW RW3, @RW1+d16	XCHW RW4,@RW1	XCHW RW4, @RW1+d16	XCHW RW5,@RW1	XCHW RW5, @RW1+d16	XCHW RW6,@RW1	XCHW RW6, @RW1+d16	XCHW RW7,@RW1	XCHW RW7, @RW1+d16
+A	XCHW RW0,@RW2	XCHW RW0, @RW2+d16	XCHW RW1,@RW2	XCHW RW1, @RW2+d16	XCHW RW2,@RW2	XCHW RW2, @RW2+d16	XCHW RW3,@RW2	XCHW RW3, @RW2+d16	XCHW RW4,@RW2	XCHW RW4, @RW2+d16	XCHW RW5,@RW2	XCHW RW5, @RW2+d16	XCHW RW6,@RW2	XCHW RW6, @RW2+d16	XCHW RW7,@RW2	XCHW RW7, @RW2+d16
+B	XCHW RW0,@RW3	XCHW RW0, @RW3+d16	XCHW RW1,@RW3	XCHW RW1, @RW3+d16	XCHW RW2,@RW3	XCHW RW2, @RW3+d16	XCHW RW3,@RW3	XCHW RW3, @RW3+d16	XCHW RW4,@RW3	XCHW RW4, @RW3+d16	XCHW RW5,@RW3	XCHW RW5, @RW3+d16	XCHW RW6,@RW3	XCHW RW6, @RW3+d16	XCHW RW7,@RW3	XCHW RW7, @RW3+d16
+C	XCHW RW0,@RW0+	XCHW RW0, @RW0+RW7	XCHW RW1,@RW0+	XCHW RW1, @RW0+RW7	XCHW RW2,@RW0+	XCHW RW2, @RW0+RW7	XCHW RW3,@RW0+	XCHW RW3, @RW0+RW7	XCHW RW4,@RW0+	XCHW RW4, @RW0+RW7	XCHW RW5,@RW0+	XCHW RW5, @RW0+RW7	XCHW RW6,@RW0+	XCHW RW6, @RW0+RW7	XCHW RW7,@RW0+	XCHW RW7, @RW0+RW7
+D	XCHW RW0,@RW1+	XCHW RW0, @RW1+RW7	XCHW RW1,@RW1+	XCHW RW1, @RW1+RW7	XCHW RW2,@RW1+	XCHW RW2, @RW1+RW7	XCHW RW3,@RW1+	XCHW RW3, @RW1+RW7	XCHW RW4,@RW1+	XCHW RW4, @RW1+RW7	XCHW RW5,@RW1+	XCHW RW5, @RW1+RW7	XCHW RW6,@RW1+	XCHW RW6, @RW1+RW7	XCHW RW7,@RW1+	XCHW RW7, @RW1+RW7
+E	XCHW RW0,@RW2+	XCHW RW0, @PC+d16	XCHW RW1,@RW2+	XCHW RW1, @PC+d16	XCHW RW2,@RW2+	XCHW RW2, @PC+d16	XCHW RW3,@RW2+	XCHW RW3, @PC+d16	XCHW RW4,@RW2+	XCHW RW4, @PC+d16	XCHW RW5,@RW2+	XCHW RW5, @PC+d16	XCHW RW6,@RW2+	XCHW RW6, @PC+d16	XCHW RW7,@RW2+	XCHW RW7, @PC+d16
+F	XCHW RW0,@RW3+	XCHW RW0, addr16	XCHW RW1,@RW3+	XCHW RW1, addr16	XCHW RW2,@RW3+	XCHW RW2, addr16	XCHW RW3,@RW3+	XCHW RW3, addr16	XCHW RW4,@RW3+	XCHW RW4, addr16	XCHW RW5,@RW3+	XCHW RW5, addr16	XCHW RW6,@RW3+	XCHW RW6, addr16	XCHW RW7,@RW3+	XCHW RW7, addr16

Note:R0 is also used as a barrel shift counter or normalizing instruction counter.

INDEX

The index follows on the next page.
This is listed in alphabetic order.

Index

Numerics

2-byte Instruction
- 2-byte Instruction Map.......................................329

A

A
- Accumulator (A) 18

Accumulator
- Accumulator (A) 18
- ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator).......82
- ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator)..84
- ASR (Arithmetic Shift Byte Data of Accumulator to Right)..102
- ASRL (Arithmetic Shift Long Word Data of Accumulator to Right) 104
- ASRW (Arithmetic Shift Word Data of Accumulator to Right)....................................106, 108
- LSL (Logical Shift Byte Data of Accumulator to Left) 170
- LSLL (Logical Shift Long Word Data of Accumulator to Left) 171
- LSLW (Logical Shift Word Data of Accumulator to Left)..172, 173
- LSR (Logical Shift Byte Data of Accumulator to Right)..174
- LSRL (Logical Shift Long Word Data of Accumulator to Right) 176
- LSRW (Logical Shift Word Data of Accumulator to Right)..178, 179
- MOV (Move Byte Data from Accumulator to Destination) 183
- MOV (Move Byte Data from Source to Accumulator) 181
- MOVB (Move Bit Data from Accumulator to Bit Address)...191
- MOVB (Move Bit Data from Bit Address to Accumulator).....................................189
- MOVL (Move Long Word Data from Accumulator to Destination) 195
- MOVL (Move Long Word Data from Source to Accumulator).....................................194
- MOVN (Move Immediate Nibble Data to Accumulator)......................................196
- MOVW (Move Word Data from Accumulator to Destination) 203
- MOVW (Move Word Data from Source to Accumulator)......................................201
- MOVX (Move Byte Data with Sign Extension from Source to Accumulator)....................... 211
- MUL (Multiply Byte Data of Accumulator)........ 213
- MUL (Multiply Byte Data of Accumulator and Effective Address) 214
- MULU (Multiply Unsigned Byte Data of Accumulator) 217
- MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218
- MULUW (Multiply Unsigned Word Data of Accumulator) 219
- MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220
- MULW (Multiply Word Data of Accumulator) 215
- MULW (Multiply Word Data of Accumulator and Effective Address) 216
- POPW (Pop Word Data of Accumulator from Stack Memory)... 237
- ROLC (Rotate Byte Data of Accumulator with Carry to Left).. 253
- RORC (Rotate Byte Data of Accumulator with Carry to Right).. 255
- SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
- SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269
- SWAP (Swap Byte Data of Accumulator) 277
- SWAPW (Swap Word Data of Accumulator) 278

ADD
- ADD (Add Byte Data of Destination and Source to Destination)... 79

ADDC
- ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator)....... 82
- ADDC (Add Byte Data of AL and AH with Carry to AL) 81

ADDCW
- ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator) 84

ADDDC
- ADDDC (Add Decimal Data of AL and AH with Carry to AL).. 86

ADDL
- ADDL (Add Long Word Data of Destination and Source to Destination) 87

Address Specification
INC (Increment Byte Data (Address Specification)) 153
Addressing
Bank Addressing Mode 9
Direct Addressing 61
Indirect Addressing 63
Linear Addressing Mode...................................... 7
ADDSP
ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer) 89
ADDW
ADDW (Add Word Data of AL and AH to AL).... 90
ADDW (Add Word Data of Destination and Source to Destination).. 91
AND
AND (And Byte Data of Destination and Source to Destination).. 93
AND (And Byte Data of Immediate Data and Condition Code Register)....................... 95
AND (And Byte Data of Destination and Source to Destination).. 93
AND (And Byte Data of Immediate Data and Condition Code Register)....................... 95
ANDL (And Long Word Data of Destination and Source to Destination) 97
ANDL
ANDL (And Long Word Data of Destination and Source to Destination) 97
ANDW
ANDW (And Word Data of AH and AL to AL).... 99
ANDW (And Word Data of Destination and Source to Destination).. 100
Application Procedure
Flowchart of Extended Intelligent I/O Service Application Procedure 51
Arithmetic Shift Byte Data
ASR (Arithmetic Shift Byte Data of Accumulator to Right) 102
Arithmetic Shift Long Word Data
ASRL (Arithmetic Shift Long Word Data of Accumulator to Right)......................... 104
Arithmetic Shift Word Data
ASRW (Arithmetic Shift Word Data of Accumulator to Right).................................... 106, 108
ASR
ASR (Arithmetic Shift Byte Data of Accumulator to Right) 102
ASRL
ASRL (Arithmetic Shift Long Word Data of Accumulator to Right)......................... 104
ASRW
ASRW (Arithmetic Shift Word Data of Accumulator to Right).................................... 106, 108

B

Bank Addressing
Bank Addressing Mode..9
Bank Registers
Bank Registers 28
Bank Select Prefix
Bank Select Prefix 34
Banks
Memory Space Divided into Banks and Values in Each Register Bank...............................11
BAP
Buffer Address Pointer (BAP) 56
Basic Page Map
Basic Page Map...322
BBcc
BBcc (Branch if Bit Condition satisfied).............110
Bcc
Bcc (Branch relative if Condition satisfied).........112
Bit Address
MOVB (Move Bit Data from Accumulator to Bit Address) 191
MOVB (Move Bit Data from Bit Address to Accumulator) 189
Bit Configuration
Interrupt Control Register (ICR) Bit Configuration 52
Bit Data
MOVB (Move Bit Data from Accumulator to Bit Address) 191
MOVB (Move Bit Data from Bit Address to Accumulator) 189
Bit Operation Instruction
Bit Operation Instruction Map 325
Branch
BBcc (Branch if Bit Condition satisfied).............110
Bcc (Branch relative if Condition satisfied).........112
CBNE (Compare Byte Data and Branch if not equal) 120
CWBNE (Compare Word Data and Branch if not Equal)..129
DBNZ (Decrement Byte Data and Branch if not Zero) 131
DWBNZ (Decrement Word Data and Branch if not Zero) 145
SBBS (Set Bit and Branch if Bit Set)..................257
Branch Instruction
Branch Instruction 314
Buffer Address Pointer
Buffer Address Pointer (BAP) 56
Byte
MUL (Multiply Byte Data of Accumulator) 213
Byte Data
AND (And Byte Data of Destination and Source to Destination) 93

AND (And Byte Data of Immediate Data and Condition Code Register) 95
CBNE (Compare Byte Data and Branch if not equal) 120
CMP (Compare Byte Data of Destination and Source) 123
DBNZ (Decrement Byte Data and Branch if not Zero) 131
DEC (Decrement Byte Data) 133
DIV (Divide Word Data by Byte Data)............... 137
EXT (Sign Extend from Byte Data to Word Data) 147
INC (Increment Byte Data (Address Specification)) 153
LSL (Logical Shift Byte Data of Accumulator to Left) 170
LSR (Logical Shift Byte Data of Accumulator to Right).. 174
MOV (Move Byte Data from Accumulator to Destination) 183
MOV (Move Byte Data from AH to Memory) 188
MOV (Move Byte Data from Source to Accumulator) 181
MOV (Move Byte Data from Source to Destination) 186
MOVX (Move Byte Data with Sign Extension from Source to Accumulator) 211
MUL (Multiply Byte Data of Accumulator and Effective Address).............................. 214
MULU (Multiply Unsigned Byte Data of Accumulator)..................................... 217
NEG (Negate Byte Data of Destination) 221
NOT (Not Byte Data of Destination).................. 224
OR (Or Byte Data of Destination and Source to Destination) 228
OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230
ROLC (Rotate Byte Data of Accumulator with Carry to Left) 253
RORC (Rotate Byte Data of Accumulator with Carry to Right) 255
SUB (Subtract Byte Data of Source from Destination to Destination) 264
SUBC (Subtract Byte Data of AL from AH with Carry to AL) 266
SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
SWAP (Swap Byte Data of Accumulator)........... 277
XCH (Exchange Byte Data of Source to Destination) 282
XOR (Exclusive Or Byte Data of Destination and Source to Destination) 286
ZEXT (Zero Extend from Byte Data to Word Data) 293

C

CALL
CALL (Call Subroutine) 114

CALLP
CALLP (Call physical Address) 116

CALLV
CALLV (Call Vectored Subroutine) 118

Carry
ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator)....... 82
ADDC (Add Byte Data of AL and AH with Carry to AL) 81
ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator) 84
ADDDC (Add Decimal Data of AL and AH with Carry to AL).. 86
ROLC (Rotate Byte Data of Accumulator with Carry to Left).. 253
RORC (Rotate Byte Data of Accumulator with Carry to Right).. 255
SUBC (Subtract Byte Data of AL from AH with Carry to AL)... 266
SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269
SUBDC (Subtract Decimal Data of AL from AH with Carry to AL)...................................... 271

CBNE
CBNE (Compare Byte Data and Branch if not equal) 120

CCR
Condition Code Register (CCR) 25

Channel Selection Bits
Extended Intelligent I/O Service Channel Selection Bits (bit15 to bit12 or bit7 to bit4: ICS0 to ICS3).. 53

Character String Operation Instruction
Character String Operation Instruction Map 327

Clear Bit
CLRB (Clear Bit).. 122

CLRB
CLRB (Clear Bit).. 122

CMP
CMP (Compare Byte Data of Destination and Source) 123

CMPL
CMPL (Compare Long Word Data of Destination and Source) 125

CMPW
CMPW (Compare Word Data of Destination and Source) 127
CMR
Common Register Bank Prefix (CMR)................ 36
Common Register Bank Prefix
Common Register Bank Prefix (CMR)................ 36
Compare
CBNE (Compare Byte Data and Branch if not equal) 120
CMP (Compare Byte Data of Destination and Source) 123
CMPL (Compare Long Word Data of Destination and Source) 125
CMPW (Compare Word Data of Destination and Source) 127
CWBNE (Compare Word Data and Branch if not Equal) 129
Condition Code Register
AND (And Byte Data of Immediate Data and Condition Code Register)...................... 95
Condition Code Register (CCR) 25
OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230
Context
JCTX (Jump Context)...................................... 165
CPU
CPU Memory Space.. 6
Features of the F^2MC-16LX CPU Core.................. 2
Overview of the F^2MC-16LX CPU Core................ 2
Create
UNLINK (Unlink and Create New Stack Frame) 279
LINK (Link and create new stack frame)............ 169
CWBNE
CWBNE (Compare Word Data and Branch if not Equal) 129

D

Data Counter
Data Counter (DCT).. 57
Data Operation Instruction
Numeric Data Operation Instructions 307
DBNZ
DBNZ (Decrement Byte Data and Branch if not Zero) 131
DCT
Data Counter (DCT).. 57
DEC
DEC (Decrement Byte Data)............................ 133
Decimal Data
ADDDC (Add Decimal Data of AL and AH with Carry to AL).. 86
SUBDC (Subtract Decimal Data of AL from AH with Carry to AL) 271
DECL
DECL (Decrement Long Word Data) 134
Decrement
DBNZ (Decrement Byte Data and Branch if not Zero) 131
DEC (Decrement Byte Data) 133
DECL (Decrement Long Word Data) 134
DECW (Decrement Word Data)..........................135
DWBNZ (Decrement Word Data and Branch if not Zero) 145
DECW
DECW (Decrement Word Data)..........................135
Dedicated Register
F^2MC-16LX Dedicated Register Types 16
Descriptor
Extended Intelligent I/O Service Descriptor (ISD) 55
Destination
ADD (Add Byte Data of Destination and Source to Destination) 79
ADDL (Add Long Word Data of Destination and Source to Destination)............................87
ADDW (Add Word Data of Destination and Source to Destination) 91
AND (And Byte Data of Destination and Source to Destination) 93
ANDL (And Long Word Data of Destination and Source to Destination)............................97
ANDW (And Word Data of Destination and Source to Destination) 100
CMP (Compare Byte Data of Destination and Source) 123
CMPL (Compare Long Word Data of Destination and Source) 125
CMPW (Compare Word Data of Destination and Source) 127
JMP (Jump Destination Address) 167
JMPP (Jump Destination Physical Address) 168
MOV (Move Byte Data from Accumulator to Destination) 183
MOV (Move Byte Data from Source to Destination) 186
MOV (Move Byte Immediate Data to Destination) 184
MOVEA (Move Effective Address to Destination) 193
MOVL (Move Long Word Data from Accumulator to Destination) 195
MOVW (Move Immediate Word Data to Destination) 205
MOVW (Move Word Data from Accumulator to Destination) 203
MOVW (Move Word Data from Source to Destination) 207

NEG (Negate Byte Data of Destination) 221
NEGW (Negate Word Data of Destination) 222
NOT (Not Byte Data of Destination)..................224
NOTW (Not Word Data of Destination) 226
OR (Or Byte Data of Destination and Source to Destination) 228
ORL (Or Long Word Data of Destination and Source to Destination) 232
ORW (Or Word Data of Destination and Source to Destination) 235
SUB (Subtract Byte Data of Source from Destination to Destination) 264
SUBL (Subtract Long Word Data of Source from Destination to Destination)...................272
SUBW (Subtract Word Data of Source from Destination to Destination)...................274
XCH (Exchange Byte Data of Source to Destination) 282
XCHW (Exchange Word Data of Source to Destination) 284
XOR (Exclusive Or Byte Data of Destination and Source to Destination) 286
XORL (Exclusive Or Long Word Data of Destination and Source to Destination) 288
XORW (Exclusive Or Word Data of Destination and Source to Destination) 291

Detailed Execution Instruction
Reading Detailed Execution Instructions 78
Symbols (abbreviations) Used in Detailed Execution Instructions 70

Device
Sample Configuration of an F^2MC-16LX Device 3

Direct Addressing
Direct Addressing..61

Direct Page Register
Direct Page Register (DPR)................................27

DIV
DIV (Divide Word Data by Byte Data)...............137

Divide
DIV (Divide Word Data by Byte Data)...............137
DIVU (Divide unsigned Word Data by unsigned Byte Data)...141
DIVUW (Divide unsigned Long Word Data by unsigned Word Data)...........................143
DIVW (Divide Long Word Data by Word Data) 139

DIVU
DIVU (Divide unsigned Word Data by unsigned Byte Data)...141

DIVUW
DIVUW (Divide unsigned Long Word Data by unsigned Word Data)...........................143

DIVW
DIVW (Divide Long Word Data by Word Data) 139

DPR
Direct Page Register (DPR) 27

DWBNZ
DWBNZ (Decrement Word Data and Branch if not Zero) 145

E

ea-type Instruction
ea-type Instruction Map................................... 331

Effective Address
ADDC (Add Byte Data of Accumulator and Effective Address with Carry to Accumulator)....... 82
ADDCW (Add Word Data of Accumulator and Effective Address with Carry to Accumulator) 84
Effective Address Field 60, 72, 301
MOVEA (Move Effective Address to Destination) 193
MUL (Multiply Byte Data of Accumulator and Effective Address) 214
MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218
MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220
MULW (Multiply Word Data of Accumulator and Effective Address) 216
SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269

Exception
Exception Occurrence because of the Execution of an Undefined Instruction............................ 58

Exchange
XCH (Exchange Byte Data of Source to Destination) 282
XCHW (Exchange Word Data of Source to Destination)....................................... 284

Exclusive Or
XOR (Exclusive Or Byte Data of Destination and Source to Destination)......................... 286
XORL (Exclusive Or Long Word Data of Destination and Source to Destination).................. 288
XORW (Exclusive Or Word Data of AH and AL to AL) 290
XORW (Exclusive Or Word Data of Destination and Source to Destination)......................... 291

Execution Cycle
Calculating Execution Cycles.............................. 74

Calculating the Number of Execution Cycles...... 302
Execution Cycles 73
Number of Execution Cycles............................ 298

EXT
EXT (Sign Extend from Byte Data to Word Data) 147

Extended Intelligent I/O Service
Extended Intelligent I/O Service Channel Selection Bits (bit15 to bit12 or bit7 to bit4: ICS0 to ICS3) 53
Extended Intelligent I/O Service Enable Bit (bit11 or bit3: ISE) 54
Extended Intelligent I/O Service End Status (bit13,bit 12 or bit5,bit4: S0,S1) 54
Flowchart of Extended Intelligent I/O Service Application Procedure 51
Flowchart of Extended Intelligent I/O Service Operation 50
Overview of Extended Intelligent I/O Service 48
Structure of Extended Intelligent I/O Service 48

Extended Intelligent I/O Service Descriptor
Extended Intelligent I/O Service Descriptor (ISD) 55

Extended Intelligent I/O Service Status Register
Extended Intelligent I/O Service Status Register (ISCS).. 56

EXTW
EXTW (Sign Extend from Word Data to Long Word Data)... 148

F

F²MC-16LX
F²MC-16LX Dedicated Register Types................ 16
Features of the F²MC-16LX CPU Core.................. 2
Overview of the F²MC-16LX CPU Core................ 2
Sample Configuration of an F²MC-16LX Device.... 3

Fill
FILS (Fill String Byte) 149
FILSW (Fill String Word)................................ 151

FILS
FILS (Fill String Byte) 149

FILSW
FILSW (Fill String Word)................................ 151

Flag Change Inhibit Prefix Code
Flag Change Inhibit Prefix Code (NCC)............... 37

G

General-purpose Registers
Calling General-purpose Registers in RAM.......... 31

H

Hardware Interrupt
Hardware Interrupt Operation Flow 44

I

I/O Register Address Pointer
I/O Register Address Pointer (IOA)......................57

ICR
Functions of Interrupt Control Registers (ICR0 to ICR15) 52
Interrupt Control Register (ICR) Bit Configuration 52

ILM
Interrupt Level Mask Register (ILM)....................23

Immediate Data
ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer) 89
AND (And Byte Data of Immediate Data and Condition Code Register) 95
MOV (Move Byte Immediate Data to Destination) 184
OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230

Immediate Nibble Data
MOVN (Move Immediate Nibble Data to Accumulator) 196

Immediate Word Data
MOVW (Move Immediate Word Data to Destination) 205
MOVW (Move Immediate Word Data to io) 209

INC
INC (Increment Byte Data (Address Specification)) 153

INCL
INCL (Increment Long Word Data)....................154

Increment
INC (Increment Byte Data (Address Specification)) 153
INCL (Increment Long Word Data)....................154
INCW (Increment Word Data)...........................155

INCW
INCW (Increment Word Data)...........................155

Indirect Addressing
Indirect Addressing..63

Inherent Register
PUSHW (Push Word Data of Inherent Register to Stack Memory)....................................244

Instruction
2-byte Instruction Map 329
Bit Operation Instruction Map 325
Branch Instruction 314
Character String Operation Instruction Map 327
ea-type Instruction Map....................................331
Exception Occurrence because of the Execution of an Undefined Instruction.............................58
Explanation of the Items Used in the Instruction Lists 297

Explanation of the Symbols Used in the Instruction Lists....299
Instruction Overview....68
Logical Operation Instruction....311
MOV ea, Ri Instruction Map....346
MOV Ri, ea Instruction Map....343
MOVEA RWi, ea Instruction Map....341
MOVW ea, RWi Instruction Map....348
MOVW RWi, ea Instruction Map....345
Numeric Data Operation Instructions....307
Other Instructions....316
Reading Detailed Execution Instructions....78
Relationships between Instructions Rejecting Interrupt Requests and Prefix Codes....38
Shift Instruction....313
Structure of the Instruction Map....320
Symbols (abbreviations) Used in Detailed Execution Instructions....70
Transfer Instruction....305
XCH Ri, ea Instruction Map....351
XCHW RWi, ea Instruction Map....352

Instruction Map
Structure of the Instruction Map....320

INT
INT (Software Interrupt (Vector Specification))....159
INT (Software Interrupt)....157

INT9
INT9 (Software Interrupt)....161

Interrupt
Hardware Interrupt Operation Flow....44
INT (Software Interrupt (Vector Specification))....159
INT (Software Interrupt)....157
INT9 (Software Interrupt)....161
Interrupt Handling....42
Interrupt Handling Flowchart....45
Interrupt Vectors....47
INTP (Software Interrupt)....163
Relationships between Instructions Rejecting Interrupt Requests and Prefix Codes....38
RETI (Return from Interrupt)....249

Interrupt Control Register
Functions of Interrupt Control Registers (ICR0 to ICR15)....52
Interrupt Control Register (ICR) Bit Configuration....52

Interrupt Handling
Interrupt Handling....42
Interrupt Handling Flowchart....45

Interrupt Level Mask Register
Interrupt Level Mask Register (ILM)....23

Interrupt Level Setting Bits
Interrupt Level Setting Bits (bit10 to bit8 or bit2 to bit0: IL2 to IL0)....54

Interrupt Vectors
Interrupt Vectors....47

INTP
INTP (Software Interrupt)....163

IOA
I/O Register Address Pointer (IOA)....57

ISCS
Extended Intelligent I/O Service Status Register (ISCS)....56

ISD
Extended Intelligent I/O Service Descriptor (ISD)....55

J

JCTX
JCTX (Jump Context)....165

JMP
JMP (Jump Destination Address)....167

JMPP
JMPP (Jump Destination Physical Address)....168

Jump
JCTX (Jump Context)....165
JMP (Jump Destination Address)....167
JMPP (Jump Destination Physical Address)....168

L

Left
LSL (Logical Shift Byte Data of Accumulator to Left)....170
LSLL (Logical Shift Long Word Data of Accumulator to Left)....171
LSLW (Logical Shift Word Data of Accumulator to Left)....172, 173
ROLC (Rotate Byte Data of Accumulator with Carry to Left)....253

Linear Addressing
Linear Addressing Mode....7

Linear Addressing Mode
Linear Addressing Mode....7

LINK
LINK (Link and create new stack frame)....169

Logical Operation Instruction
Logical Operation Instruction....311

Logical Shift
LSL (Logical Shift Byte Data of Accumulator to Left)....170
LSLL (Logical Shift Long Word Data of Accumulator to Left)....171
LSLW (Logical Shift Word Data of Accumulator to Left)....172, 173
LSR (Logical Shift Byte Data of Accumulator to Right)....174

LSRL (Logical Shift Long Word Data of Accumulator to Right).. 176
LSRW (Logical Shift Word Data of Accumulator to Right) 178, 179

Long Word
ANDL (And Long Word Data of Destination and Source to Destination) 97
NRML (NORMALIZE Long Word) 227

Long Word Data
ADDL (Add Long Word Data of Destination and Source to Destination) 87
DECL (Decrement Long Word Data)................. 134
DIVW (Divide Long Word Data by Word Data) 139
EXTW (Sign Extend from Word Data to Long Word Data).. 148
INCL (Increment Long Word Data) 154
LSLL (Logical Shift Long Word Data of Accumulator to Left).. 171
LSRL (Logical Shift Long Word Data of Accumulator to Right).. 176
MOVL (Move Long Word Data from Accumulator to Destination)....................................... 195
MOVL (Move Long Word Data from Source to Accumulator)..................................... 194
ORL (Or Long Word Data of Destination and Source to Destination) 232
SUBL (Subtract Long Word Data of Source from Destination to Destination).................. 272
XORL (Exclusive Or Long Word Data of Destination and Source to Destination) 288
ZEXTW (Zero Extend from Word Data to Long Word Data).. 294

LSL
LSL (Logical Shift Byte Data of Accumulator to Left) 170

LSLL
LSLL (Logical Shift Long Word Data of Accumulator to Left).. 171

LSLW
LSLW (Logical Shift Word Data of Accumulator to Left) 172, 173

LSR
LSR (Logical Shift Byte Data of Accumulator to Right) 174

LSRL
LSRL (Logical Shift Long Word Data of Accumulator to Right).. 176

LSRW
LSRW (Logical Shift Word Data of Accumulator to Right) 178, 179

M

Memory Space
CPU Memory Space.. 6
Memory Space Divided into Banks and Values in Each Register Bank...............................11
Multi-byte Data Layout in a Memory Space 12

Mode
Bank Addressing Mode 9
Linear Addressing Mode 7

MOV
MOV (Move Byte Data from Accumulator to Destination) 183
MOV (Move Byte Data from AH to Memory).....188
MOV (Move Byte Data from Source to Accumulator) 181
MOV (Move Byte Data from Source to Destination) 186
MOV (Move Byte Immediate Data to Destination) 184

MOV ea, Ri Instruction
MOV ea, Ri Instruction Map 346

MOV Ri, ea Instruction
MOV Ri, ea Instruction Map 343

MOVB
MOVB (Move Bit Data from Accumulator to Bit Address) 191
MOVB (Move Bit Data from Bit Address to Accumulator) 189

Move
MOV (Move Byte Data from Accumulator to Destination) 183
MOV (Move Byte Data from AH to Memory).....188
MOV (Move Byte Data from Source to Accumulator) 181
MOV (Move Byte Data from Source to Destination) 186
MOV (Move Byte Immediate Data to Destination) 184
MOVB (Move Bit Data from Accumulator to Bit Address) 191
MOVB (Move Bit Data from Bit Address to Accumulator) 189
MOVEA (Move Effective Address to Destination) 193
MOVL (Move Long Word Data from Accumulator to Destination) 195
MOVL (Move Long Word Data from Source to Accumulator) 194
MOVN (Move Immediate Nibble Data to Accumulator) 196
MOVS (Move String Byte)................................197
MOVSW (Move String Word)...........................199
MOVW (Move Immediate Word Data to Destination) 205
MOVW (Move Immediate Word Data to io) 209
MOVW (Move Word Data from Accumulator to Destination) 203
MOVW (Move Word Data from AH to Memory) 210

MOVW (Move Word Data from Source to Accumulator) 201
MOVW (Move Word Data from Source to Destination) 207
MOVX (Move Byte Data with Sign Extension from Source to Accumulator) 211

MOVEA
MOVEA (Move Effective Address to Destination) 193

MOVEA RWi, ea Instruction
MOVEA RWi, ea Instruction Map 341

MOVL
MOVL (Move Long Word Data from Accumulator to Destination) 195
MOVL (Move Long Word Data from Source to Accumulator) 194

MOVN
MOVN (Move Immediate Nibble Data to Accumulator) 196

MOVS
MOVS (Move String Byte) 197

MOVSW
MOVSW (Move String Word) 199

MOVW
MOVW (Move Immediate Word Data to Destination) 205
MOVW (Move Immediate Word Data to io) 209
MOVW (Move Word Data from Accumulator to Destination) 203
MOVW (Move Word Data from AH to Memory) 210
MOVW (Move Word Data from Source to Accumulator) 201
MOVW (Move Word Data from Source to Destination) 207

MOVW ea, RWi Instruction
MOVW ea, RWi Instruction Map 348

MOVW RWi, ea Instruction
MOVW RWi, ea Instruction Map 345

MOVX
MOVX (Move Byte Data with Sign Extension from Source to Accumulator) 211

MUL
MUL (Multiply Byte Data of Accumulator) 213
MUL (Multiply Byte Data of Accumulator and Effective Address) 214

Multi-byte Data
Access to Multi-byte Data 13
Multi-byte Data Layout in a Memory Space 12

Multiply
MUL (Multiply Byte Data of Accumulator) 213
MUL (Multiply Byte Data of Accumulator and Effective Address) 214
MULU (Multiply Unsigned Byte Data of Accumulator) 217
MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218
MULUW (Multiply Unsigned Word Data of Accumulator) 219
MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220
MULW (Multiply Word Data of Accumulator) 215
MULW (Multiply Word Data of Accumulator and Effective Address) 216

MULU
MULU (Multiply Unsigned Byte Data of Accumulator) 217
MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218

MULUW
MULUW (Multiply Unsigned Word Data of Accumulator) 219
MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220

MULW
MULW (Multiply Word Data of Accumulator) 215
MULW (Multiply Word Data of Accumulator and Effective Address) 216

N

NCC
Flag Change Inhibit Prefix Code (NCC) 37

NEG
NEG (Negate Byte Data of Destination) 221

Negate
NEG (Negate Byte Data of Destination) 221
NEGW (Negate Word Data of Destination) 222

NEGW
NEGW (Negate Word Data of Destination) 222

No Operation
NOP (No Operation) 223

NOP
NOP (No Operation) 223

NORMALIZE
NRML (NORMALIZE Long Word) 227

NOT
NOT (Not Byte Data of Destination) 224

NOTW
NOTW (Not Word Data of Destination) 226

NRML
NRML (NORMALIZE Long Word) 227

O

OR
- OR (Or Byte Data of Destination and Source to Destination) 228
- OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230

Or
- OR (Or Byte Data of Destination and Source to Destination) 228
- OR (Or Byte Data of Immediate Data and Condition Code Register to Condition Code Register) 230
- ORL (Or Long Word Data of Destination and Source to Destination) 232
- ORW (Or Word Data of AH and AL to AL) 234
- ORW (Or Word Data of Destination and Source to Destination) 235

ORL
- ORL (Or Long Word Data of Destination and Source to Destination) 232

ORW
- ORW (Or Word Data of AH and AL to AL) 234
- ORW (Or Word Data of Destination and Source to Destination) 235

P

Page Map
- Basic Page Map 322

PC
- Program Counter (PC) 26

Physical Address
- JMPP (Jump Destination Physical Address) 168
- RETP (Return from Physical Address) 251

physical Address
- CALLP (Call physical Address) 116

Pop
- POPW (Pop Registers from Stack Memory) 242
- POPW (Pop Word Data of Accumulator from Stack Memory) 237
- POPW (Pop Word Data of AH from Stack Memory) 239
- POPW (Pop Word Data of Program Status from Stack Memory) 240

POPW
- POPW (Pop Registers from Stack Memory) 242
- POPW (Pop Word Data of Accumulator from Stack Memory) 237
- POPW (Pop Word Data of AH from Stack Memory) 239
- POPW (Pop Word Data of Program Status from Stack Memory) 240

Prefix
- Bank Select Prefix 34
- Common Register Bank Prefix (CMR) 36
- Flag Change Inhibit Prefix Code (NCC) 37
- If Two or More Prefix Codes Appear in Succession 39
- Relationships between Instructions Rejecting Interrupt Requests and Prefix Codes 38

Processor Status
- Processor Status (PS) 22

Program Counter
- Program Counter (PC) 26

Program Status
- POPW (Pop Word Data of Program Status from Stack Memory) 240

PS
- Processor Status (PS) 22

Push
- PUSHW (Push Registers to Stack Memory) 246
- PUSHW (Push Word Data of Inherent Register to Stack Memory) 244

PUSHW
- PUSHW (Push Registers to Stack Memory) 246
- PUSHW (Push Word Data of Inherent Register to Stack Memory) 244

R

RAM
- Calling General-purpose Registers in RAM 31
- Register Banks in RAM 30

Register
- AND (And Byte Data of Immediate Data and Condition Code Register) 95
- Condition Code Register (CCR) 25
- Direct Page Register (DPR) 27
- Extended Intelligent I/O Service Status Register (ISCS) 56
- Functions of Interrupt Control Registers (ICR0 to ICR15) 52
- Interrupt Control Register (ICR) Bit Configuration 52
- Interrupt Level Mask Register (ILM) 23

Register Bank
- Memory Space Divided into Banks and Values in Each Register Bank 11

Register Bank Pointer
- Register Bank Pointer (RP) 24

Register Banks
- Register Banks in RAM 30

RET
- RET (Return from Subroutine) 248

RETI
- RETI (Return from Interrupt) 249

RETP
- RETP (Return from Physical Address) 251

Return
- RET (Return from Subroutine) 248

RETI (Return from Interrupt) 249
RETP (Return from Physical Address) 251

Right
ASR (Arithmetic Shift Byte Data of Accumulator to Right)..102
ASRL (Arithmetic Shift Long Word Data of Accumulator to Right) 104
ASRW (Arithmetic Shift Word Data of Accumulator to Right)....................................106, 108
LSR (Logical Shift Byte Data of Accumulator to Right)..174
LSRL (Logical Shift Long Word Data of Accumulator to Right)..176
LSRW (Logical Shift Word Data of Accumulator to Right)..178, 179
RORC (Rotate Byte Data of Accumulator with Carry to Right)..255

ROLC
ROLC (Rotate Byte Data of Accumulator with Carry to Left)...253

RORC
RORC (Rotate Byte Data of Accumulator with Carry to Right)..255

Rotate
ROLC (Rotate Byte Data of Accumulator with Carry to Left)...253
RORC (Rotate Byte Data of Accumulator with Carry to Right)..255

RP
Register Bank Pointer (RP) 24

S

Sample Configuration
Sample Configuration of an F^2MC-16LX Device 3

SBBS
SBBS (Set Bit and Branch if Bit Set)..................257

Scan
SCEQ (Scan String Byte Until Equal) 259
SCWEQ (Scan String Word until Equal) 261

SCEQ
SCEQ (Scan String Byte Until Equal) 259

SCWEQ
SCWEQ (Scan String Word until Equal) 261

Set
SBBS (Set Bit and Branch if Bit Set)..................257
SETB (Set Bit)..263

SETB
SETB (Set Bit)..263

Shift Instruction
Shift Instruction 313

Sign
EXT (Sign Extend from Byte Data to Word Data) 147
EXTW (Sign Extend from Word Data to Long Word Data) 148

Sign Extension
MOVX (Move Byte Data with Sign Extension from Source to Accumulator)....................... 211

Software Interrupt
INT (Software Interrupt (Vector Specification)) 159
INT (Software Interrupt)................................... 157
INT9 (Software Interrupt)................................ 161
INTP (Software Interrupt)................................ 163

Source
ADD (Add Byte Data of Destination and Source to Destination)... 79
ADDL (Add Long Word Data of Destination and Source to Destination) 87
ADDW (Add Word Data of Destination and Source to Destination)... 91
AND (And Byte Data of Destination and Source to Destination)... 93
ANDL (And Long Word Data of Destination and Source to Destination) 97
ANDW (And Word Data of Destination and Source to Destination)....................................... 100
CMP (Compare Byte Data of Destination and Source) 123
CMPL (Compare Long Word Data of Destination and Source) 125
CMPW (Compare Word Data of Destination and Source) 127
MOV (Move Byte Data from Source to Accumulator) 181
MOV (Move Byte Data from Source to Destination) 186
MOVL (Move Long Word Data from Source to Accumulator) 194
MOVW (Move Word Data from Source to Accumulator) 201
MOVW (Move Word Data from Source to Destination)....................................... 207
MOVX (Move Byte Data with Sign Extension from Source to Accumulator)....................... 211
OR (Or Byte Data of Destination and Source to Destination)....................................... 228
ORL (Or Long Word Data of Destination and Source to Destination)................................... 232
ORW (Or Word Data of Destination and Source to Destination)....................................... 235
SUB (Subtract Byte Data of Source from Destination to Destination)................................... 264
SUBL (Subtract Long Word Data of Source from Destination to Destination).................. 272
SUBW (Subtract Word Data of Source from Destination to Destination).................. 274
XCH (Exchange Byte Data of Source to Destination) 282

XCHW (Exchange Word Data of Source to Destination).. 284
XOR (Exclusive Or Byte Data of Destination and Source to Destination) 286
XORL (Exclusive Or Long Word Data of Destination and Source to Destination) 288
XORW (Exclusive Or Word Data of Destination and Source to Destination) 291

SSP
User Stack Pointer (USP) and System Stack Pointer (SSP) 20

Stack
POPW (Pop Registers from Stack Memory) 242
POPW (Pop Word Data of Accumulator from Stack Memory) 237
POPW (Pop Word Data of AH from Stack Memory) 239
POPW (Pop Word Data of Program Status from Stack Memory) 240
PUSHW (Push Registers to Stack Memory) 246
PUSHW (Push Word Data of Inherent Register to Stack Memory) 244

Stack Frame
UNLINK (Unlink and Create New Stack Frame) 279
LINK (Link and create new stack frame)............ 169

Stack Pointer
ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer) 89

String Byte
FILS (Fill String Byte) 149
MOVS (Move String Byte) 197
SCEQ (Scan String Byte Until Equal) 259

String Word
FILSW (Fill String Word)................................. 151
MOVSW (Move String Word) 199
SCWEQ (Scan String Word until Equal) 261

SUB
SUB (Subtract Byte Data of Source from Destination to Destination) 264

SUBC
SUBC (Subtract Byte Data of AL from AH with Carry to AL).. 266
SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267

SUBCW
SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator)...................................... 269

SUBDC
SUBDC (Subtract Decimal Data of AL from AH with Carry to AL)....................................... 271

SUBL
SUBL (Subtract Long Word Data of Source from Destination to Destination) 272

Subroutine
RET (Return from Subroutine)...........................248

Subtract
SUB (Subtract Byte Data of Source from Destination to Destination).....................................264
SUBC (Subtract Byte Data of AL from AH with Carry to AL) 266
SUBC (Subtract Byte Data of Effective Address from Accumulator with Carry to Accumulator) 267
SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269
SUBDC (Subtract Decimal Data of AL from AH with Carry to AL) 271
SUBL (Subtract Long Word Data of Source from Destination to Destination) 272
SUBW (Subtract Word Data of AL from AH to AL) 276
SUBW (Subtract Word Data of Source from Destination to Destination) 274

SUBW
SUBW (Subtract Word Data of AL from AH to AL) 276
SUBW (Subtract Word Data of Source from Destination to Destination) 274

SWAP
SWAP (Swap Byte Data of Accumulator)...........277

SWAPW
SWAPW (Swap Word Data of Accumulator) 278

Symbol
Explanation of the Symbols Used in the Instruction Lists 299
Symbols (abbreviations) Used in Detailed Execution Instructions 70

System Stack Pointer
User Stack Pointer (USP) and System Stack Pointer (SSP)...20

T

Transfer Instruction
Transfer Instruction 305

U

Undefined Instruction
Exception Occurrence because of the Execution of an Undefined Instruction.............................58

UNLINK
UNLINK (Unlink and Create New Stack Frame) 279

Unsigned Byte Data
MULU (Multiply Unsigned Byte Data of Accumulator and Effective Address) 218
unsigned Byte Data
DIVU (Divide unsigned Word Data by unsigned Byte Data) 141
unsigned Long Word Data
DIVUW (Divide unsigned Long Word Data by unsigned Word Data) 143
Unsigned Word Data
MULUW (Multiply Unsigned Word Data of Accumulator) 219
MULUW (Multiply Unsigned Word Data of Accumulator and Effective Address) 220
unsigned Word Data
DIVU (Divide unsigned Word Data by unsigned Byte Data) 141
DIVUW (Divide unsigned Long Word Data by unsigned Word Data) 143
User Stack Pointer
User Stack Pointer (USP) and System Stack Pointer (SSP) 20
USP
User Stack Pointer (USP) and System Stack Pointer (SSP) 20

V

Vector Specification
INT (Software Interrupt (Vector Specification)) 159
Vectored Subroutine
CALLV (Call Vectored Subroutine) 118

W

Wait
WBTc (Wait until Bit Condition Satisfied) 280
WBTc
WBTc (Wait until Bit Condition Satisfied) 280
Word Data
ADDSP (Add Word Data of Stack Pointer and Immediate Data to Stack Pointer) 89
ADDW (Add Word Data of AL and AH to AL) 90
ADDW (Add Word Data of Destination and Source to Destination) 91
ANDW (And Word Data of AH and AL to AL) 99
ANDW (And Word Data of Destination and Source to Destination) 100
CMPL (Compare Long Word Data of Destination and Source) 125
CMPW (Compare Word Data of Destination and Source) 127
CWBNE (Compare Word Data and Branch if not Equal) 129
DECW (Decrement Word Data) 135
DIV (Divide Word Data by Byte Data) 137
DIVW (Divide Long Word Data by Word Data) 139
DWBNZ (Decrement Word Data and Branch if not Zero) 145
EXT (Sign Extend from Byte Data to Word Data) 147
EXTW (Sign Extend from Word Data to Long Word Data) 148
INCW (Increment Word Data) 155
LSLW (Logical Shift Word Data of Accumulator to Left) 172, 173
LSRW (Logical Shift Word Data of Accumulator to Right) 178, 179
MOVW (Move Word Data from Accumulator to Destination) 203
MOVW (Move Word Data from AH to Memory) 210
MOVW (Move Word Data from Source to Accumulator) 201
MOVW (Move Word Data from Source to Destination) 207
MULW (Multiply Word Data of Accumulator) 215
MULW (Multiply Word Data of Accumulator and Effective Address) 216
NEGW (Negate Word Data of Destination) 222
NOTW (Not Word Data of Destination) 226
ORW (Or Word Data of AH and AL to AL) 234
ORW (Or Word Data of Destination and Source to Destination) 235
POPW (Pop Word Data of Accumulator from Stack Memory) 237
POPW (Pop Word Data of AH from Stack Memory) 239
POPW (Pop Word Data of Program Status from Stack Memory) 240
PUSHW (Push Word Data of Inherent Register to Stack Memory) 244
SUBCW (Subtract Word Data of Effective Address from Accumulator with Carry to Accumulator) 269
SUBW (Subtract Word Data of AL from AH to AL) 276
SUBW (Subtract Word Data of Source from Destination to Destination) 274
SWAPW (Swap Word Data of Accumulator) 278
XCHW (Exchange Word Data of Source to Destination) 284
XORW (Exclusive Or Word Data of AH and AL to AL) 290
XORW (Exclusive Or Word Data of Destination and Source to Destination) 291
ZEXT (Zero Extend from Byte Data to Word Data) 293

ZEXTW (Zero Extend from Word Data to Long Word Data).. 294

X

XCH
XCH (Exchange Byte Data of Source to Destination) 282
XCH Ri, ea Instruction
XCH Ri, ea Instruction Map............................. 351
XCHW
XCHW (Exchange Word Data of Source to Destination)....................................... 284
XCHW RWi, ea Instruction
XCHW RWi, ea Instruction Map....................... 352
XOR
XOR (Exclusive Or Byte Data of Destination and Source to Destination) 286
XORL
XORL (Exclusive Or Long Word Data of Destination and Source to Destination)...................288
XORW
XORW (Exclusive Or Word Data of AH and AL to AL) 290
XORW (Exclusive Or Word Data of Destination and Source to Destination)..........................291

Z

Zero Extend
ZEXT (Zero Extend from Byte Data to Word Data)..293
ZEXTW (Zero Extend from Word Data to Long Word Data) 294
ZEXT
ZEXT (Zero Extend from Byte Data to Word Data)..293
ZEXTW
ZEXTW (Zero Extend from Word Data to Long Word Data) 294

CM44-00201-3E

FUJITSU MICROELECTRONICS • CONTROLLER MANUAL

F^{2}MC-16LX

16-BIT MICROCONTROLLER

PROGRAMMING MANUAL

May 2008 the third edition

Published **FUJITSU MICROELECTRONICS LIMITED**

Edited Strategic Business Development Dept.