To our customers,

Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010
Renesas Electronics Corporation

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

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

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

User's Manual

16

M16C/60, M16C/20, M16C/Tiny Series

Software Manual

RENESAS 16-BIT SINGLE-CHIP MICROCOMPUTER

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev. 4.00   2004.01

Keep safety first in your circuit designs!

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

Notes regarding these materials

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

Using This Manual

This manual is written for the M16C/60, M16C/20, M16C/Tiny series software. This manual can be used for all types of microcomputers having the M16C/60 series CPU core.

The reader of this manual is expected to have the basic knowledge of electric and logic circuits and microcomputers.

This manual consists of five chapters. The following lists the chapters and sections to be referred to when you want to know details on some specific subject.

- To understand the outline of the M16C/60, M16C/20, M16C/Tiny series and its features ....**Chapter 1, “Overview”**
- To understand the operation of each addressing mode ....**Chapter 2, “Addressing Modes”**
- To understand instruction functions
(Syntax, operation, function, selectable src/dest (label), flag changes, description example, related instructions).. **Chapter 3, “Functions”**
- To understand instruction code and cycles......... **Chapter 4, “Instruction Code/Number of Cycles”**

This manual also contains quick references immediately after the Table of Contents. These quick references will help you quickly find the pages for the functions or instruction code/ number of cycles you want to know.

- To find pages from mnemonic.................................**Quick Reference in Alphabetic Order**
- To find pages from function and mnemonic .... **Quick Reference by Function**
- To find pages from mnemonic and addressing................**Quick Reference by Addressing**

A table of symbols, a glossary, and an index are appended at the end of this manual.

M16C Family Documents

The following documents were prepared for the M16C family. (1)

Document	Contents
Short Sheet	Hardware overview
Data Sheet	Hardware overview and electrical characteristics
Hardware Manual	Hardware specifications (pin assignments, memory maps, peripheral specifications, electrical characteristics, timing charts)
Software Manual	Detailed description of assembly instructions and microcomputer performance of each instruction
Application Note	• Application examples of peripheral functions • Sample programs • Introduction to the basic functions in the M16C family • Programming method with Assembly and C languages
Technical Update	Preliminary report about the specification of a product, a document, etc.

NOTES :

1. Before using this material, please visit the our website to confirm that this is the most current document available.

Table of Contents

Chapter 1 Overview

1.1 Features of M16C/60, M16C/20, M16C/Tiny series 2
1.1.1 Features of M16C/60, M16C/20, M16C/Tiny series 2
1.1.2 Speed performance 2

1.2 Address Space 3

1.3 Register Configuration 4
1.3.1 Data registers (R0, R0H, R0L, R1, R1H, R1L, R2, and R3) 4
1.3.2 Address registers (A0 and A1) 5
1.3.3 Frame base register (FB) 5
1.3.4 Program counter (PC) 5
1.3.5 Interrupt table register (INTB) 5
1.3.6 User stack pointer (USP) and interrupt stack pointer (ISP) 5
1.3.7 Static base register (SB) 5
1.3.8 Flag register (FLG) 5

1.4 Flag Register(FLG) 6
1.4.1 Bit 0: Carry flag (C flag) 6
1.4.2 Bit 1: Debug flag (D flag) 6
1.4.3 Bit 2: Zero flag (Z flag) 6
1.4.4 Bit 3: Sign flag (S flag) 6
1.4.5 Bit 4: Register bank select flag (B flag) 6
1.4.6 Bit 5: Overflow flag (O flag) 6
1.4.7 Bit 6: Interrupt enable flag (I flag) 6
1.4.8 Bit 7: Stack pointer select flag (U flag) 6
1.4.9 Bits 8-11: Reserved area 6
1.4.10 Bits 12-14: Processor interrupt priority level (IPL) 7
1.4.11 Bit 15: Reserved area 7

1.5 Register Bank 8

1.6 Internal State after Reset is Cleared 9

1.7 Data Types 10
1.7.1 Integer 10
1.7.2 Decimal 11
1.7.3 Bits 12
1.7.4 String 15

1.8 Data Arrangement 16
1.8.1 Data Arrangement in Register 16
1.8.2 Data Arrangement in Memory 17

1.9 Instruction Format 18
1.9.1 Generic format (:G) 18
1.9.2 Quick format (:Q) 18
1.9.3 Short format (:S) 18
1.9.4 Zero format (:Z) 18

1.10 Vector Table 19
1.10.1 Fixed Vector Table 19
1.10.2 Variable Vector Table 20

Chapter 2 Addressing Modes

2.1 Addressing Modes 22
2.1.1 General instruction addressing 22
2.1.2 Special instruction addressing 22
2.1.3 Bit instruction addressing 22
2.2 Guide to This Chapter 23
2.3 General Instruction Addressing 24
2.4 Specific Instruction Addressing 27
2.5 Bit Instruction Addressing 30

Chapter 3 Functions

3.1 Guide to This Chapter 34
3.2 Functions 39

Chapter 4 Instruction Code/Number of Cycles

4.1 Guide to This Chapter 138
4.2 Instruction Code/Number of Cycles 140

Chapter 5 Interrupt

5.1 Outline of Interrupt 248
5.1.1 Types of Interrupts 248
5.1.2 Software Interrupts 249
5.1.3 Hardware Interrupts 250
5.2 Interrupt Control 251
5.2.1 Interrupt Enable Flag (I Flag) 251
5.2.2 Interrupt Request Bit 251
5.2.3 Interrupt Priority Level Select Bit and Processor Interrupt Priority Level (IPL) 252
5.2.4 Rewrite the Interrupt Control Register 253
5.3 Interrupt Sequence 254
5.3.1 Interrupt Response Time 255
5.3.2 Changes of IPL When Interrupt Request Acknowledged 256
5.3.3 Saving Registers 256
5.4 Return from Interrupt Routine 258
5.5 Interrupt Priority 259
5.6 Multiple Interrupts 260
5.7 Precautions for Interrupts 262
5.7.1 Reading address 0000016 262
5.7.2 Setting the SP 262
5.7.3 Rewrite the Interrupt Control Register 262

Chapter 6 Calculation Number of Cycles

6.1 Instruction queue buffer 266

Quick Reference in Alphabetic Order

Mnemonic	See page for function	See page for instruction code /number of cycles	Mnemonic	See page for function	See page for instruction code /number of cycles
ABS	39	140	DIVU	68	173
ADC	40	140	DIVX	69	174
ADCF	41	142	DSBB	70	175
ADD	42	142	DSUB	71	177
ADJNZ	44	148	ENTER	72	179
AND	45	149	EXITD	73	180
BAND	47	152	EXTS	74	180
BCLR	48	152	FCLR	75	181
BM*Cnd*	49	154	FSET	76	182
BMEQ/Z	49	154	INC	77	182
BMGE	49	154	INT	78	183
BMGEU/C	49	154	INTO	79	184
BMGT	49	154	J*Cnd*	80	184
BMGTU	49	154	JEQ/Z	80	184
BMLE	49	154	JGE	80	184
BMLEU	49	154	JGEU/C	80	184
BMLT	49	154	JGT	80	184
BMLTU/NC	49	154	JGTU	80	184
BMN	49	154	JLE	80	184
BMNE/NZ	49	154	JLEU	80	184
BMNO	49	154	JLT	80	184
BMO	49	154	JLTU/NC	80	184
BMPZ	49	154	JN	80	184
BNAND	50	155	JNE/NZ	80	184
BNOR	51	156	JNO	80	184
BNOT	52	156	JO	80	184
BNTST	53	157	JPZ	80	184
BNXOR	54	158	JMP	81	185
BOR	55	158	JMPI	82	187
BRK	56	159	JMPS	83	188
BSET	57	159	JSR	84	189
BTST	58	160	JSRI	85	190
BTSTC	59	161	JSRS	86	191
BTSTS	60	162	LDC	87	191
BXOR	61	162	LDCTX	88	192
CMP	62	163	LDE	89	193
DADC	64	167	LDINTB	90	194
DADD	65	169	LDIPL	91	195
DEC	66	171	MOV	92	195
DIV	67	172	MOVA	94	202

Quick Reference in Alphabetic Order

Mnemonic	See page for function	See page for instruction code /number of cycles	Mnemonic	See page for function	See page for instruction code /number of cycles
MOV*Dir*	95	203	ROT	114	222
MOVHH	95	203	RTS	115	223
MOVHL	95	203	SBB	116	224
MOVLH	95	203	SBJNZ	117	226
MOVLL	95	203	SHA	118	227
MUL	96	205	SHL	119	230
MULU	97	207	SMOVB	120	232
NEG	98	209	SMOVF	121	233
NOP	99	209	SSTR	122	233
NOT	100	210	STC	123	234
OR	101	211	STCTX	124	235
POP	103	213	STE	125	235
POPC	104	215	STNZ	126	237
POPM	105	215	STZ	127	237
PUSH	106	216	STZX	128	238
PUSHA	107	218	SUB	129	238
PUSHC	108	218	TST	131	241
PUSHM	109	219	UND	132	243
REIT	110	219	WAIT	133	243
RMPA	111	220	XCHG	134	244
ROLC	112	220	XOR	135	245
RORC	113	221			

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Transfer	MOV	Transfer	92	195
	MOVA	Transfer effective address	94	202
	MOVDir	Transfer 4-bit data	95	203
	POP	Restore register/memory	103	213
	POPM	Restore multiple registers	105	215
	PUSH	Save register/memory/immediate data	106	216
	PUSHA	Save effective address	107	218
	PUSHM	Save multiple registers	109	219
	LDE	Transfer from extended data area	89	193
	STE	Transfer to extended data area	125	235
	STNZ	Conditional transfer	126	237
	STZ	Conditional transfer	127	237
	STZX	Conditional transfer	128	238
	XCHG	Exchange	134	244
Bit manipulation	BAND	Logically AND bits	47	152
	BCLR	Clear bit	48	152
	BM*Cnd*	Conditional bit transfer	49	154
	BNAND	Logically AND inverted bits	50	155
	BNOR	Logically OR inverted bits	51	156
	BNOT	Invert bit	52	156
	BNTST	Test inverted bit	53	157
	BNXOR	Exclusive OR inverted bits	54	158
	BOR	Logically OR bits	55	158
	BSET	Set bit	57	159
	BTST	Test bit	58	160
	BTSTC	Test bit & clear	59	161
	BTSTS	Test bit & set	60	162
	BXOR	Exclusive OR bits	61	162
Shift	ROLC	Rotate left with carry	112	220
	RORC	Rotate right with carry	113	221
	ROT	Rotate	114	222
	SHA	Shift arithmetic	118	227
	SHL	Shift logical	119	230
Arithmetic	ABS	Absolute value	39	140
	ADC	Add with carry	40	140
	ADCF	Add carry flag	41	142
	ADD	Add without carry	42	142
	CMP	Compare	62	163
	DADC	Decimal add with carry	64	167

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Arithmetic	DADD	Decimal add without carry	65	169
	DEC	Decrement	66	171
	DIV	Signed divide	67	172
	DIVU	Unsigned divide	68	173
	DIVX	Singed divide	69	174
	DSBB	Decimal subtract with borrow	70	175
	DSUB	Decimal subtract without borrow	71	177
	EXTS	Extend sign	74	180
	INC	Increment	77	182
	MUL	Signed multiply	96	205
	MULU	Unsigned multiply	97	207
	NEG	Two’s complement	98	209
	RMPA	Calculate sum-of-products	111	220
	SBB	Subtract with borrow	116	224
	SUB	Subtract without borrow	129	238
Logical	AND	Logical AND	45	149
	NOT	Invert all bits	100	210
	OR	Logical OR	101	211
	TST	Test	131	241
	XOR	Exclusive OR	135	245
Jump	ADJNZ	Add & conditional jump	44	148
	SBJNZ	Subtract & conditional jump	117	226
	JCnd	Jump on condition	80	184
	JMP	Unconditional jump	81	185
	JMPI	Jump indirect	82	187
	JMPS	Jump to special page	83	188
	JSR	Subroutine call	84	189
	JSRI	Indirect subroutine call	85	190
	JSRS	Special page subroutine call	86	191
	RTS	Return from subroutine	115	223
String	SMOVB	Transfer string backward	120	232
	SMOVF	Transfer string forward	121	233
	SSTR	Store string	122	233
Other	BRK	Debug interrupt	56	159
	ENTER	Build stack frame	72	179
	EXITD	Deallocate stack frame	73	180
	FCLR	Clear flag register bit	75	181
	FSET	Set flag register bit	76	182
	INT	Interrupt by INT instruction	78	183
	INTO	Interrupt on overflow	79	184
	LDC	Transfer to control register	87	191

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code /number of cycles
Other	LDCTX	Restore context	88	192
	LDINTB	Transfer to INTB register	90	194
	LDIPL	Set interrupt enable level	91	195
	NOP	No operation	99	209
	POPC	Restore control register	104	215
	PUSHC	Save control register	108	218
	REIT	Return from interrupt	110	219
	STC	Transfer from control register	123	234
	STCTX	Save context	124	235
	UND	Interrupt for undefined instruction	132	243
	WAIT	Wait	133	243

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
ABS	○	○	○	○	○	○	○	○	○	○	○					39	140
ADC	○	○	○	○	○	○	○	○	○	○	○	○	○			40	140
ADCF	○	○	○	○	○	○	○	○	○	○	○					41	142
ADD[*1]	○	○	○	○	○	○	○	○	○	○	○	○	○			42	142
ADJNZ[*1]	○	○	○	○	○	○	○	○	○	○	○				○	44	148
AND	○	○	○	○	○	○	○	○	○	○	○	○	○			45	149
CMP	○	○	○	○	○	○	○	○	○	○	○	○	○			62	163
DADC	○	○										○	○			64	167
DADD	○	○										○	○			65	169
DEC	○	○			○			○			○					66	171
DIV	○	○	○	○	○	○	○	○	○	○	○	○	○			67	172
DIVU	○	○	○	○	○	○	○	○	○	○	○	○	○			68	173
DIVX	○	○	○	○	○	○	○	○	○	○	○	○	○			69	174
DSBB	○	○										○	○			70	175
DSUB	○	○										○	○			71	177
ENTER												○				72	179
EXTS	○		○[*2]			○	○	○	○	○	○					74	180
INC	○[*3]	○[*4]			○			○			○					77	182
INT															○	78	183
JMPI[*1]	○	○	○	○	○	○	○	○		○	○					82	187
JMPS												○				83	188
JSRI[*1]	○	○	○	○	○	○	○	○		○	○					85	190
JSRS												○				86	191
LDC[*1]	○	○	○	○	○	○	○	○	○	○	○		○			87	191
LDE[*1]	○	○	○	○	○	○	○	○	○	○	○					89	193

[*1] Has special instruction addressing.

[*2] Only R1L can be selected.

[*3] Only R0L can be selected.

[*4] Only R0H can be selected.

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
LDINTB														○		90	194
LDIPL															○	91	195
MOV[*1]	○	○	○	○	○	○	○	○	○	○	○	○	○			92	195
MOVA	○	○	○	○	○		○	○	○	○	○					94	202
MOV*Dir*	○	○	○	○		○	○	○	○	○	○					95	203
MUL	○	○	○	○	○	○	○	○	○	○	○	○	○			96	205
MULU	○	○	○	○	○	○	○	○	○	○	○	○	○			97	207
NEG	○	○	○	○	○	○	○	○	○	○	○					98	209
NOT	○	○	○	○	○	○	○	○	○	○	○					100	210
OR	○	○	○	○	○	○	○	○	○	○	○	○	○			101	211
POP	○	○	○	○	○	○	○	○	○	○	○					103	213
POPM[*1]	○	○	○	○	○											105	215
PUSH	○	○	○	○	○	○	○	○	○	○	○					106	216
PUSHA							○	○	○	○	○					107	218
PUSHM[*1]	○	○	○	○	○											109	219
ROLC	○	○	○	○	○	○	○	○	○	○	○					112	220
RORC	○	○	○	○	○	○	○	○	○	○	○					113	221
ROT	○	○	○	○	○	○	○	○	○	○	○				○	114	222
SBB	○	○	○	○	○	○	○	○	○	○	○	○	○			116	224
SBJNZ[*1]	○	○	○	○	○	○	○	○	○	○	○				○	117	226
SHA[*1]	○	○	○	○	○	○	○	○	○	○	○				○	118	227
SHL[*1]	○	○	○	○	○	○	○	○	○	○	○				○	119	230
STC[*1]	○	○	○	○	○	○	○	○	○	○	○					123	234
STCTX[*1]											○					124	235
STE[*1]	○	○	○	○	○	○	○	○	○	○	○					125	235

[*1] Has special instruction addressing.

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing															See page for function	See page for instruction code /number of cycles
	R0L/R0	R0H/R1	R1L/R2	R1H/R3	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16	#IMM8	#IMM16	#IMM20	#IMM		
STNZ	○	○						○			○	○				126	237
STZ	○	○						○			○	○				127	237
STZX	○	○						○			○	○				128	238
SUB	○	○	○	○	○	○	○	○	○	○	○	○	○			129	238
TST	○	○	○	○	○	○	○	○	○	○	○	○	○			131	241
XCHG	○	○	○	○	○	○	○	○	○	○	○					134	244
XOR	○	○	○	○	○	○	○	○	○	○	○	○	○			135	245

Quick Reference by Addressing (special instruction addressing)

Mnemonic	Addressing													See page for function	See page for instruction code /number of cycles
	dsp:20[A0]	dsp:20[A1]	abs20	R2R0/R3R1	A1A0	[A1A0]	dsp:8[SP]	label	SB/FB	ISP/USP	FLG	INTBL/INTBH	PC		
ADD[*1]										○				42	142
ADJNZ[*1]								○						44	148
JCnd								○						80	184
JMP			○					○						81	185
JMPI[*1]	○	○		○	○									82	187
JSR			○					○						84	189
JSRI[*1]	○	○		○	○									85	190
LDC[*1]									○	○	○	○		87	191
LDCTX			○											88	192
LDE[*1]	○		○			○								89	193
MOV[*1]							○							92	195
POPC									○	○	○	○		104	215
POPM[*1]									○					105	215
PUSHC									○	○	○	○		108	218
PUSHM[*1]									○					109	219
SBJNZ[*1]								○						117	226
SHA[*1]				○										118	227
SHL[*1]				○										119	230
STC[*1]				○	○				○	○	○	○	○	123	234
STCTX[*1]			○											124	235
STE[*1]	○		○			○								125	235

*1 Has general instruction addressing.

*2 INTBL and INTBH cannot be set simultaneously when using the LDINTB instruction.

Quick Reference by Addressing (bit instruction addressing)

Mnemonic	Addressing										See page for function	See page for instruction code /number of cycles
	bit,Rn	bit,An	[An]	base:8[An]	bit,base:8[SB/FB]	base:16[An]	bit,base:16[SB]	bit,base:16	bit,base:11	U/I/O/B/S/Z/D/C		
BAND	○	○	○	○	○	○	○	○			47	152
BCLR	○	○	○	○	○	○	○	○	○		48	152
BM*Cnd*	○	○	○	○	○	○	○	○		○	49	154
BNAND	○	○	○	○	○	○	○	○			50	155
BNOR	○	○	○	○	○	○	○	○			51	156
BNOT	○	○	○	○	○	○	○	○	○		52	156
BNTST	○	○	○	○	○	○	○	○			53	157
BNXOR	○	○	○	○	○	○	○	○			54	158
BOR	○	○	○	○	○	○	○	○			55	158
BSET	○	○	○	○	○	○	○	○	○		57	159
BTST	○	○	○	○	○	○	○	○	○		58	160
BTSTC	○	○	○	○	○	○	○	○			59	161
BTSTS	○	○	○	○	○	○	○	○			60	162
BXOR	○	○	○	○	○	○	○	○			61	162
FCLR										○	75	181
FSET										○	76	182

Chapter 1

Overview

1.1 Features of M16C/60, M16C/20, M16C/Tiny series
1.2 Address Space
1.3 Register Configuration
1.4 Flag Register (FLG)
1.5 Register Bank
1.6 Internal State after Reset is Cleared
1.7 Data Types
1.8 Data Arrangement
1.9 Instruction Format
1.10 Vector Table

1.1 Features of M16C/60, M16C/20, M16C/Tiny series

The M16C/60, M16C/20, M16C/Tiny series are single-chip microcomputer developed for built-in applications where the microcomputer is built into applications equipment.
The M16C/60, M16C/20, M16C/Tiny series support instructions suitable for the C language with frequently used instructions arranged in one- byte op-code. Therefore, it allows you for efficient program development with few memory capacity regardless of whether you are using the assembly language or C language. Furthermore, some instructions can be executed in clock cycle, making fast arithmetic processing possible. Its instruction set consists of 91 discrete instructions matched to the M16C's abundant addressing modes. This powerful instruction set allows to perform register-register, register-memory, and memory-memory operations, as well as arithmetic/logic operations on bits and 4-bit data.
Some models incorporate a multiplier, allowing for high-speed computation.

1.1.1 Features of M16C/60, M16C/20, M16C/Tiny series

- **Register configuration**

Data registers	Four 16-bit registers (of which two registers can be used as 8-bit registers)
Address registers	Two 16-bit registers
Base registers	Two 16-bit registers

- **Versatile instruction set**

C language-suited instructions (stack frame manipulation): ENTER, EXITD, etc.
Register and memory-indiscriminated instructions: MOV, ADD, SUB, etc.
Powerful bit manipulate instructions: BNOT, BTST, BSET, etc.
4-bit transfer instructions: MOVLL, MOVHL, etc.
Frequently used 1-byte instructions: MOV, ADD, SUB, JMP, etc.
High-speed 1-cycle instructions: MOV, ADD, SUB, etc.

- **1M-byte linear address space**

Relative jump instructions matched to distance of jump

- **Fast instruction execution time**

Shortest 1-cycle instructions: 91 instructions include 20 1-cycle instructions.
(Approximately 75% of instructions execute in five cycles or under.)

1.1.2 Speed performance

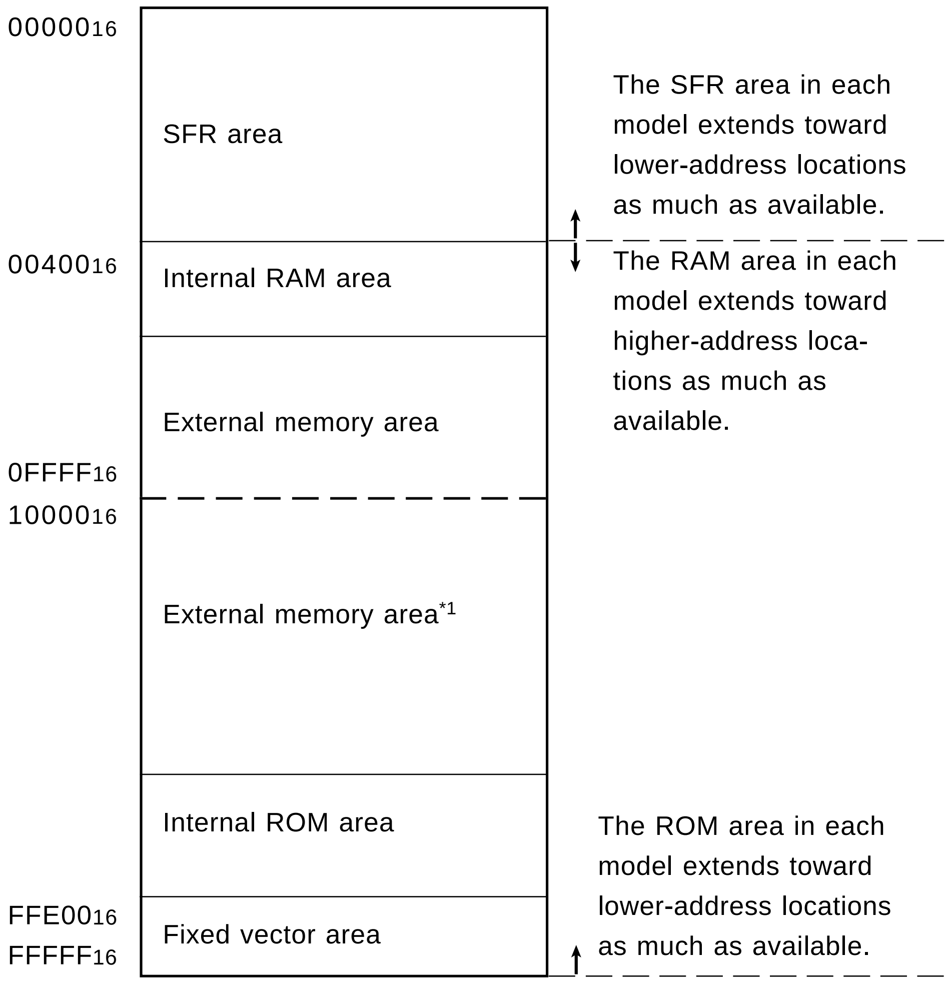
Register-register transfer 0.125 µs
Register-memory transfer 0.125 µs
Register-register addition/subtraction 0.125 µs
8 bits x 8 bits register-register operation 0.25 µs
16 bits x 16 bits register-register operation 0.313 µs
16 bits / 8 bits register-register operation 1.13 µs
32 bits / 16 bits register-register operation 1.56 µs

•Conditions
-Products with built-in Multiplier
-Clock frequency 16 MHz

1.2 Address Space

Fig. 1.2.1 shows an address space.

Addresses 00000_{16} through $003FF_{16}$ make up an SFR (special function register) area. In individual models of the M16C/60, M16C/20, M16C/Tiny series, the SFR area extends from $003FF_{16}$ toward lower addresses. Addresses from 00400_{16} on make up a memory area. In individual models of the M16C/60, M16C/20, M16C/Tiny series, a RAM area extends from address 00400_{16} toward higher addresses, and a ROM area extends from $FFFFF_{16}$ toward lower addresses. Addresses $FFE00_{16}$ through $FFFFF_{16}$ make up a fixed vector area.

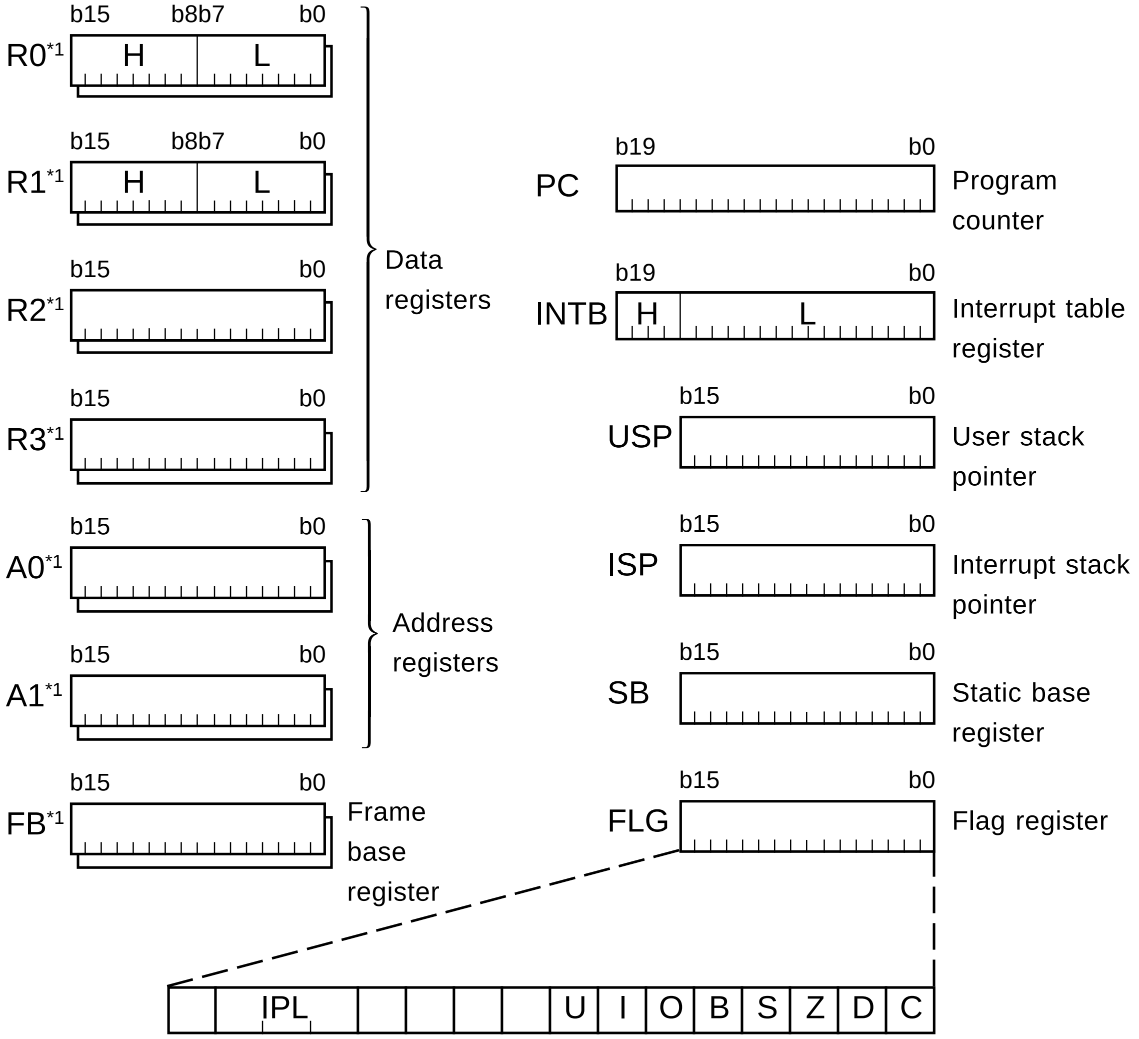

00000₁₆
SFR area
The SFR area in each model extends toward lower-address locations as much as available.
00400₁₆
Internal RAM area
The RAM area in each model extends toward higher-address locations as much as available.
External memory area
0FFFF₁₆
10000₁₆
External memory area*1
Internal ROM area
The ROM area in each model extends toward lower-address locations as much as available.
FFE00₁₆
Fixed vector area
FFFFF₁₆

*1 Locations above address 10000_{16} have restrictions on the types of instructions that can be used.

Figure 1.2.1 Address space

1.3 Register Configuration

The central processing unit (CPU) contains the 13 registers shown in Figure 1.3.1. Of these registers, R0, R1, R2, R3, A0, A1, and FB each consist of two sets of registers configuring two register banks.



*1 These registers have two register banks.

Figure 1.3.1 CPU register configuration

1.3.1 Data registers (R0, R0H, R0L, R1, R1H, R1L, R2, and R3)

The data registers (R0, R1, R2, and R3) consist of 16 bits, and are used primarily for transfers and arithmetic/logic operations.

Registers R0 and R1 can be halved into separate high-order (R0H, R1H) and low-order (R0L, R1L) parts for use as 8-bit data registers. For some instructions, moreover, you can combine R2 and R0 or R3 and R1 to configure a 32-bit data register (R2R0 or R3R1).

1.3.2 Address registers (A0 and A1)

The address registers (A0 and A1) consist of 16 bits, and have the similar functions as the data registers. These registers are used for address register-based indirect addressing and address register-based relative addressing.

For some instructions, registers A1 and A0 can be combined to configure a 32-bit address register (A1A0).

1.3.3 Frame base register (FB)

The frame base register (FB) consists of 16 bits, and is used for FB-based relative addressing.

1.3.4 Program counter (PC)

The program counter (PC) consists of 20 bits, indicating the address of an instruction to be executed next.

1.3.5 Interrupt table register (INTB)

The interrupt table register (INTB) consists of 20 bits, indicating the initial address of an interrupt vector table.

1.3.6 User stack pointer (USP) and interrupt stack pointer (ISP)

There are two types of stack pointers: user stack pointer (USP) and interrupt stack pointer (ISP), each consisting of 16 bits.

The stack pointer (USP/ISP) you want can be switched by a stack pointer select flag (U flag).

The stack pointer select flag (U flag) is bit 7 of the flag register (FLG).

1.3.7 Static base register (SB)

The static base register (SB) consists of 16 bits, and is used for SB-based relative addressing.

1.3.8 Flag register (FLG)

The flag register (FLG) consists of 11 bits, and is used as a flag, one bit for one flag. For details about the function of each flag, see Section 1.4, “Flag Register (FLG).”

1.4 Flag Register (FLG)

Figure 1.4.1 shows a configuration of the flag register (FLG). The function of each flag is detailed below.

1.4.1 Bit 0: Carry flag (C flag)

This flag holds a carry, borrow, or shifted-out bit that has occurred in the arithmetic/logic unit.

1.4.2 Bit 1: Debug flag (D flag)

This flag enables a single-step interrupt.

When this flag is set (= 1), a single-step interrupt is generated after an instruction is executed. When an interrupt is acknowledged, this flag is cleared to 0.

1.4.3 Bit 2: Zero flag (Z flag)

This flag is set when an arithmetic operation resulted in 0; otherwise, this flag is 0.

1.4.4 Bit 3: Sign flag (S flag)

This flag is set when an arithmetic operation resulted in a negative value; otherwise, this flag is 0.

1.4.5 Bit 4: Register bank select flag (B flag)

This flag selects a register bank. If this flag is 0, register bank 0 is selected; if the flag is 1, register bank 1 is selected.

1.4.6 Bit 5: Overflow flag (O flag)

This flag is set when an arithmetic operation resulted in overflow.

1.4.7 Bit 6: Interrupt enable flag (I flag)

This flag enables a maskable interrupt.

When this flag is 0, the interrupt is disabled; when the flag is 1, the interrupt is enabled. When the interrupt is acknowledged, this flag is cleared to 0.

1.4.8 Bit 7: Stack pointer select flag (U flag)

When this flag is 0, the interrupt stack pointer (ISP) is selected; when the flag is 1, the user stack pointer (USP) is selected.

This flag is cleared to 0 when a hardware interrupt is acknowledged or an INT instruction of software interrupt numbers 0 to 31 is executed.

1.4.9 Bits 8-11: Reserved area

1.4.10 Bits 12-14: Processor interrupt priority level (IPL)

The processor interrupt priority level (IPL) consists of three bits, allowing you to specify eight processor interrupt priority levels from level 0 to level 7. If a requested interrupt's priority level is higher than the processor interrupt priority level (IPL), this interrupt is enabled.

1.4.11 Bit 15: Reserved area

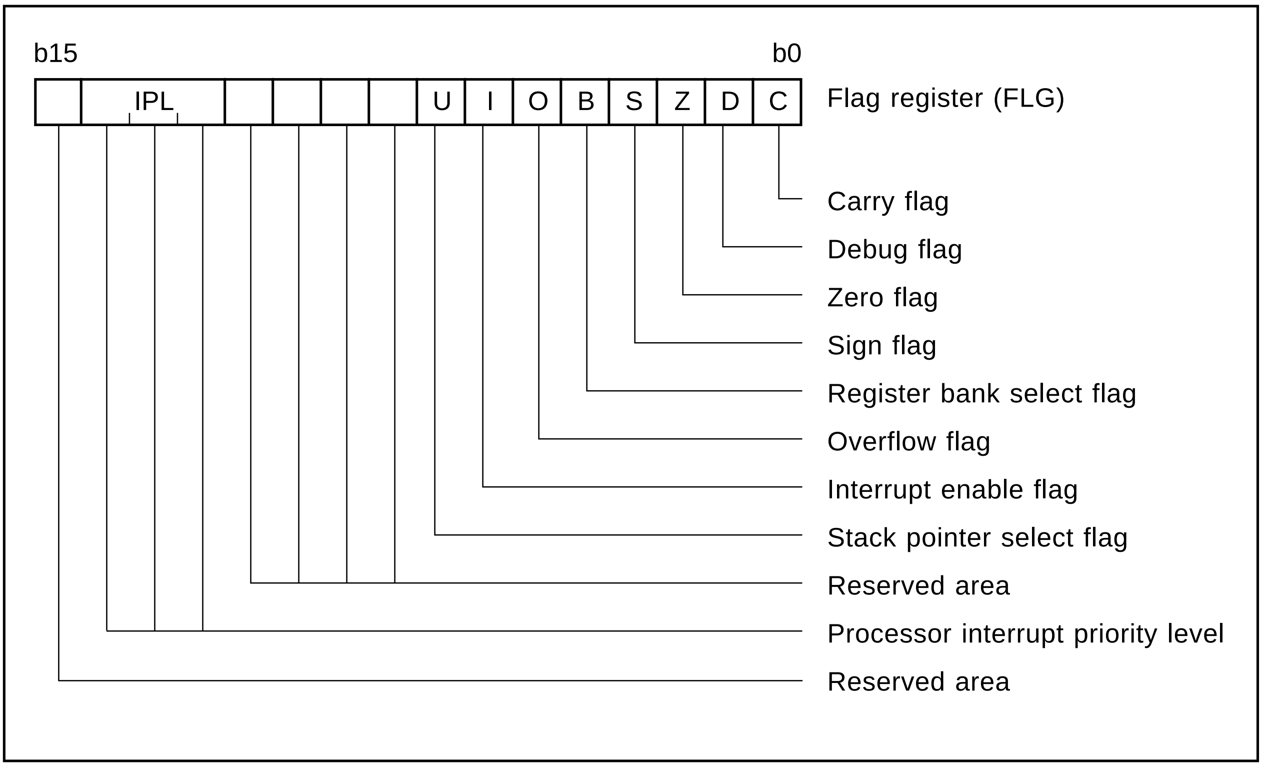


Figure 1.4.1 Configuration of flag register (FLG)

1.5 Register Bank

The M16C has two register banks, each configured with data registers (R0, R1, R2, and R3), address registers (A0 and A1), and frame base register (FB). These two register banks are switched over by the register bank select flag (B flag) of the flag register (FLG).

Figure 1.5.1 shows a configuration of register banks.

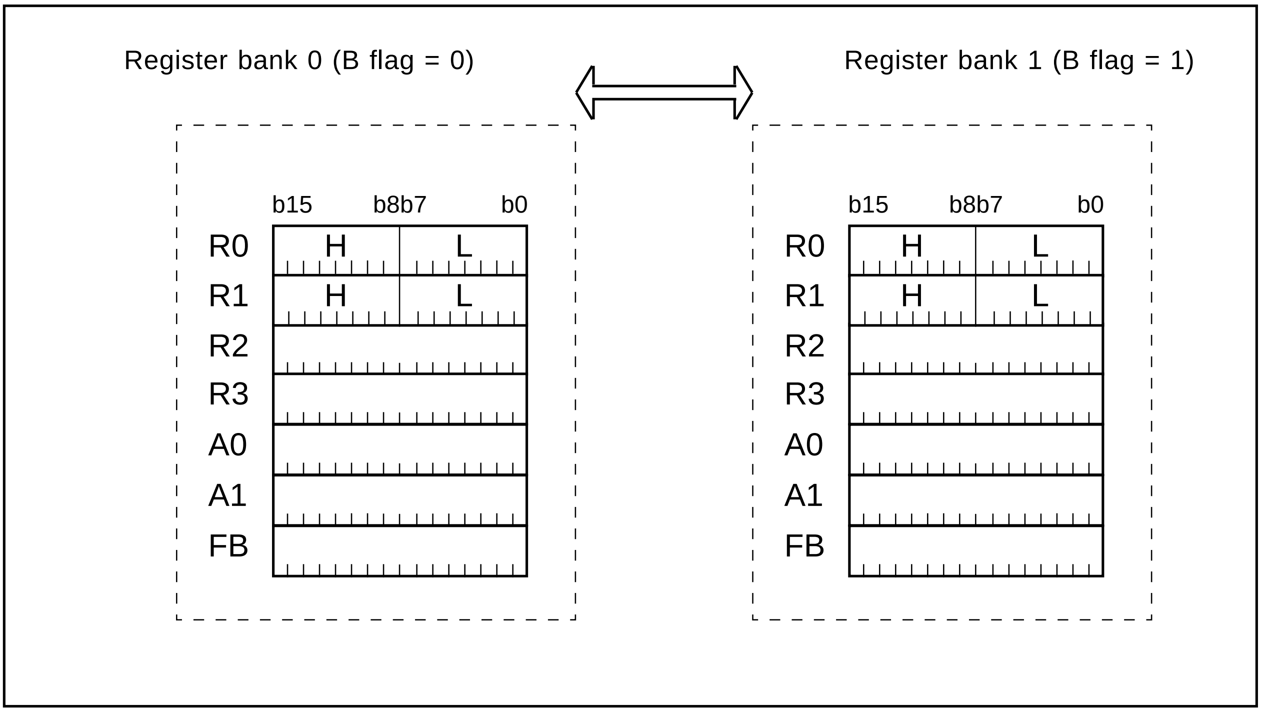


Figure 1.5.1 Configuration of register banks

1.6 Internal State after Reset is Cleared

The following lists the content of each register after a reset is cleared.

- Data registers (R0, R1, R2, and R3): 0000_{16}
- Address registers (A0 and A1): 0000_{16}
- Frame base register (FB): 0000_{16}
- Interrupt table register (INTB): 00000_{16}
- User stack pointer (USP): 0000_{16}
- Interrupt stack pointer (ISP): 0000_{16}
- Static base register (SB): 0000_{16}
- Flag register (FLG): 0000_{16}

1.7 Data Types

There are four data types: integer, decimal, bit, and string.

1.7.1 Integer

An integer can be a signed or an unsigned integer. A negative value of a signed integer is represented by two’s complement.

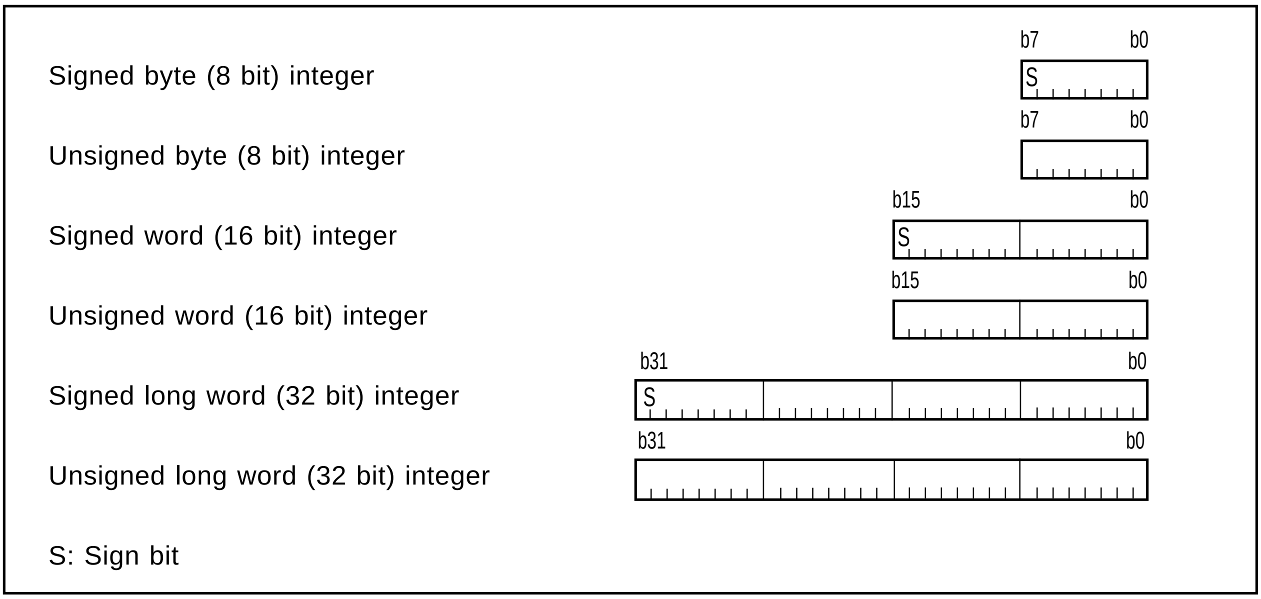

Signed byte (8 bit) integer
b7
b0
S
Unsigned byte (8 bit) integer
b7
b0
Signed word (16 bit) integer
b15
b0
S
Unsigned word (16 bit) integer
b15
b0
Signed long word (32 bit) integer
b31
b0
S
Unsigned long word (32 bit) integer
b31
b0
S: Sign bit

Figure 1.7.1 Integer data

1.7.2 Decimal

This type of data can be used in DADC, DADD, DSBB, and DSUB.

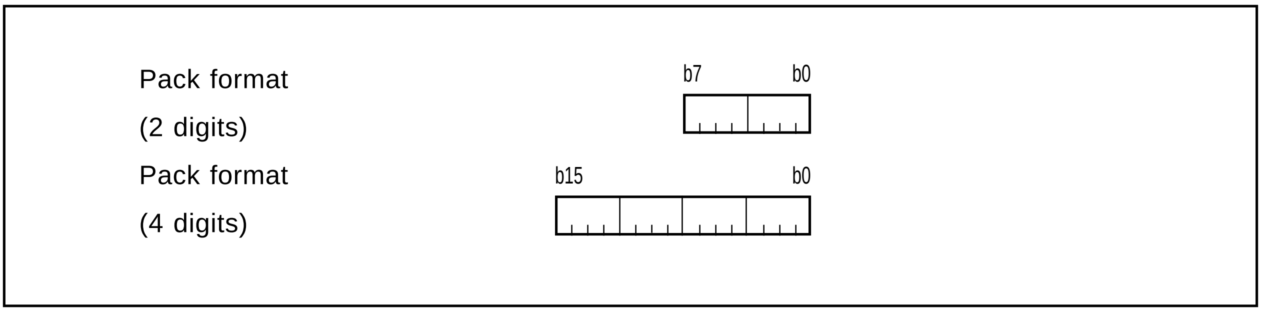


Figure 1.7.2 Decimal data

1.7.3 Bits

● **Register bits**

Figure 1.7.3 shows register bit specification.

Register bits can be specified by register direct (**bit, Rn** or **bit, An**). Use **bit, Rn** to specify a bit in data register (**Rn**); use **bit, An** to specify a bit in address register (**An**).

Bits in each register are assigned bit numbers 0-15, from LSB to MSB. For bit in **bit, Rn** and **bit, An**, you can specify a bit number in the range of 0 to 15.

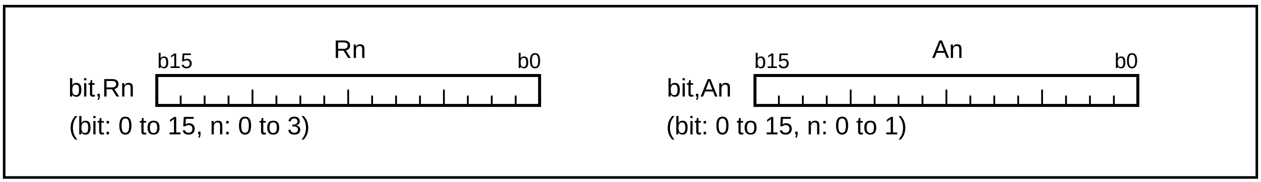


Figure 1.7.3 Register bit specification

● **Memory bits**

Figure 1.7.4 shows addressing modes used for memory bit specification. Table 1.7.1 lists the address range in which you can specify bits in each addressing mode. Be sure to observe the address range in Table 1.7.1 when specifying memory bits.

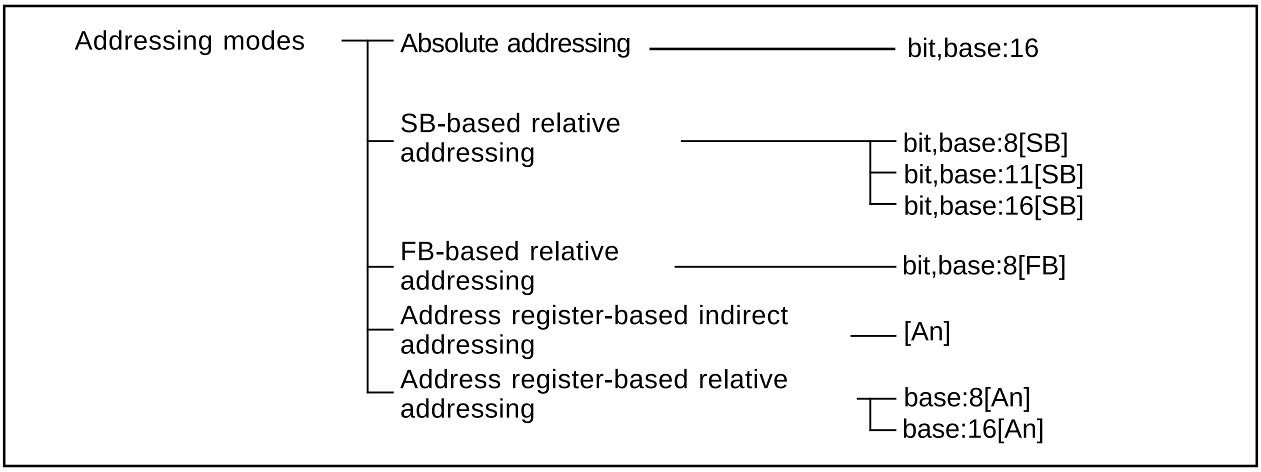

Addressing modes
Absolute addressing
bit,base:16
SB-based relative addressing
bit,base:8[SB]
bit,base:11[SB]
bit,base:16[SB]
FB-based relative addressing
bit,base:8[FB]
Address register-based indirect addressing
[An]
Address register-based relative addressing
base:8[An]
base:16[An]

Figure 1.7.4 Addressing modes used for memory bit specification

Table 1.7.1 Bit-Specifying Address Range

Addressing	Specification range		Remarks
	Lower limit (address)	Upper limit (address)	
bit,base:16	00000$_{16}$	01FFF$_{16}$	
bit,base:8[SB]	[SB]	[SB]+0001F$_{16}$	The access range is 00000$_{16}$ to 0FFFF$_{16}$.
bit,base:11[SB]	[SB]	[SB]+000FF$_{16}$	The access range is 00000$_{16}$ to 0FFFF$_{16}$.
bit,base:16[SB]	[SB]	[SB]+01FFF$_{16}$	The access range is 00000$_{16}$ to 0FFFF$_{16}$.
bit,base:8[FB]	[FB]Å|00010$_{16}$	[FB]+0000F$_{16}$	The access range is 00000$_{16}$ to 0FFFF$_{16}$.
[An]	00000$_{16}$	01FFF$_{16}$	
base:8[An]	base:8	base:8+01FFF$_{16}$	The access range is 00000$_{16}$ to 020FE$_{16}$.
base:16[An]	base:16	base:16+01FFF$_{16}$	The access range is 00000$_{16}$ to 0FFFF$_{16}$.

(1) Bit specification by bit, base

Figure 1.7.5 shows the relationship between memory map and bit map.

Memory bits can be handled as an array of consecutive bits. Bits can be specified by a given combination of **bit** and **base**. Using bit 0 of the address that is set to **base** as the reference (= 0), set the desired bit position to **bit**. Figure 1.7.6 shows examples of how to specify bit 2 of address 0000A16.

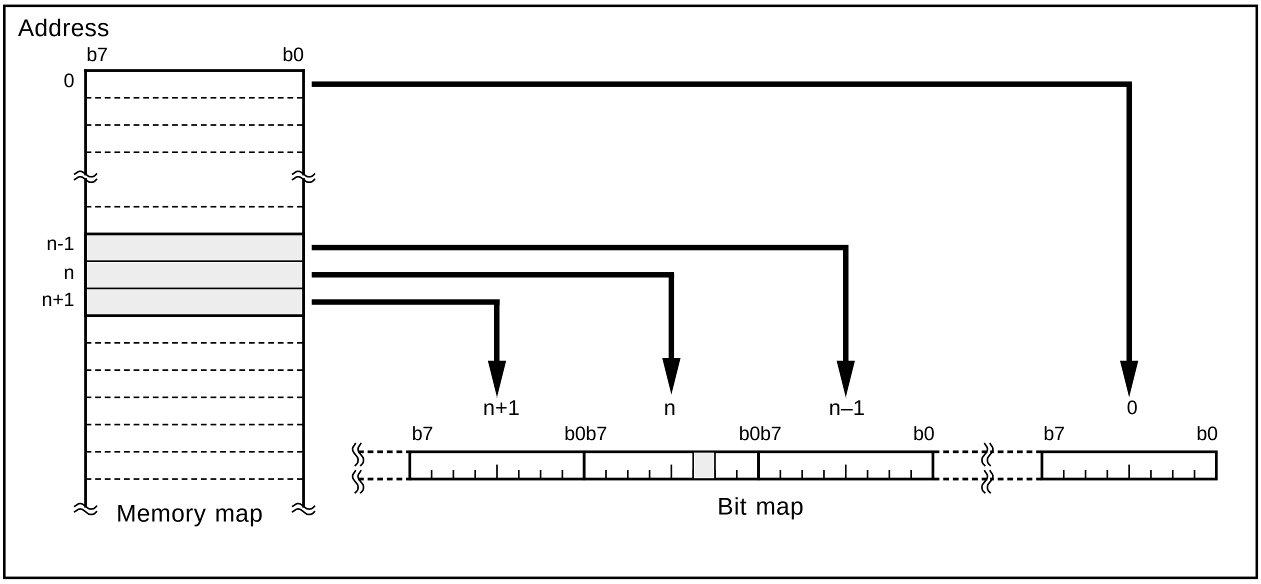


Figure 1.7.5 Relationship between memory map and bit map

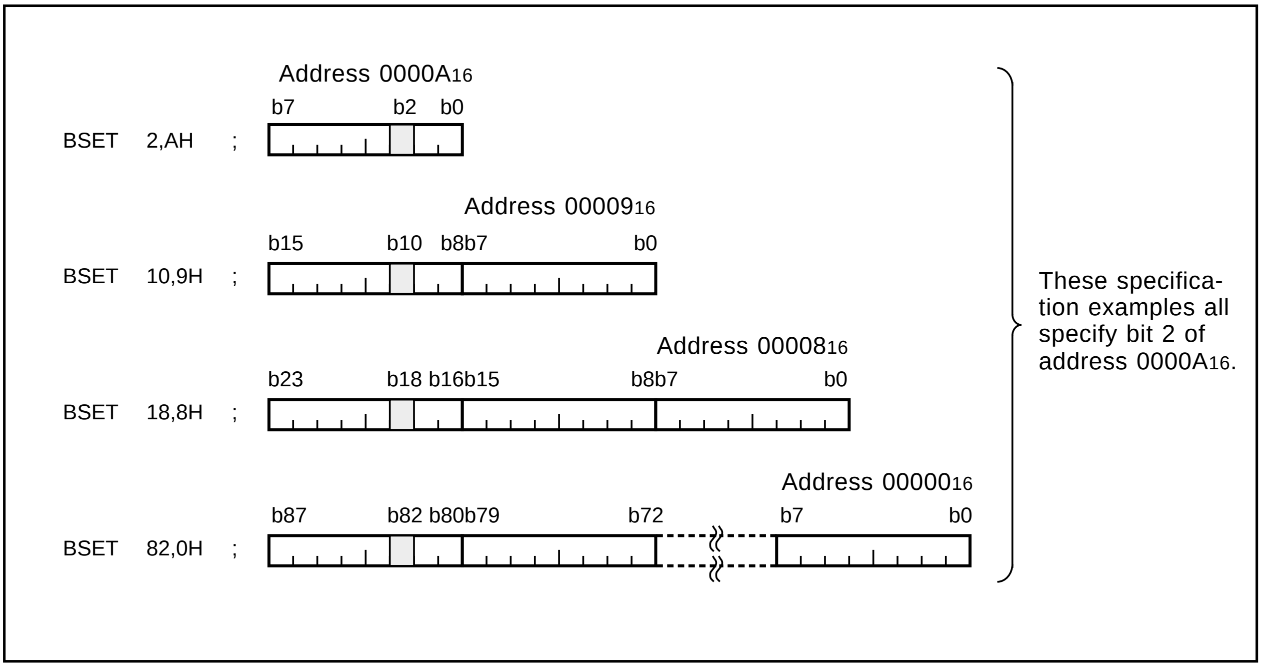


Figure 1.7.6 Examples of how to specify bit 2 of address 0000A16

(2) SB/FB relative bit specification

For SB/FB-based relative addressing, use bit 0 of the address that is the sum of the address set to static base register (**SB**) or frame base register (**FB**) plus the address set to **base** as the reference (= 0), and set your desired bit position to **bit**.

(3) Address register indirect/relative bit specification

For address register-based indirect addressing, use bit 0 of address 0000_{16} as the reference (= 0) and set your desired bit position to address register (**An**).
For address register-based relative addressing, use bit 0 of the address set to **base** as the reference (= 0) and set your desired bit position to address register (**An**).

1.7.4 String

String is a type of data that consists of a given length of consecutive byte (8-bit) or word (16-bit) data. This data type can be used in three types of string instructions: character string backward transfer (SMOVB instruction), character string forward transfer (SMOVF instruction), and specified area initialize (SSTR instruction).

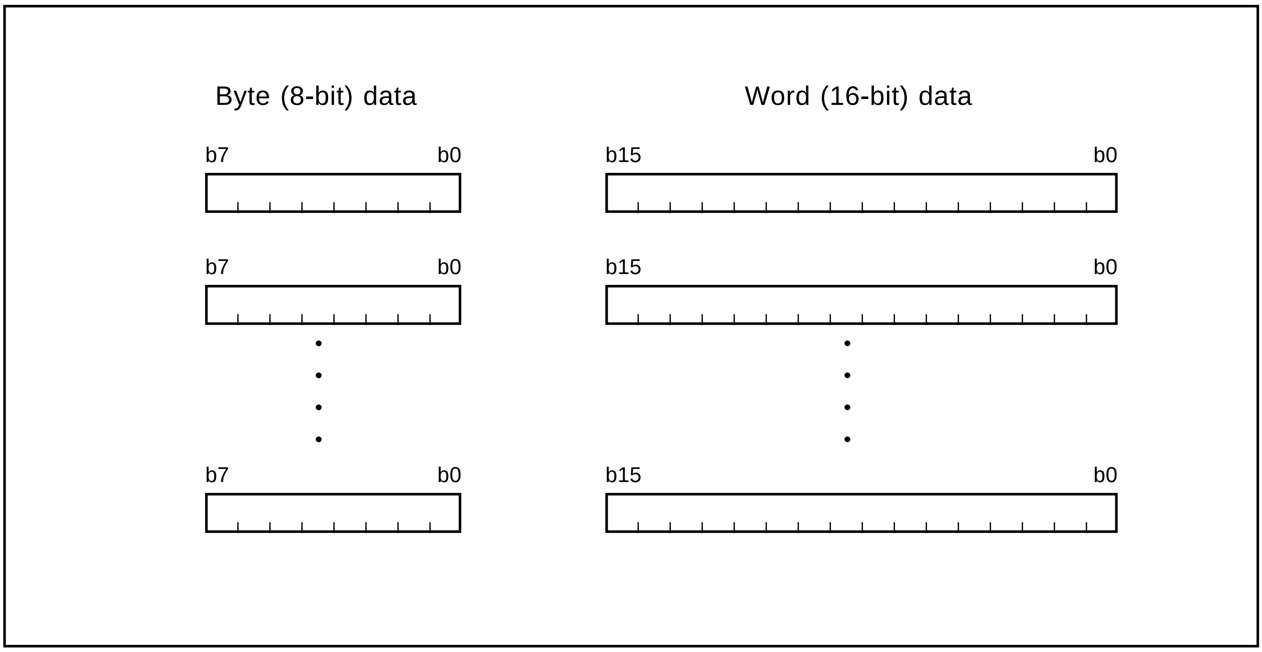


Figure 1.7.7 String data

1.8 Data Arrangement

1.8.1 Data Arrangement in Register

Figure 1.8.1 shows the relationship between a register’s data size and bit numbers.

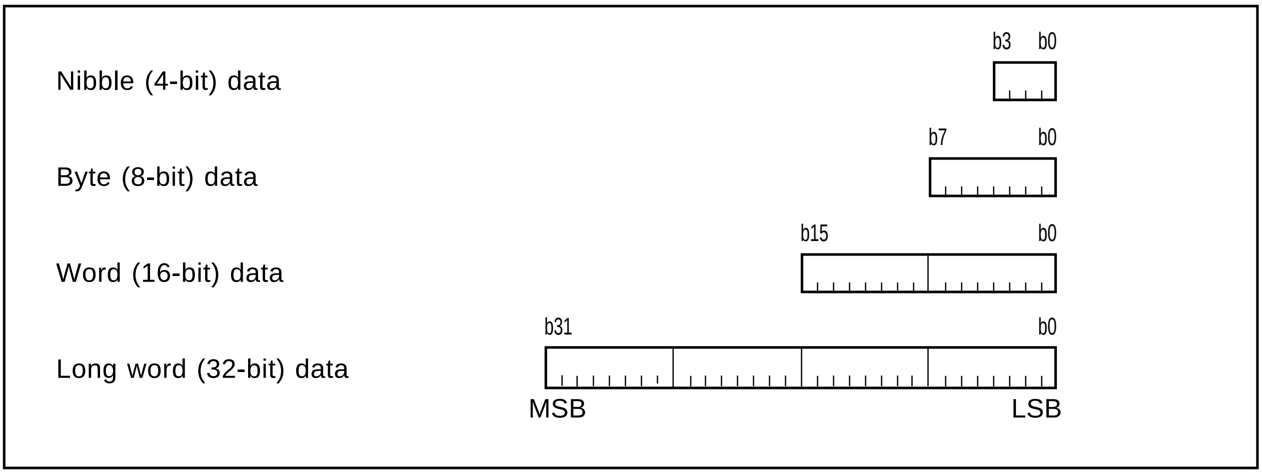


Figure 1.8.1 Data arrangement in register

1.8.2 Data Arrangement in Memory

Figure 1.8.2 shows data arrangement in memory. Figure 1.8.3 shows some examples of operation.

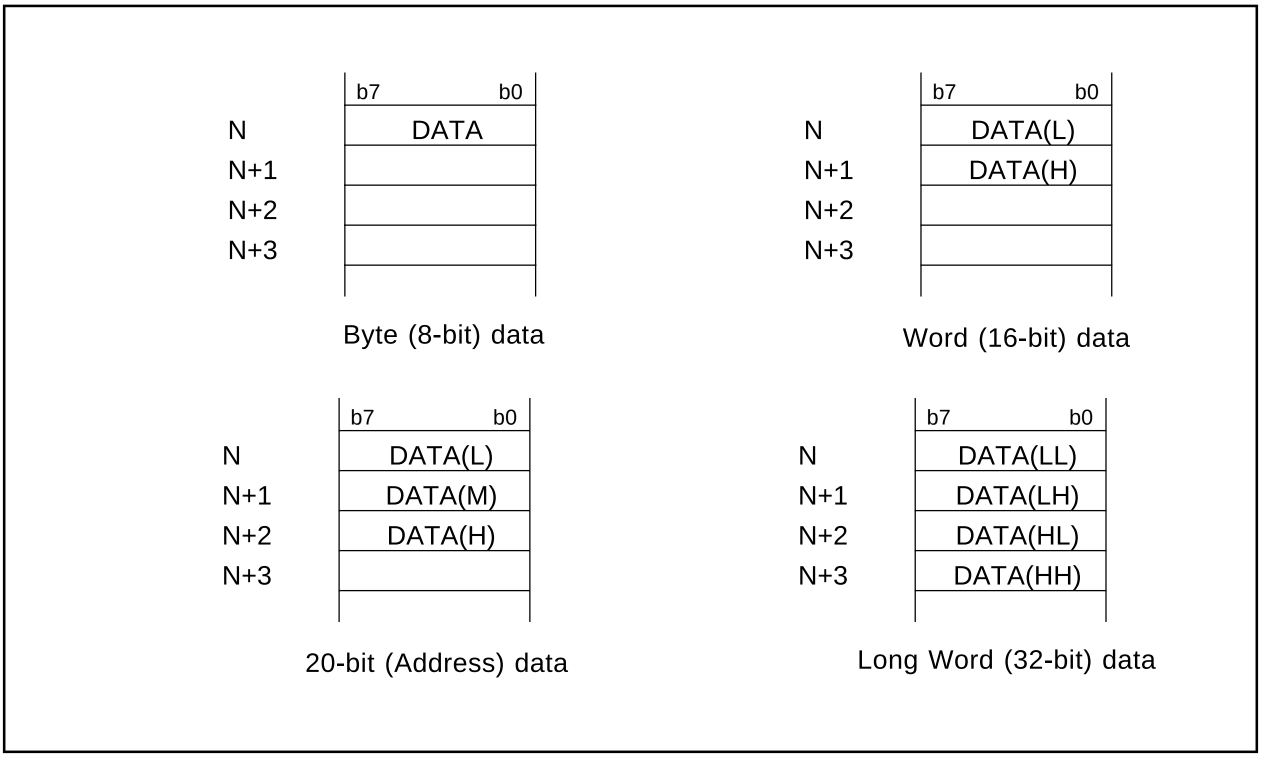


Figure 1.8.2 Data arrangement in memory

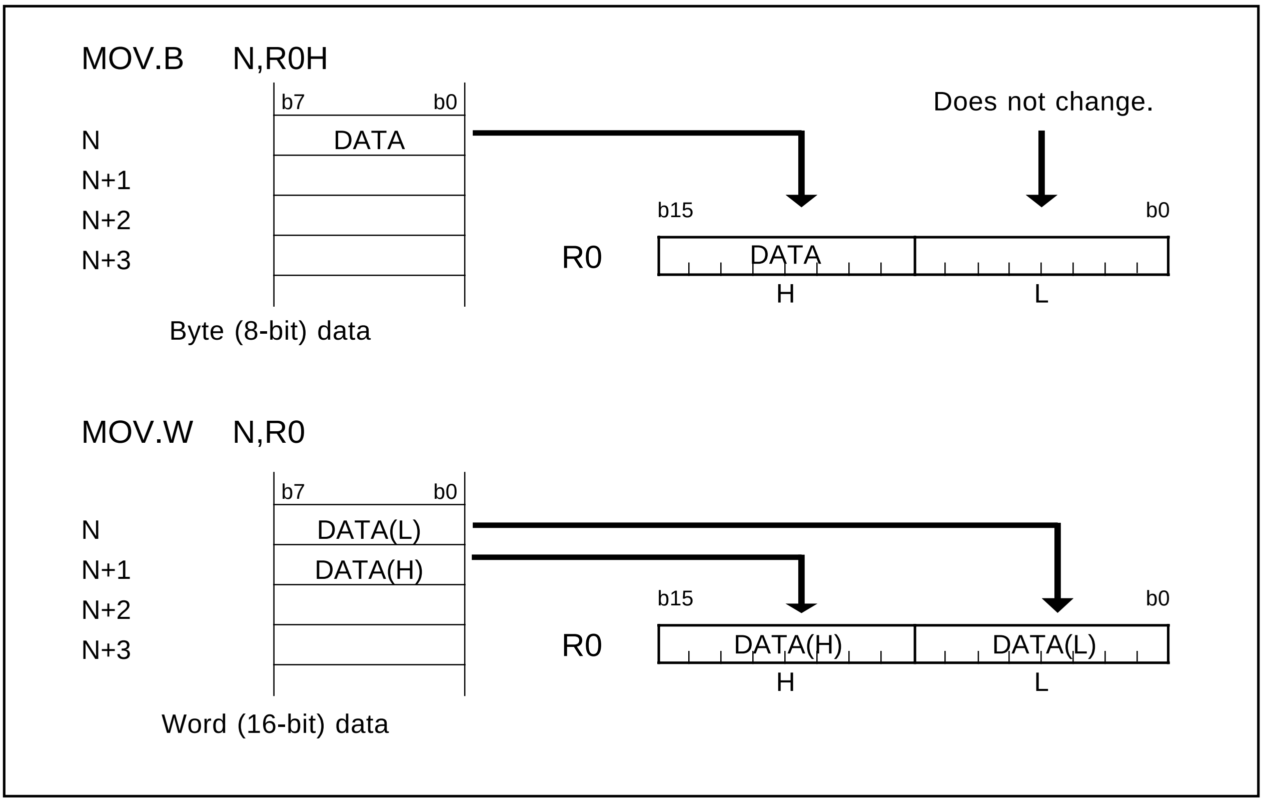


Figure 1.8.3 Examples of operation

1.9 Instruction Format

The instruction format can be classified into four types: generic, quick, short, and zero. The number of instruction bytes that can be chosen by a given format is least for the zero format, and increases successively for the short, quick, and generic formats in that order.
The following describes the features of each format.

1.9.1 Generic format (:G)

Op-code in this format consists of two bytes. This op-code contains information on operation and src[*1] and dest[*2] addressing modes.
Instruction code here is comprised of op-code (2 bytes), src code (0-3 bytes), and dest code (0-3 bytes).

1.9.2 Quick format (:Q)

Op-code in this format consists of two bytes. This op-code contains information on operation and immediate data and dest addressing modes. Note however that the immediate data in this op-code is a numeric value that can be expressed by -7 to +8 or -8 to +7 (varying with instruction).
Instruction code here is comprised of op-code (2 bytes) containing immediate data and dest code (0-2 bytes).

1.9.3 Short format (:S)

Op-code in this format consists of one byte. This op-code contains information on operation and src and dest addressing modes.Note however that the usable addressing modes are limited.
Instruction code here is comprised of op-code (1 byte), src code (0-2 bytes), and dest code (0-2 bytes).

1.9.4 Zero format (:Z)

Op-code in this format consists of one byte. This op-code contains information on operation (plus immediate data) and dest addressing modes. Note however that the immediate data is fixed to 0, and that the usable addressing modes are limited.
Instruction code here is comprised of op-code (1 byte) and dest code (0-2 bytes).

*1 src is the abbreviation of “source.”
*2 dest is the abbreviation of “destination.”

1.10 Vector Table

The vector table comes in two types: a special page vector table and an interrupt vector table. The special page vector table is a fixed vector table. The interrupt vector table can be a fixed or a variable vector table.

1.10.1 Fixed Vector Table

The fixed vector table is an address-fixed vector table. The special page vector table is allocated to addresses $FFE00_{16}$ through $FFFDB_{16}$, and part of the interrupt vector table is allocated to addresses $FFFDC_{16}$ through $FFFFF_{16}$. Figure 1.10.1 shows a fixed vector table.

The special page vector table is comprised of two bytes per table. Each vector table must contain the 16 low-order bits of the subroutine's entry address. Each vector table has special page numbers (18 to 255) which are used in JSRS and JMPS instructions.

The interrupt vector table is comprised of four bytes per table. Each vector table must contain the interrupt handler routine's entry address.

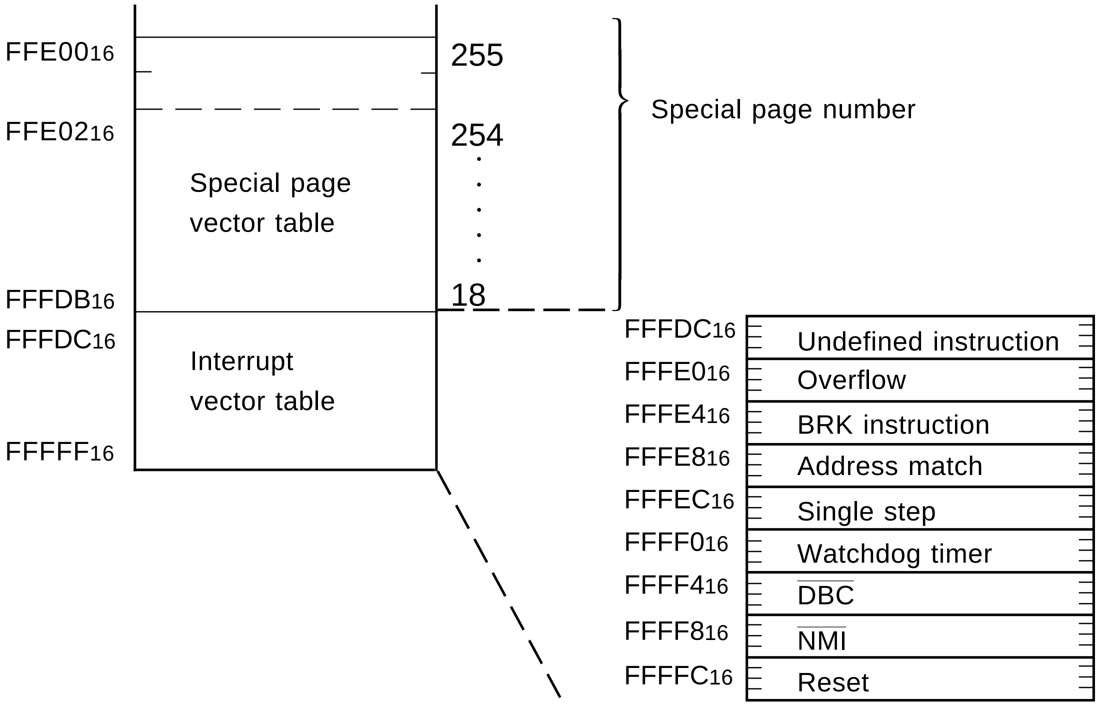


Figure 1.10.1 Fixed vector table

1.10.2 Variable Vector Table

The variable vector table is an address-variable vector table. Specifically, this vector table is a 256-byte interrupt vector table that uses the value indicated by the interrupt table register (INTB) as the entry address (IntBase). Figure 1.10.2 shows a variable vector table.

The variable vector table is comprised of four bytes per table. Each vector table must contain the interrupt handler routine's entry address.

Each vector table has software interrupt numbers (0 to 63). The INT instruction uses these software interrupt numbers.

Interrupts from the peripheral functions built in each M16C model are allocated to software interrupt numbers 0 through 31.

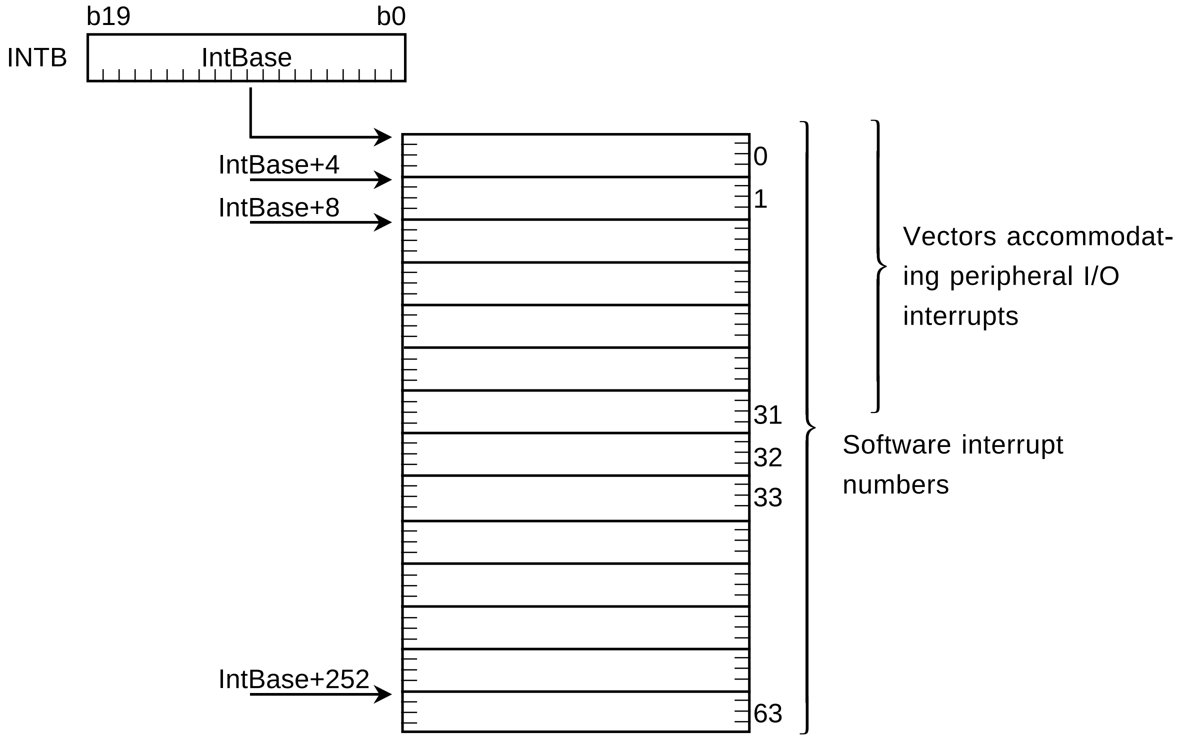

b19
b0
INTB
IntBase
IntBase+4
IntBase+8
IntBase+252
0
1
31
32
33
63
Vectors accommodating peripheral I/O interrupts
Software interrupt numbers

Figure 1.10.2 Variable vector table

Chapter 2

Addressing Modes

2.1 Addressing Modes
2.2 Guide to This Chapter
2.3 General Instruction Addressing
2.4 Special Instruction Addressing
2.5 Bit Instruction Addressing

2.1 Addressing Modes

This section describes addressing mode-representing symbols and operations for each addressing mode. The M16C/60, M16C/20, M16C/Tiny series have three addressing modes outlined below.

2.1.1 General instruction addressing

This addressing accesses an area from address 00000_{16} through address $0FFFF_{16}$.

The following lists the name of each general instruction addressing:

- Immediate
- Register direct
- Absolute
- Address register indirect
- Address register relative
- SB relative
- FB relative
- Stack pointer relative

2.1.2 Special instruction addressing

This addressing accesses an area from address 00000_{16} through address $FFFFF_{16}$ and control registers.

The following lists the name of each specific instruction addressing:

- 20-bit absolute
- Address register relative with 20-bit displacement
- 32-bit address register indirect
- 32-bit register direct
- Control register direct
- Program counter relative

2.1.3 Bit instruction addressing

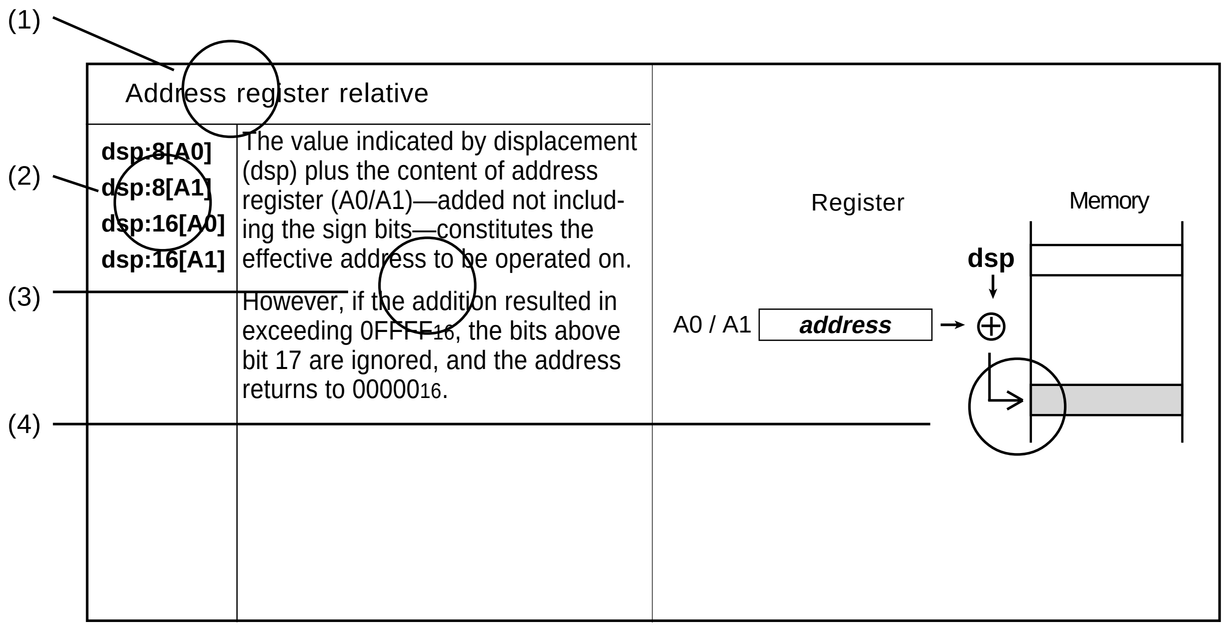
This addressing accesses an area from address 00000_{16} through address $0FFFF_{16}$.

The following lists the name of each bit instruction addressing:

- Register direct
- Absolute
- Address register indirect
- Address register relative
- SB relative
- FB relative
- FLG direct

2.2 Guide to This Chapter

The following shows how to read this chapter using an actual example.



Address register relative

dsp:8[A0]
dsp:8[A1]
dsp:16[A0]
dsp:16[A1]

The value indicated by displacement (dsp) plus the content of address register (A0/A1)—added not including the sign bits—constitutes the effective address to be operated on.

However, if the addition resulted in exceeding 0FFFF16, the bits above bit 17 are ignored, and the address returns to 0000016.

(1) Name

Indicates the name of addressing.

(2) Symbol

Represents the addressing mode.

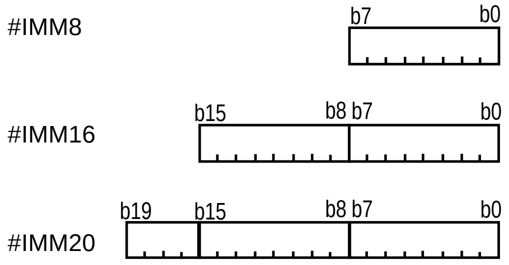
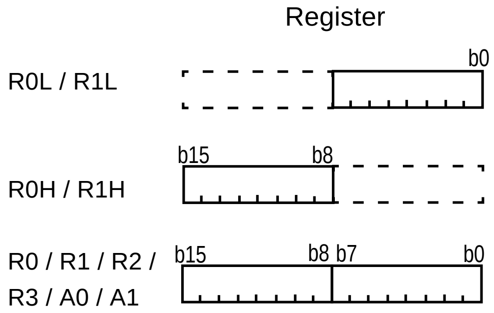
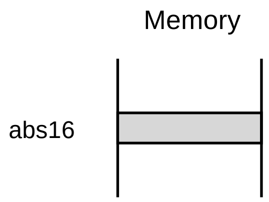
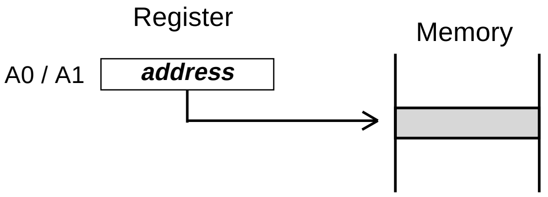
(3) Explanation

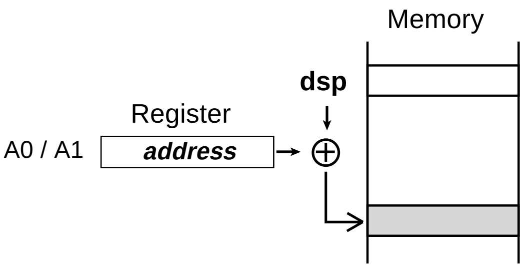
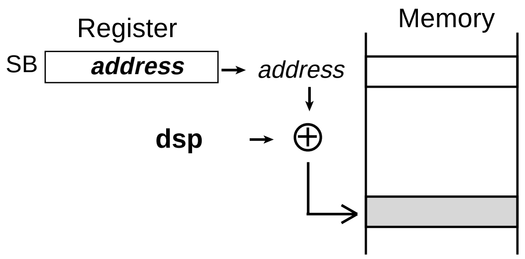
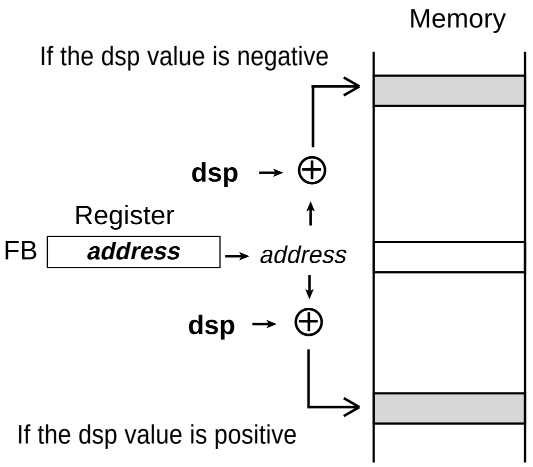
Describes the addressing operation and the effective address range.

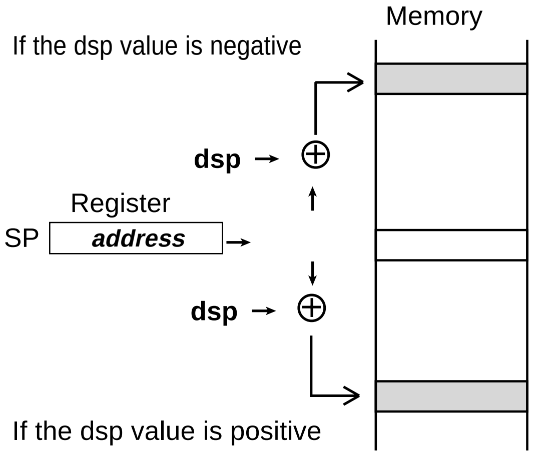
(4) Operation diagram

Diagrammatically explains the addressing operation.

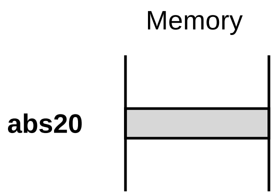
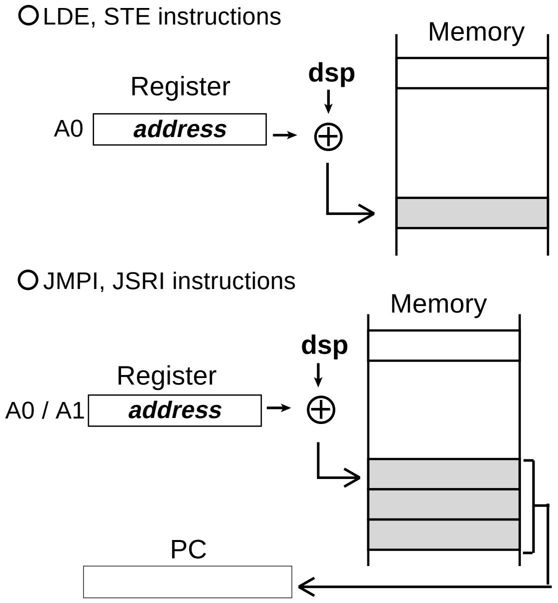
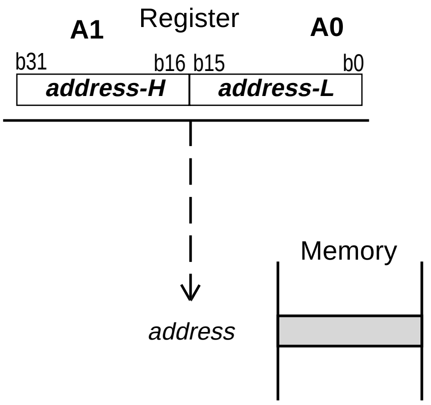
2.3 General Instruction Addressing

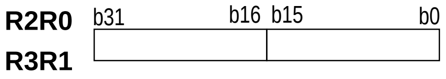
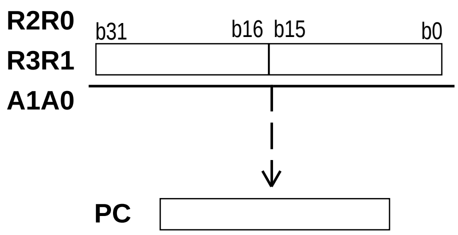
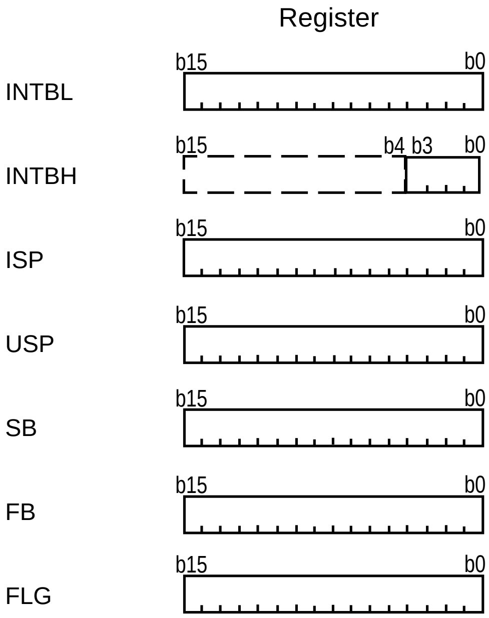
Immediate		 #IMM8 b7 b0 #IMM16 b15 b8 b7 b0 #IMM20 b19 b15 b8 b7 b0
#IMM **#IMM8** **#IMM16** **#IMM20**	The immediate data indicated by #IMM is the object to be operated on.	
Register direct		 Register b0 R0L / R1L b15 b8 R0H / R1H R0 / R1 / R2 / R3 / A0 / A1 b15 b8 b7 b0
R0L **R0H** **R1L** **R1H** **R0** **R1** **R2** **R3** **A0** **A1**	The specified register is the object to be operated on.	
Absolute		
abs16	The value indicated by abs16 constitutes the effective address to be operated on. The effective address range is 00000_{16} to $0FFFF_{16}$.	
Address register indirect		
[A0] **[A1]**	The value indicated by the content of address register (A0/A1) constitutes the effective address to be operated on. The effective address range is 00000_{16} to $0FFFF_{16}$.	

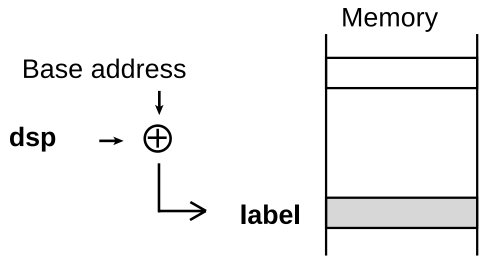
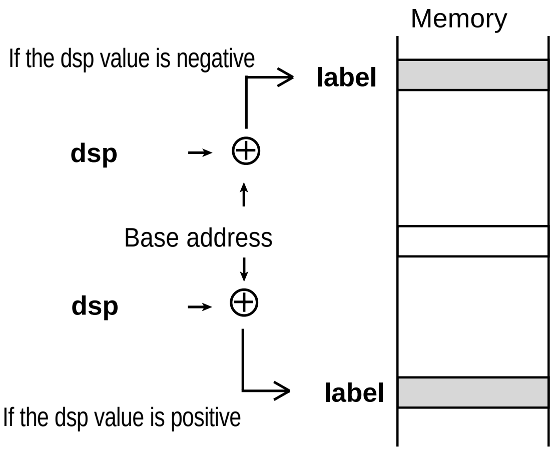
Address register relative		
dsp:8[A0] **dsp:8[A1]** **dsp:16[A0]** **dsp:16[A1]**	The value indicated by displacement (dsp) plus the content of address register (A0/A1)—added not including the sign bits—constitutes the effective address to be operated on. However, if the addition resulted in exceeding $0FFFF_{16}$, the bits above bit 17 are ignored, and the address returns to 00000_{16}.	
SB relative		
dsp:8[SB] **dsp:16[SB]**	The address indicated by the content of static base register (SB) plus the value indicated by displacement (dsp)—added not including the sign bits—constitutes the effective address to be operated on. However, if the addition resulted in exceeding $0FFFF_{16}$, the bits above bit 17 are ignored, and the address returns to 00000_{16}.	
FB relative		 Memory If the dsp value is negative dsp Register FB address address dsp If the dsp value is positive
dsp:8[FB]	The address indicated by the content of frame base register (FB) plus the value indicated by displacement (dsp)—added including the sign bits—constitutes the effective address to be operated on. However, if the addition resulted in exceeding 00000_{16}- $0FFFF_{16}$, the bits above bit 17 are ignored, and the address returns to 00000_{16} or $0FFFF_{16}$.	

<table>
<tr><td colspan="2">Stack pointer relative</td><td rowspan="2">
</td></tr>
<tr><td><b>dsp:8[SP]</b></td><td>The address indicated by the content of stack pointer (SP) plus the value indicated by displacement (dsp)—added including the sign bits—constitutes the effective address to be operated on. The stack pointer (SP) here is the one indicated by the U flag.
However, if the addition resulted in exceeding 00000_{16}- $0FFFF_{16}$, the bits above bit 17 are ignored, and the address returns to 00000_{16} or $0FFFF_{16}$.
This addressing can be used in MOV instruction.</td></tr>
</table>

2.4 Special Instruction Addressing

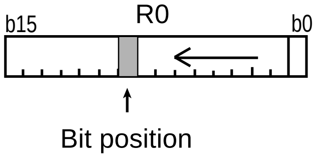
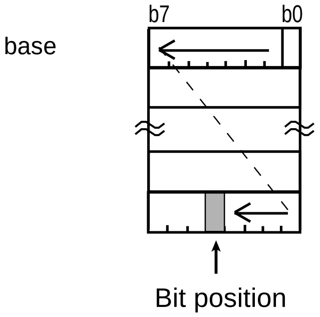
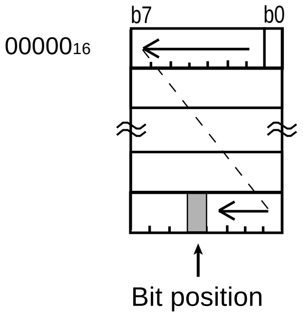
20-bit absolute		
abs20	The value indicated by abs20 constitutes the effective address to be operated on. The effective address range is 00000_{16} to $FFFFF_{16}$. This addressing can be used in LDE, STE, JSR, and JMP instructions.	
Address register relative with 20-bit displacement		 ○LDE, STE instructions Memory Register dsp A0 address ○JMPI, JSRI instructions Memory dsp Register A0 / A1 address PC
dsp:20[A0] **dsp:20[A1]**	The address indicated by displacement (dsp) plus the content of address register (A0/A1)—added not including the sign bits—constitutes the effective address to be operated on. However, if the addition resulted in exceeding $FFFFF_{16}$, the bits above bit 21 are ignored, and the address returns to 00000_{16}. This addressing can be used in LDE, STE, JMPI, and JSRI instructions. The following lists the addressing mode and instruction combinations that can be used. dsp:20[A0] → LDE, STE, JMPI, and JSRI instructions dsp:20[A1] → JMPI and JSRI instructions	
32-bit address register indirect		 A1 Register A0 b31 b16 b15 b0 address-H address-L Memory address
[A1A0]	The address indicated by 32 concatenated bits of address registers (A0 and A1) constitutes the effective address to be operated on. However, if the concatenated register value exceeds $FFFFF_{16}$, the bits above bit 21 are ignored. This addressing can be used in LDE and STE instructions.	

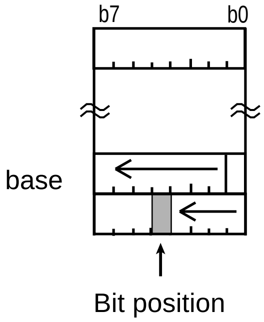
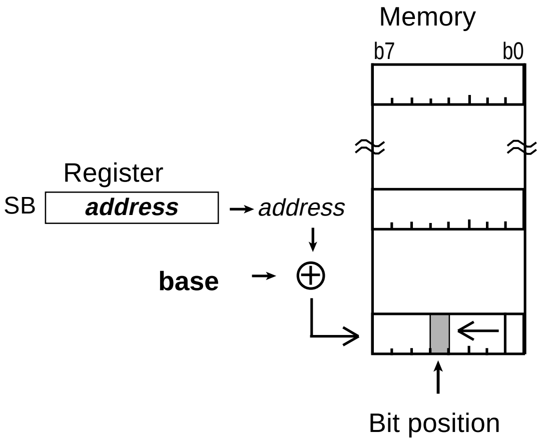
32-bit register direct		
R2R0 **R3R1** **A1A0**	The 32-bit concatenated register content of two specified registers is the object to be operated on. This addressing can be used in SHL, SHA, JMPI, and JSRI instructions. The following lists the register and instruction combinations that can be used. R2R0, R3R1 → SHL, SHA, JMPI, and JSRI instructions A1A0 → JMPI and JSRI instructions	○ SHL, SHA instructions  ○ JMPI, JSRI instructions 
Control register direct		
INTBL **INTBH** **ISP** **SP** **SB** **FB** **FLG**	The specified control register is the object to be operated on. This addressing can be used in LDC, STC, PUSHC, and POPC instructions. If you specify SP, the stack pointer indicated by the U flag is the object to be operated on.	 Register INTBL b15 b0 INTBH b15 b4 b3 b0 ISP b15 b0 USP b15 b0 SB b15 b0 FB b15 b0 FLG b15 b0

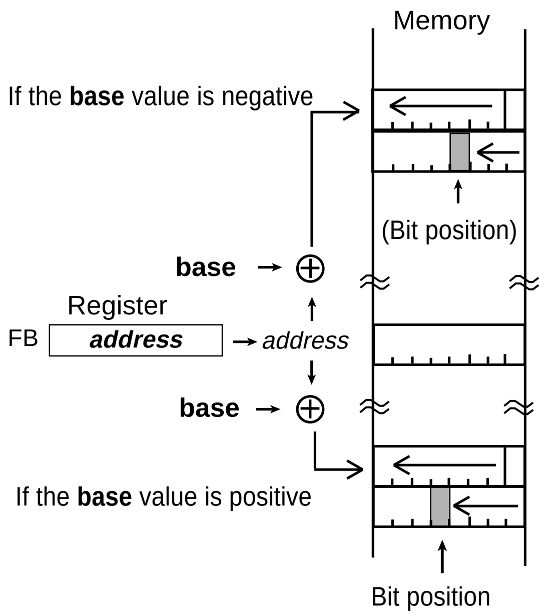
Program counter relative		
label	• If the jump length specifier (.length) is (.S)... the base address plus the value indicated by displacement (dsp)—added not including the sign bits—constitutes the effective address. This addressing can be used in JMP instruction.	 $+0 \leqq dsp \leqq +7$ *1 The base address is the (start address of instruction + 2).
	• If the jump length specifier (.length) is (.B) or (.W)... the base address plus the value indicated by displacement (dsp)—added including the sign bits—constitutes the effective address. However, if the addition resulted in exceeding 00000_{16}- $FFFFF_{16}$, the bits above bit 21 are ignored, and the address returns to 00000_{16} or $FFFFF_{16}$. This addressing can be used in JMP and JSR instructions.	 Memory If the dsp value is negative label dsp Base address dsp label If the dsp value is positive If the specifier is (.B), $-128 \leq dsp \leq +127$ If the specifier is (.W), $-32768 \leq dsp \leq +32767$ *2 The base address varies with each instruction.

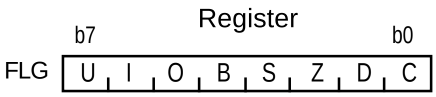
2.5 Bit Instruction Addressing

This addressing can be used in the following instructions:
BCLR, BSET, BNOT, BTST, BNTST, BAND, BNAND, BOR, BNOR, BXOR, BNXOR, BM*Cnd*, BTSTS, BTSTC

Register direct		bit , R0 
bit,R0 **bit,R1** **bit,R2** **bit,R3** **bit,A0** **bit,A1**	The specified register bit is the object to be operated on. For the bit position (**bit**) you can specify 0 to 15.	
Absolute		
bit,base:16	The bit that is as much away from bit 0 at the address indicated by **base** as the number of bits indicated by **bit** is the object to be operated on. Bits at addresses 00000_{16} through $01FFF_{16}$ can be the object to be operated on.	
Address register indirect		
[A0] **[A1]**	The bit that is as much away from bit 0 at address 00000_{16} as the number of bits indicated by address register (A0/A1) is the object to be operated on. Bits at addresses 00000_{16} through $01FFF_{16}$ can be the object to be operated on.	

Address register relative		
base:8[A0] **base:8[A1]** **base:16[A0]** **base:16[A1]**	The bit that is as much away from bit 0 at the address indicated by **base** as the number of bits indicated by address register (A0/A1) is the object to be operated on. However, if the address of the bit to be operated on exceeds 0FFFF16, the bits above bit 17 are ignored and the address returns to 0000016. The address range that can be specified by address register (A0/A1) is 8,192 bytes from **base**.	
SB relative		
bit,base:8[SB] **bit,base:11[SB]** **bit,base:16[SB]**	The bit that is as much away from bit 0 at the address indicated by static base register (SB) plus the value indicated by **base** (added not including the sign bits) as the number of bits indicated by **bit** is the object to be operated on. However, if the address of the bit to be operated on exceeds 0FFFF16, the bits above bit 17 are ignored and the address returns to 0000016. The address ranges that can be specified by bit,base: 8, bit,base: 11, and bit,base:16 respectively are 32 bytes, 256 bytes, and 8,192 bytes from the static base register (SB) value.	 Memory b7 b0 Register SB address address base Bit position

FB relative		
bit,base:8[FB]	The bit that is as much away from bit 0 at the address indicated by frame base register (FB) plus the value indicated by **base** (added including the sign bit) as the number of bits indicated by **bit** is the object to be operated on. However, if the address of the bit to be operated on exceeds 0000016-0FFFF16, the bits above bit 17 are ignored and the address returns to 0000016 or 0FFFF16. The address range that can be specified by bit,base: 8 is 16 bytes toward lower addresses or 15 bytes toward higher addresses from the frame base register (FB) value.	 Memory If the **base** value is negative (Bit position) **base** Register FB ***address*** *address* **base** If the **base** value is positive Bit position

FLG direct		
U **I** **O** **B** **S** **Z** **D** **C**	The specified flag is the object to be operated on. This addressing can be used in FCLR and FSET instructions.	 Register b7 b0 FLG U I O B S Z D C

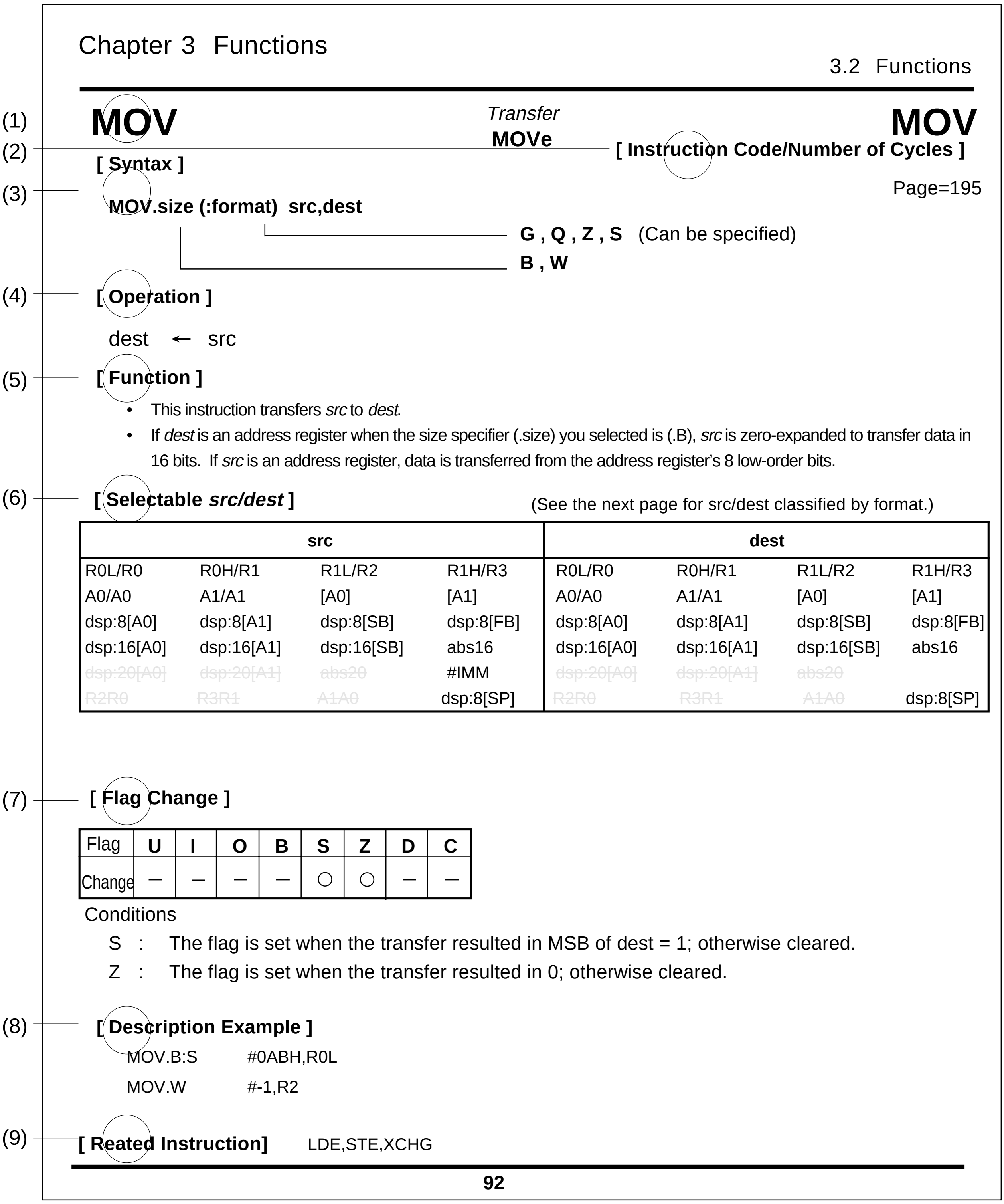
Chapter 3

Functions

3.1 Guide to This Chapter
3.2 Functions

3.1 Guide to This Chapter

This chapter describes the functionality of each instruction by showing syntax, operation, function, selectable src/dest, flag changes, description examples, and related instructions.
The following shows how to read this chapter by using an actual page as an example.



Chapter 3 Functions

3.2 Functions

(1) **MOV** *Transfer* **MOVe** **MOV**

(2) **[Instruction Code/Number of Cycles]**

Page=195

[Syntax]

(3) **MOV.size (:format) src,dest**

G , Q , Z , S (Can be specified)
B , W

(4) **[Operation]**

dest ← src

(5) **[Function]**

- This instruction transfers *src* to *dest*.
- If *dest* is an address register when the size specifier (.size) you selected is (.B), *src* is zero-expanded to transfer data in 16 bits. If *src* is an address register, data is transferred from the address register's 8 low-order bits.

(6) **[Selectable *src/dest*]** (See the next page for src/dest classified by format.)

src				**dest**			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]	A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]	~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]

(7) **[Flag Change]**

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions
S : The flag is set when the transfer resulted in MSB of dest = 1; otherwise cleared.
Z : The flag is set when the transfer resulted in 0; otherwise cleared.

(8) **[Description Example]**

```
MOV.B:S    #0ABH,R0L
MOV.W      #-1,R2
```

(9) **[Reated Instruction]** LDE,STE,XCHG

92

(1) Mnemonic

Indicates the mnemonic explained in this page.

(2) Instruction code/Number of Cycles

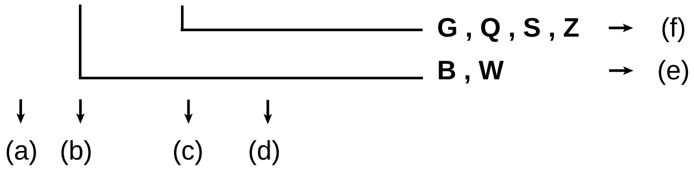
Indicates the page in which instruction code/number of cycles is listed.

Refer to this page for instruction code and number of cycles.

(3) Syntax

Indicates the syntax of the instruction using symbols. If (:format) is omitted, the assembler chooses the optimum specifier.

MOV.size (: format) src , dest



(a) Mnemonic **MOV**

Describes the mnemonic.

(b) Size specifier **size**

Describes the data size in which data is handled. The following lists the data sizes that can be specified:

.B Byte (8 bits)
.W Word (16 bits)
.L Long word (32 bits)

Some instructions do not have a size specifier.

(c) Instruction format specifier **(: format)**

Describes the instruction format. If (.format) is omitted, the assembler chooses the optimum specifier. If (.format) is entered, its content is given priority. The following lists the instruction formats that can be specified:

:G Generic format
:Q Quick format
:S Short format
:Z Zero format

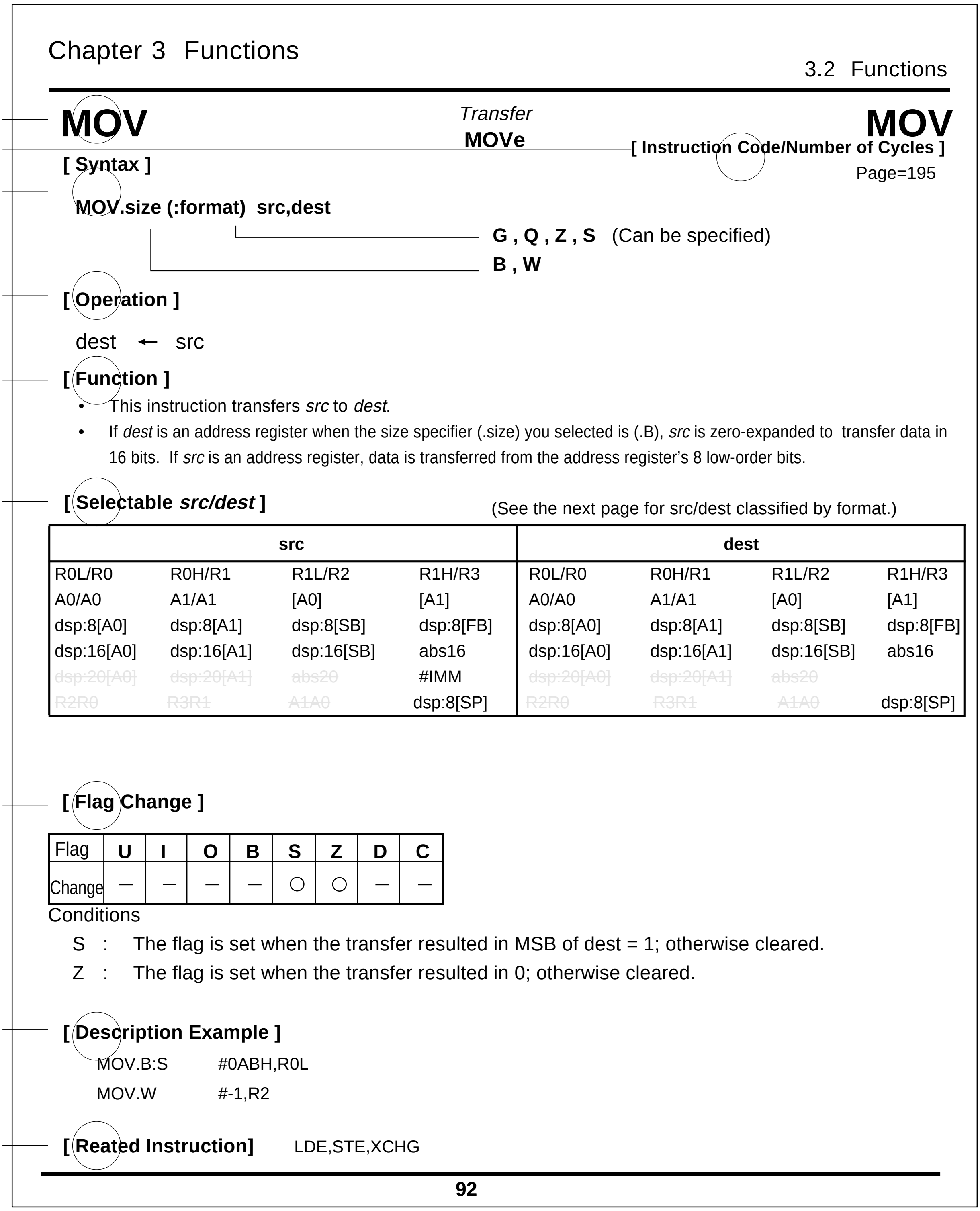
Some instructions do not have an instruction format specifier.

(d) Operand **src, dest**

Describes the operand.

(e) Indicates the data size you can specify in (b).

(f) Indicates the instruction format you can specify in (c).



Chapter 3 Functions

3.2 Functions

(1) **MOV** *Transfer* **MOVe** **MOV**

(2) **[Instruction Code/Number of Cycles]**

Page=195

[Syntax]

(3) **MOV.size (:format) src,dest**

G , Q , Z , S (Can be specified)

B , W

(4) **[Operation]**

dest ← src

(5) **[Function]**

- This instruction transfers *src* to *dest.*
- If *dest* is an address register when the size specifier (.size) you selected is (.B), *src* is zero-expanded to transfer data in 16 bits. If *src* is an address register, data is transferred from the address register's 8 low-order bits.

(6) **[Selectable *src/dest*]**

(See the next page for src/dest classified by format.)

src				**dest**			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]	A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]	~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]

(7) **[Flag Change]**

Flag	**U**	**I**	**O**	**B**	**S**	**Z**	**D**	**C**
Change	–	–	–	–	○	○	–	–

Conditions

S : The flag is set when the transfer resulted in MSB of dest = 1; otherwise cleared.

Z : The flag is set when the transfer resulted in 0; otherwise cleared.

(8) **[Description Example]**

```
MOV.B:S     #0ABH,R0L
MOV.W       #-1,R2
```

(9) **[Reated Instruction]** LDE,STE,XCHG

92

(4) Operation

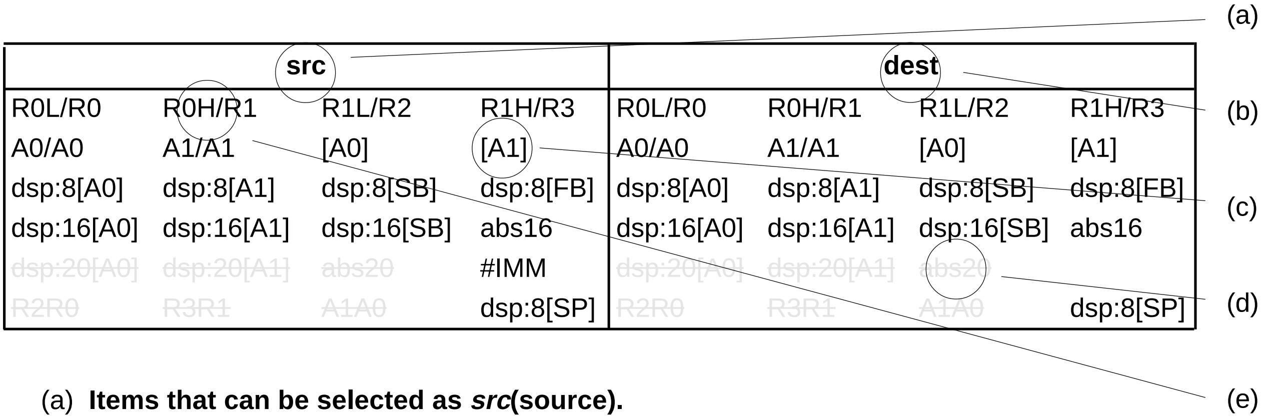
Explains the operation of the instruction using symbols.

(5) Function

Explains the function of the instruction and precautions to be taken when using the instruction.

(6) Selectable *src* / *dest* (label)

If the instruction has an operand, this indicates the format you can choose for the operand.



src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]	A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]	~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]

(a) **Items that can be selected as *src*(source).**

(b) **Items that can be selected as *dest*(destination).**

(c) **Addressing that can be selected.**

(d) **Addressing that cannot be selected.**

(e) **Shown on the left side of the slash (R0H) is the addressing when data is handled in bytes (8 bits). Shown on the right side of the slash (R1) is the addressing when data is handled in words (16 bits).**

(7) Flag change

Indicates a flag change that occurs after the instruction is executed. The symbols in the table mean the following:

"_" The flag does not change.
"O" The flag changes depending on condition.

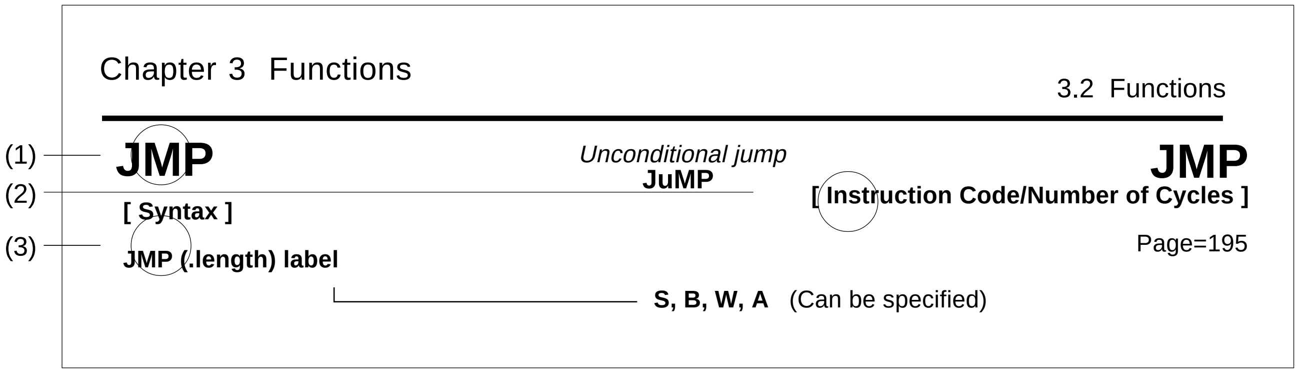
(8) Description example

Shows a description example for the instruction.

(9) Related instructions

Shows related instructions that cause an operation similar or opposite that of this instruction.

The following explains the syntax of each jump instruction—JMP, JPMI, JSR, and JSRI by using an actual example.



Chapter 3 Functions

3.2 Functions

(1) **JMP**

Unconditional jump
JuMP

JMP

(2) **[Syntax]**

[Instruction Code/Number of Cycles]

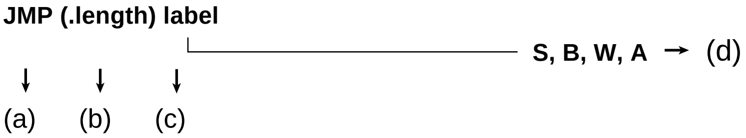
Page=195

(3) **JMP (.length) label**

S, B, W, A (Can be specified)

(3) Syntax

Indicates the instruction syntax using a symbol.



JMP (.length) label

S, B, W, A → (d)

(a) (b) (c)

(a) Mnemonic **JMP**

Describes the mnemonic.

(b) Jump distance specifier **.length**

Describes the distance of jump. If (.length) is omitted in JMP or JSR instruction, the assembler chooses the optimum specifier. If (.length) is entered, its content is given priority.

The following lists the jump distances that can be specified:

.S	3-bit PC forward relative (+2 to +9)
.B	8-bit PC relative
.W	16-bit PC relative
.A	20-bit absolute

(c) Operand **label**

Describes the operand.

(d) Shows the jump distance that can be specified in (b).

ABS *Absolute value* ABSolute ABS

[Syntax]

ABS.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 140

[Operation]

dest ← | dest |

[Function]

- This instruction takes on an absolute value of *dest* and stores it in *dest*.

[Selectable dest]

dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	○	○	—	○

Conditions

O : The flag is set (= 1) when dest before the operation is –128 (.B) or –32768 (.W); otherwise cleared (= 0).

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is indeterminate.

[Description Example]

```
ABS.B     R0L
ABS.W     A0
```

ADC *Add with carry* ADDition with Carry ADC

[Syntax]

ADC.size src,dest
└──── **B , W**

[Instruction Code/Number of Cycles]

Page= 140

[Operation]

dest ← src + dest + C

[Function]

- This instruction adds *dest*, *src*, and C flag together and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform calculation in 16 bits. If *src* is an A0 or A1, operation is performed on the eight low-order bits of the A0 or A1.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	○	–	○	○	–	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W) or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in exceeding +65535 (.W) or +255 (.B); otherwise cleared.

[Description Example]

```
ADC.B    #2,R0L
ADC.W    A0,R0
ADC.B    A0,R0L                 ; A0's 8 low-order bits and R0L are added.
ADC.B    R0L,A0                 ; R0L is zero-expanded and added with A0.
```

[Related Instructions] ADCF,ADD,SBB,SUB

ADCF

Add carry flag

ADdition Carry Flag

[Syntax]

ADCF.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 142

[Operation]

dest ← dest + C

[Function]

This instruction adds *dest* and C flag together and stores the result in *dest*.

[Selectable dest]

dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	○	–	○	○	–	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W) or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in exceeding +65535 (.W) or +255 (.B); otherwise cleared.

[Description Example]

```
ADCF.B    R0L
ADCF.W    Ram:16[A0]
```

[Related Instructions] ADC,ADD,SBB,SUB

ADD

Add without carry
ADDition

ADD

[Syntax]

[Instruction Code/Number of Cycles]

ADD.size (:format) src,dest

Page= 142

G , Q , S (Can be specified)

B , W

[Operation]

dest ← dest + src

[Function]

- This instruction adds *dest* and *src* together and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform calculation in 16 bits. If *src* is an A0 or A1, operation is performed on the eight low-order bits of the A0 or A1.
- If *dest* is a stack pointer when the size specifier (.size) you selected is (.B), *src* is sign extended to perform calculation in 16 bits.

[Selectable src/dest] (See the next page for *src*/*dest* classified by format.)

src				**dest**			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	SP/SP*2
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

*2 Operation is performed on the stack pointer indicated by the U flag. You can choose only #IMM for *src*.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	○	–	○	○	–	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W) or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in exceeding +65535 (.W) or +255 (.B); otherwise cleared.

[Description Example]

```
ADD.B    A0,R0L              ; A0's 8 low-order bits and R0L are added.
ADD.B    R0L,A0              ; R0L is zero-expanded and added with A0.
ADD.B    Ram:8[SB],R0L
ADD.W    #2,[A0]
```

[Related Instructions] ADC,ADCF,SBB,SUB

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	SP/SP*2
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

*2 Operation is performed on the stack pointer indicated by the U flag. You can choose only #IMM for *src*.

Q format

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0	A1/A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*3	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	SP/SP*2
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*2 Operation is performed on the stack pointer indicated by the U flag. You can choose only #IMM for *src*.

*3 The range of values that can be taken on is –8 ≤ #IMM ≤ +7.

S format*4

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#IMM			abs16	~~A0~~	~~A1~~	
R0L*5	R0H*5	dsp:8[SB]	dsp:8[FB]	R0L*5	R0H*5	~~dsp:8[SB]~~	~~dsp:8[FB]~~
abs16	~~#IMM~~			~~abs16~~	~~A0~~	~~A1~~	

*4 You can only specify (.B) for the size specifier (.size).

*5 You cannot choose the same register for *src* and *dest*.

ADJNZ

Add & conditional jump

ADdition then Jump on Not Zero

ADJNZ

[Syntax]

ADJNZ.size src,dest,label

B , W

[Instruction Code/Number of Cycles]

Page= 148

[Operation]

dest ← dest + src

if dest ≠ 0 **then** jump label

[Function]

- This instruction adds *dest* and *src* together and stores the result in *dest*.
- If the addition resulted in any value other than 0, control jumps to **label**. If the addition resulted in 0, the next instruction is executed.
- The op-code of this instruction is the same as that of SBJNZ.

[Selectable src/dest/label]

src	dest	label
#IMM*1	R0L/R0 R0H/R1 R1L/R2 R1H/R3 ~~A0~~/A0 ~~A1~~/A1 [A0] [A1] dsp:8[A0] dsp:8[A1] dsp:8[SB] dsp:8[FB] dsp:16[A0] dsp:16[A1] dsp:16[SB] abs16	PC*2–126≦label≦PC*2+129

*1 The range of values that can be taken on is –8 ≤ #IMM ≤ +7.

*2 PC indicates the start address of the instruction.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
ADJNZ.W    #–1,R0,label
```

[Related Instructions] SBJNZ

AND

Logically AND
AND

AND

[Syntax]

[Instruction Code/Number of Cycles]

AND.size (:format) src,dest

Page=149

G , S (Can be specified)

B , W

[Operation]

dest ← src ∧ dest

[Function]

- This instruction logically ANDs *dest* and *src* together and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform calculation in 16 bits. If *src* is an A0 or A1, operation is performed on the eight low-order bits of the A0 or A1.

[Selectable src/dest] (See the next page for *src*/*dest* classified by format.)

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
AND.B      Ram:8[SB],R0L
AND.B:G    A0,R0L             ; A0's 8 low-order bits and R0L are ANDed.
AND.B:G    R0L,A0             ; R0L is zero-expanded and ANDed with A0.
AND.B:S    #3,R0L
```

[Related Instructions] OR,XOR,TST

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

S format[*2]

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#IMM			abs16	~~A0~~	~~A1~~	
R0L[*3]	R0H[*3]	dsp:8[SB]	dsp:8[FB]	R0L[*3]	R0H[*3]	~~dsp:8[SB]~~	~~dsp:8[FB]~~
abs16	~~#IMM~~			~~abs16~~	~~A0~~	~~A1~~	

*2 You can only specify (.B) for the size specifier (.size).

*3 You cannot choose the same register for *src* and *dest*.

BAND

Logically AND bits
Bit AND carry flag

BAND

[Syntax]

BAND src

[Instruction Code/Number of Cycles]

Page=152

[Operation]

C ← src ∧ C

[Function]

- This instruction logically ANDs the C flag and *src* together and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Conditions

C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BAND    flag
BAND    4,Ram
BAND    16,Ram:16[SB]
BAND    [A0]
```

[Related Instructions] BOR,BXOR,BNAND,BNOR,BNXOR

BCLR

Clear bit

Bit CLeaR

BCLR

[Syntax]

BCLR (:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page= 152

[Operation]

dest ← 0

[Function]

- This instruction stores 0 in *dest*.

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
∈	bit,base:11[SB]*1		

*1 This *dest* can only be selected when in S format.

[Flag Change]

Flag	**U**	**I**	**O**	**B**	**S**	**Z**	**D**	**C**
Change	—	—	—	—	—	—	—	—

[Description Example]

```
BCLR      flag
BCLR      4,Ram:8[SB]
BCLR      16,Ram:16[SB]
BCLR      [A0]
```

[Related Instructions] BSET,BNOT,BNTST,BTST,BTSTC,BTSTS

BM*Cnd*

Conditional bit transfer

Bit Move Condition

BM*Cnd*

[Syntax]

BM*Cnd* **dest**

[Instruction Code/Number of Cycles]

Page=154

[Operation]

if true then dest ← 1

else dest ← 0

[Function]

- This instruction transfers the true or false value of the condition indicated by *Cnd* to *dest*. If the condition is true, 1 is transferred; if false, 0 is transferred.
- There are following kinds of *Cnd*.

Cnd	Condition		Expression	*Cnd*	Condition		Expression
GEU/C	C=1	Equal to or greater than C flag is 1.	≦	LTU/NC	C=0	Smaller than C flag is 0.	>
EQ/Z	Z=1	Equal to Z flag is 1.	=	NE/NZ	Z=0	Not equal Z flag is 0.	≠
GTU	$C\wedge\overline{Z}=1$	Greater than	<	LEU	$C\wedge\overline{Z}=0$	Equal to or smaller than	≧
PZ	S=0	Positive or zero	0≦	N	S=1	Negative	0>
GE	S∀O=0	Equal to or greater than (signed value)	≦	LE	(S∀O)∨Z=1	Equal to or smaller than (signed value)	≧
GT	(S∀O)∨Z=0	Greater than (signed value)	<	LT	S∀O=1	Smaller than (signed value)	>
O	O=1	O flag is 1.		NO	O=0	O flag is 0.	

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
C	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	*1

*1 The flag changes if you specified the C flag for *dest*.

[Description Example]

```
BMN      3,Ram:8[SB]
BMZ      C
```

[Related Instructions] J*Cnd*

BNAND

Logically AND inverted bits

Bit Not AND carry flag

BNAND

[Syntax]

BNAND src

[Instruction Code/Number of Cycles]

Page=155

[Operation]

C ← $\overline{\text{src}}$ ∨ C

[Function]

- This instruction logically ANDs the C flag and inverted *src* together and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Condition

C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BNAND     flag
BNAND     4,Ram
BNAND     16,Ram:16[SB]
BNAND     [A0]
```

[Related Instructions] BAND,BOR,BXOR,BNOR,BNXOR

BNOR

Logically OR inverted bits
Bit Not OR carry flag

[Syntax]
BNOR src

[Instruction Code/Number of Cycles]
Page= 156

[Operation]
C ← $\overline{\text{src}}$ ∨ C

[Function]
- This instruction logically ORs the C flag and inverted *src* together and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Condition
C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BNOR    flag
BNOR    4,Ram
BNOR    16,Ram:16[SB]
BNOR    [A0]
```

[Related Instructions] BAND,BOR,BXOR,BNAND,BNXOR

BNOT

Invert bit
Bit NOT

BNOT

[Syntax]

BNOT(:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page=156

[Operation]

dest ← $\overline{\text{dest}}$

[Function]

- This instruction inverts *dest* and stores the result in *dest*.

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
∈	bit,base:11[SB][*1]		

*1 This *dest* can only be selected when in S format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
BNOT      flag
BNOT      4,Ram:8[SB]
BNOT      16,Ram:16[SB]
BNOT      [A0]
```

[Related Instructions] BCLR,BSET,BNTST,BTST,BTSTC,BTSTS

BNTST

Test inverted bit
Bit Not TeST

BNTST

[Syntax]

BNTST src

[Instruction Code/Number of Cycles]

Page= 157

[Operation]

$Z \leftarrow \overline{\text{src}}$

$C \leftarrow \overline{\text{src}}$

[Function]

- This instruction transfers inverted *src* to the Z flag and inverted *src* to the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	○	—	○

Conditions

Z : The flag is set when *src* is 0; otherwise cleared.

C : The flag is set when *src* is 0; otherwise cleared.

[Description Example]

```
BNTST    flag
BNTST    4,Ram:8[SB]
BNTST    16,Ram:16[SB]
BNTST    [A0]
```

[Related Instructions] BCLR,BSET,BNOT,BTST,BTSTC,BTSTS

BNXOR

Exclusive OR inverted bits

Bit Not eXclusive OR carry flag

BNXOR

[Syntax]

BNXOR src

[Instruction Code/Number of Cycles]

Page= 158

[Operation]

C ← $\overline{\text{src}}$ ∀ C

[Function]

- This instruction exclusive ORs the C flag and inverted *src* and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Conditions

C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BNXOR    flag
BNXOR    4,Ram
BNXOR    16,Ram:16[SB]
BNXOR    [A0]
```

[Related Instructions] BAND,BOR,BXOR,BNAND,BNOR

BOR *Logically OR bits* **Bit OR carry flag** BOR

[Syntax]

BOR src

[Instruction Code/Number of Cycles]

Page= 158

[Operation]

C ← src ∨ C

[Function]

- This instruction logically ORs the C flag and *src* together and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Conditions

C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BOR     flag
BOR     4,Ram
BOR     16,Ram:16[SB]
BOR     [A0]
```

[Related Instructions] BAND,BXOR,BNAND,BNOR,BNXOR

BRK

Debug interrupt
BReaK

BRK

[Syntax]

BRK

[Instruction Code/Number of Cycles]

Page= 159

[Operation]

```
SP      ←   SP  –  2
M(SP)   ←   (PC  +  1)H, FLG
SP      ←   SP  –  2
M(SP)   ←   (PC  +  1)ML
PC      ←   M(FFFE416)
```

[Function]

- This instruction generates a BRK interrupt.
- The BRK interrupt is a nonmaskable interrupt.

[Flag Change][*1]

Flag	U	I	O	B	S	Z	D	C
Change	○	○	–	–	–	–	○	–

Conditions

U : The flag is cleared.

I : The flag is cleared.

D : The flag is cleared.

*1 The flags are saved to the stack area before the BRK instruction is executed. After the interrupt, the flags change state as shown on the left.

[Description Example]

BRK

[Related Instructions] INT,INTO

BSET *Set bit* **Bit SET** BSET

[Syntax]

BSET (:format) dest

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page= 159

[Operation]

dest ← 1

[Function]

- This instruction stores 1 in *dest*.

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
	bit,base:11[SB]*1		

*1 This *dest* can only be selected when in S format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
BSET      flag
BSET      4,Ram:8[SB]
BSET      16,Ram:16[SB]
BSET      [A0]
```

[Related Instructions] BCLR,BNOT,BNTST,BTST,BTSTC,BTSTS

BTST *Test bit* Bit TeST BTST

[Syntax]

BTST (:format) src

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page= 160

[Operation]

Z ← $\overline{\text{src}}$

C ← src

[Function]

- This instruction transfers inverted *src* to the Z flag and non-inverted *src* to the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
∈	bit,base:11[SB]*1		

*1 This *src* can only be selected when in S format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	○	—	○

Conditions

Z : The flag is set when *src* is 0; otherwise cleared.

C : The flag is set when *src* is 1; otherwise cleared.

[Description Example]

```
BTST      flag
BTST      4,Ram:8[SB]
BTST      16,Ram:16[SB]
BTST      [A0]
```

[Related Instructions] BCLR,BSET,BNOT,BNTST,BTSTC,BTSTS

BTSTC

Test bit & clear
Bit TeST & Clear

BTSTC

[Syntax]

BTSTC dest

[Instruction Code/Number of Cycles]

Page= 161

[Operation]

Z ← $\overline{\text{dest}}$

C ← dest

dest ← 0

[Function]

- This instruction transfers inverted *dest* to the Z flag and non-inverted *dest* to the C flag. Then it stores 0 in *dest*.

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	○	—	○

Conditions

Z : The flag is set when *dest* is 0; otherwise cleared.

C : The flag is set when *dest* is 1; otherwise cleared.

[Description Example]

```
BTSTC     flag
BTSTC     4,Ram
BTSTC     16,Ram:16[SB]
BTSTC     [A0]
```

[Related Instructions] BCLR,BSET,BNOT,BNTST,BTST,BTSTS

BTSTS

Test bit & set

Bit TeST & Set

BTSTS

[Syntax]

BTSTS dest

[Instruction Code/Number of Cycles]

Page= 162

[Operation]

Z ← $\overline{\text{dest}}$

C ← dest

dest ← 1

[Function]

- This instruction transfers inverted *dest* to the Z flag and non-inverted *dest* to the C flag. Then it stores 1 in *dest*.

[Selectable dest]

dest			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	○	—	○

Conditions

Z : The flag is set when *dest* is 0; otherwise cleared.

C : The flag is set when *dest* is 1; otherwise cleared.

[Description Example]

```
BTSTS     flag
BTSTS     4,Ram
BTSTS     16,Ram:16[SB]
BTSTS     [A0]
```

[Related Instructions] BCLR,BSET,BNOT,BNTST,BTST,BTSTC

BXOR

Exclusive OR bits

Bit eXclusive OR carry flag

BXOR

[Syntax]

BXOR src

[Instruction Code/Number of Cycles]

Page= 162

[Operation]

C ← src ∀ C

[Function]

- This instruction exclusive ORs the C flag and *src* together and stores the result in the C flag.

[Selectable src]

src			
bit,R0	bit,R1	bit,R2	bit,R3
bit,A0	bit,A1	[A0]	[A1]
base:8[A0]	base:8[A1]	bit,base:8[SB]	bit,base:8[FB]
base:16[A0]	base:16[A1]	bit,base:16[SB]	bit,base:16
~~C~~	~~bit,base:11[SB]~~		

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	○

Conditions

C : The flag is set when the operation resulted in 1; otherwise cleared.

[Description Example]

```
BXOR      flag
BXOR      4,Ram
BXOR      16,Ram:16[SB]
BXOR      [A0]
```

[Related Instructions] BAND,BOR,BNAND,BNOR,BNXOR

CMP *Compare* **CoMPare** CMP

[Syntax]

CMP.size (:format) src,dest

G , Q , S (Can be specified)

B , W

[Instruction Code/Number of Cycles]

Page= 163

[Operation]

dest – src

[Function]

- Each flag bit of the flag register varies depending on the result of subtraction of *src* from *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest] (See the next page for *src*/*dest* classified by format.)

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	○	○	—	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W), or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

```
CMP.B:S    #10,R0L
CMP.W:G    R0,A0
CMP.W      #–3,R0
CMP.B      #5,Ram:8[FB]
CMP.B      A0,R0L                       ; A0’s 8 low-order bits and R0L are compared.
```

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

Q format

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0	A1/A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*2	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*2 The range of values that can be taken on is –8 ≤ #IMM ≤ +7.

S format*3

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#IMM			abs16	~~A0~~	~~A1~~	
R0L*4	R0H*4	dsp:8[SB]	dsp:8[FB]	R0L*4	R0H*4	~~dsp:8[SB]~~	~~dsp:8[FB]~~
abs16	~~#IMM~~			~~abs16~~	~~A0~~	~~A1~~	

*3 You can only specify (.B) for the size specifier (.size).

*4 You cannot choose the same register for *src* and *dest*.

DADC

Decimal add with carry

Decimal ADdition with Carry

DADC

[Syntax]

DADC.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 167

[Operation]

dest ← src + dest + C

[Function]

- This instruction adds *dest*, *src*, and C flag together in decimal and stores the result in *dest*.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	R0H/R1	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	–	–	○	○	–	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the operation resulted in exceeding +9999 (.W) or +99 (.B); otherwise cleared.

[Description Example]

```
DADC.B    #3,R0L
DADC.W    R1,R0
```

[Related Instructions] DADD,DSUB,DSBB

DADD *Decimal add without carry* **Decimal ADDition** DADD

[Syntax]

DADD.size src,dest
B , W

[Instruction Code/Number of Cycles]
Page= 169

[Operation]

dest ← src + dest

[Function]

- This instruction adds *dest* and *src* together in decimal and stores the result in *dest*.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	R0H/R1	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the operation resulted in exceeding +9999 (.W) or +99 (.B); otherwise cleared.

[Description Example]

```
DADD.B    #3,R0L
DADD.W    R1,R0
```

[Related Instructions] DADC,DSUB,DSBB

DEC *Decrement* DECrement DEC

[Syntax]

DEC.size dest

└ **B , W**

[Instruction Code/Number of Cycles]

Page= 171

[Operation]

dest ← dest – 1

[Function]

- This instruction decrements 1 from *dest* and stores the result in *dest*.

[Selectable dest]

dest			
R0L*1	R0H*1	dsp:8[SB]*1	dsp:8[FB]*1
abs16*1	A0*2	A1*2	

*1 You can only specify (.B) for the size specifier (.size).

*2 You can only specify (.W) for the size specifier (.size).

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
DEC.W     A0
DEC.B     R0L
```

[Related Instructions] INC

DIV *Signed divide* DIVide DIV

[Syntax]

DIV.size src

B , W

[Instruction Code/Number of Cycles]

Page= 172

[Operation]

If the size specifier (.size) is (.B)

R0L (quotient), R0H (remainder) ← R0 ÷ src

If the size specifier (.size) is (.W)

R0 (quotient), R2 (remainder) ← R2R0 ÷ src

[Function]

- This instruction divides R2R0 (R0)*1 by signed *src* and stores the quotient in R0 (R0L)*1 and the remainder in R2 (R0H)*1. The remainder has the same sign as the dividend. Shown in ( )*1 are the registers that are operated on when you selected (.B) for the size specifier (.size).
- If *src* is an A0 or A1 when the size specifier (.size) you selected is (.B), operation is performed on the 8 low-order bits of A0 or A1.
- If you specify (.B) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 8 bits or the divisor is 0. At this time, R0L and R0H are indeterminate.
- If you specify (.W) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 16 bits or the divisor is 0. At this time, R0 and R2 are indeterminate.

[Selectable src]

src			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	—	—	—	—

Conditions

O : The flag is set when the operation resulted in the quotient exceeding 16 bits (.W) or 8 bits (.B) or the divisor is 0; otherwise cleared.

[Description Example]

```
DIV.B      A0                                    ;A0's 8 low-order bits is the divisor.
DIV.B      #4
DIV.W      R0
```

[Related Instructions] DIVU,DIVX,MUL,MULU

DIVU

Unsigned divide

DIVide Unsigned

DIVU

[Syntax]

DIVU.size src

B , W

[Instruction Code/Number of Cycles]

Page= 173

[Operation]

If the size specifier (.size) is (.B)

R0L (quotient), R0H (remainder) ← R0 ÷ src

If the size specifier (.size) is (.W)

R0 (quotient), R2 (remainder) ← R2R0 ÷ src

[Function]

- This instruction divides R2R0 (R0)[*1] by unsigned *src* and stores the quotient in R0 (R0L)[*1] and the remainder in R2 (R0H)[*1]. Shown in ( )[*1] are the registers that are operated on when you selected (.B) for the size specifier (.size).
- If *src* is an A0 or A1 when the size specifier (.size) you selected is (.B), operation is performed on the 8 low-order bits of A0 or A1.
- If you specify (.B) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 8 bits or the divisor is 0. At this time, R0L and R0H are indeterminate.
- If you specify (.W) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 16 bits or the divisor is 0. At this time, R0 and R2 are indeterminate.

[Selectable src]

src			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	○	–	–	–	–	–

Conditions

O : The flag is set when the operation resulted in the quotient exceeding 16 bits (.W) or 8 bits (.B) or the divisor is 0; otherwise cleared.

[Description Example]

```
DIVU.B   A0                              ;A0's 8 low-order bits is the divisor.
DIVU.B   #4
DIVU.W   R0
```

[Related Instructions] DIV,DIVX,MUL,MULU

DIVX

Singed divide

DIVide eXtension

DIVX

[Syntax]

DIVX.size src

└──────── **B , W**

[Instruction Code/Number of Cycles]

Page= 174

[Operation]

If the size specifier (.size) is (.B)

R0L (quotient), R0H (remainder) ← R0 ÷ src

If the size specifier (.size) is (.W)

R0 (quotient), R2 (remainder) ← R2R0 ÷ src

[Function]

- This instruction divides R2R0 (R0)[*1] by signed *src* and stores the quotient in R0 (R0L)[*1] and the remainder in R2 (R0H)[*1]. The remainder has the same sign as the divisor. Shown in ( )[*1] are the registers that are operated on when you selected (.B) for the size specifier (.size).
- If *src* is an A0 or A1 when the size specifier (.size) you selected is (.B), operation is performed on the 8 low-order bits of A0 or A1.
- If you specify (.B) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 8 bits or the divisor is 0. At this time, R0L and R0H are indeterminate.
- If you specify (.W) for the size specifier (.size), the O flag is set when the operation resulted in the quotient exceeding 16 bits or the divisor is 0. At this time, R0 and R2 are indeterminate.

[Selectable src]

src			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0	A1/A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	—	—	—	—

Conditions

O : The flag is set when the operation resulted in the quotient exceeding 16 bits (.W) or 8 bits (.B) or the divisor is 0; otherwise cleared.

[Description Example]

```
DIVX.B     A0                                    ;A0's 8 low-order bits is the divisor.
DIVX.B     #4
DIVX.W     R0
```

[Related Instructions] DIV,DIVU,MUL,MULU

DSBB *Decimal subtract with borrow* Decimal SuBtract with Borrow DSBB

[Syntax]

DSBB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 175

[Operation]

dest ← dest – src – $\overline{C}$

[Function]

- This instruction subtracts *src* and inverted C flag from *dest* in decimal and stores the result in *dest*.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	R0H/R1	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

```
DSBB.B     #3,R0L
DSBB.W     R1,R0
```

[Related Instructions] DADC,DADD,DSUB

DSUB

Decimal subtract without borrow

Decimal SUBtract

DSUB

[Syntax]

DSUB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 177

[Operation]

dest ← dest – src

[Function]

- This instruction subtracts *src* from *dest* in decimal and stores the result in *dest*.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	R0H/R1	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

```
DSUB.B    #3,R0L
DSUB.W    R1,R0
```

[Related Instructions] DADC,DADD,DSBB

ENTER

Build stack frame
ENTER function

ENTER

[Syntax]

ENTER src

[Instruction Code/Number of Cycles]

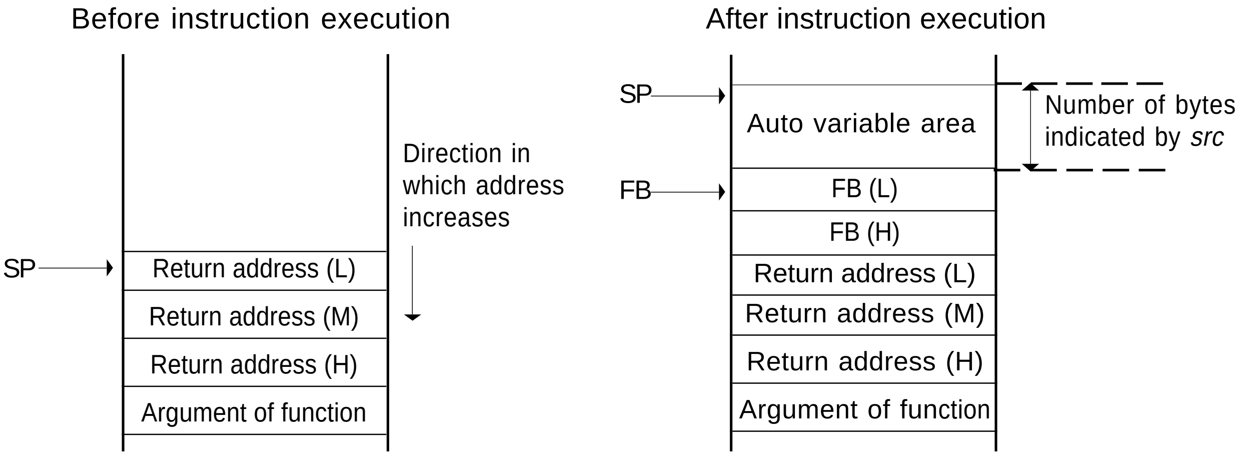
Page= 179

[Operation]

```
SP      ←   SP  –  2
M(SP)   ←   FB
FB      ←   SP
SP      ←   SP  –  src
```

[Function]

- This instruction generates a stack frame. *src* represents the size of the stack frame.
- The diagrams below show the stack area status before and after the ENTER instruction is executed at the beginning of a called subroutine.



[Selectable src]

src
#IMM8

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

ENTER #3

[Related Instructions] EXITD

EXITD

Deallocate stack frame

EXIT and Deallocate stack frame

EXITD

[Syntax]

EXITD

[Instruction Code/Number of Cycles]

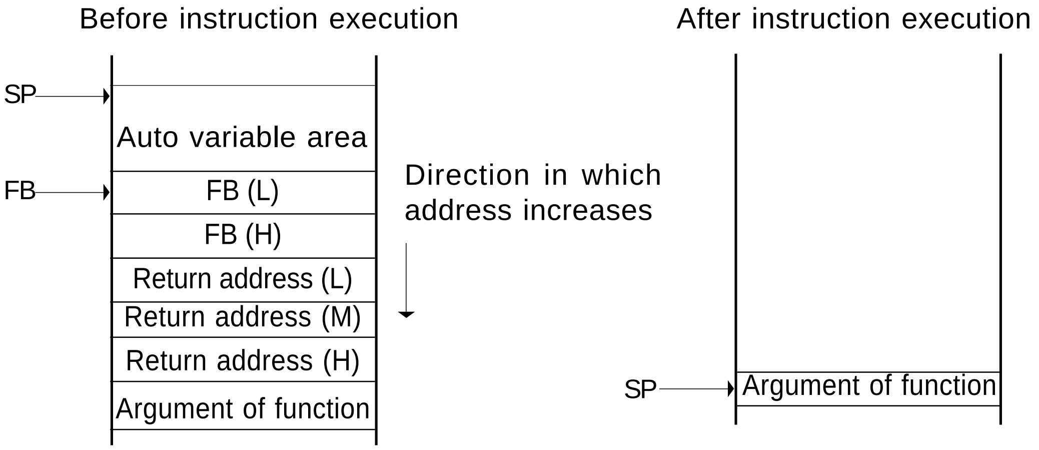
Page= 180

[Operation]

SP	←	FB
FB	←	M(SP)
SP	←	SP + 2
PC_{ML}	←	M(SP)
SP	←	SP + 2
PC_H	←	M(SP)
SP	←	SP + 1

[Function]

- This instruction deallocates the stack frame and exits from the subroutine.
- Use this instruction in combination with the ENTER instruction.
- The diagrams below show the stack area status before and after the EXITD instruction is executed at the end of a subroutine in which an ENTER instruction was executed.



[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

EXITD

[Related Instructions] ENTER

EXTS

Extend sign

EXTend Sign

EXTS

[Syntax]

EXTS.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 180

[Operation]

dest ← EXT(dest)

[Function]

- This instruction sign extends *dest* and stores the result in *dest*.
- If you selected (.B) for the size specifier (.size), *dest* is sign extended to 16 bits.
- If you selected (.W) for the size specifier (.size), R0 is sign extended to 32 bits. In this case, R2 is used for the upper bytes.

[Selectable dest]

dest			
R0L/R0	~~R0H/R1~~	R1L~~/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : If you selected (.B) for the size specifier (.size), the flag is set when the operation resulted in MSB = 1; otherwise cleared. The flag does not change if you selected (.W) for the size specifier (.size).

Z : If you selected (.B) for the size specifier (.size), the flag is set when the operation resulted in 0; otherwise cleared. The flag does not change if you selected (.W) for the size specifier (.size).

[Description Example]

```
EXTS.B    R0L
EXTS.W    R0
```

FCLR *Clear flag register bit* **Flag register CLeaR** FCLR

[Syntax]

FCLR dest

[Instruction Code/Number of Cycles]

Page= 181

[Operation]

dest ← 0

[Function]

- This instruction stores 0 in *dest*.

[Selectable dest]

dest							
C	D	Z	S	B	O	I	U

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*1	*1	*1	*1	*1	*1	*1	*1

*1 The selected flag is cleared to 0.

[Description Example]

```
FCLR    I
FCLR    S
```

[Related Instructions] FSET

FSET *Set flag register bit* **Flag register SET** FSET

[Syntax]

FSET dest

[Instruction Code/Number of Cycles]

Page= 182

[Operation]

dest ← 1

[Function]

- This instruction stores 1 in *dest*.

[Selectable dest]

dest							
C	D	Z	S	B	O	I	U

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*1	*1	*1	*1	*1	*1	*1	*1

*1 The selected flag is set (= 1).

[Description Example]

```
FSET     I
FSET     S
```

[Related Instructions] FCLR

INC *Increment* INCrement INC

[Syntax]

INC.size **dest**
└── **B , W**

[Instruction Code/Number of Cycles]

Page= 182

[Operation]

dest ← dest + 1

[Function]

- This instruction adds 1 to *dest* and stores the result in *dest*.

[Selectable dest]

dest			
R0L*1	R0H*1	dsp:8[SB]*1	dsp:8[FB]*1
abs16*1	A0*2	A1*2	

*1 You can only specify (.B) for the size specifier (.size).

*2 You can only specify (.W) for the size specifier (.size).

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
INC.W    A0
INC.B    R0L
```

[Related Instructions] DEC

INT *Interrupt by INT instruction* **INTerrupt** INT

[Syntax]

INT **src**

[Instruction Code/Number of Cycles]

Page= 183

[Operation]

```
SP      ←   SP  –  2
M(SP)   ←   (PC  +  2)H, FLG
SP      ←   SP  –  2
M(SP)   ←   (PC  +  2)ML
PC      ←   M(IntBase  +  src  ×  4)
```

[Function]

- This instruction generates a software interrupt specified by *src*. *src* represents a software interrupt number.
- If *src* is 31 or smaller, the U flag is cleared to 0 and the interrupt stack pointer (ISP) is used.
- If *src* is 32 or larger, the stack pointer indicated by the U flag is used.
- The interrupts generated by the INT instruction are nonmaskable interrupts.

[Selectable src]

src
#IMM*1*2

*1 #IMM denotes a software interrupt number.

*2 The range of values that can be taken on is 0 $\leq$ #IMM $\leq$ 63.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	○	○	—	—	—	—	○	—

*3 The flags are saved to the stack area before the INT instruction is executed. After the interrupt, the flags change state as shown on the left.

Conditions

U : The flag is cleared if the software interrupt number is 31 or smaller. The flag does not change if the software interrupt number is 32 or larger.

I : The flag is cleared.

D : The flag is cleared.

[Description Example]

```
INT     #0
```

[Related Instructions] BRK,INTO

INTO *Interrupt on overflow* INTerrupt on Overflow INTO

[Syntax]

INTO

[Instruction Code/Number of Cycles]

Page= 184

[Operation]

SP	←	SP – 2
M(SP)	←	(PC + 1)$_H$, FLG
SP	←	SP – 2
M(SP)	←	(PC + 1)$_{ML}$
PC	←	M(FFFE0$_{16}$)

[Function]

- If the O flag is 1, this instruction generates an overflow interrupt. If the flag is 0, the next instruction is executed.
- The overflow interrupt is a nonmaskable interrupt.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	○	○	—	—	—	—	○	—

*1 The flags are saved to the stack area before the INTO instruction is executed. After the interrupt, the flags change state as shown on the left.

Conditions

U : The flag is cleared.

I : The flag is cleared.

D : The flag is cleared.

[Description Example]

INTO

[Related Instructions] BRK,INT

J*Cnd* — *Jump on condition* — Jump on Condition — J*Cnd*

[Syntax]

J*Cnd* label

[Instruction Code/Number of Cycles]

Page= 184

[Operation]

if true then jump label

[Function]

- This instruction causes program flow to branch off after checking the execution result of the preceding instruction against the following condition. If the condition indicated by *Cnd* is true, control jumps to **label**. If false, the next instruction is executed.
- The following conditions can be used for *Cnd*:

Cnd	Condition		Expression	*Cnd*	Condition		Expression
GEU/C	C=1	Equal to or greater than C flag is 1.	≦	LTU/NC	C=0	Smaller than C flag is 0.	>
EQ/Z	Z=1	Equal to Z flag is 1.	=	NE/NZ	Z=0	Not equal Z flag is 0.	≠
GTU	$C\wedge\overline{Z}=1$	Greater than	<	LEU	$C\wedge\overline{Z}=0$	Equal to or smaller than	≧
PZ	S=0	Positive or zero	0≦	N	S=1	Negative	0>
GE	S∀O=0	Equal to or greater than (signed value)	≦	LE	(S∀O)∨Z=1	Equal to or smaller than (signed value)	≧
GT	(S∀O)∨Z=0	Greater than (signed value)	<	LT	S∀O=1	Smaller than (signed value)	>
O	O=1	O flag is 1.		NO	O=0	O flag is 0.	

[Selectable label]

label	*Cnd*
PC^{*1}–127 ≦ label ≦ PC^{*1}+128	GEU/C,GTU,EQ/Z,N,LTU/NC,LEU,NE/NZ,PZ
PC^{*1}–126 ≦ label ≦ PC^{*1}+129	LE,O,GE,GT,NO,LT

*1 PC indicates the start address of the instruction.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JEQ     label
JNE     label
```

[Related Instructions] BM*Cnd*

JMP

Unconditional jump

JuMP

JMP

[Syntax]

JMP(.length) label

└ **S , B , W , A** (Can be specified)

[Instruction Code/Number of Cycles]

Page= 185

[Operation]

PC ← label

[Function]

- This instruction causes control to jump to **label**.

[Selectable label]

.length	label
.S	PC*1+2 ≦ label ≦ PC*1+9
.B	PC*1–127 ≦ label ≦ PC*1+128
.W	PC*1–32767 ≦ label ≦ PC*1+32768
.A	abs20

*1 The PC indicates the start address of the instruction.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JMP        label
```

[Related Instructions] JMPI,JMPS

JMPI

Jump indirect
JuMP Indirect

[Syntax]

JMPI.length src

W , A

[Instruction Code/Number of Cycles]

Page= 187

[Operation]

When jump distance specifier (.length) is (.W)

PC ← PC ± src

When jump distance specifier (.length) is (.A)

PC ← src

[Function]

- This instruction causes control to jump to the address indicated by *src*. If *src* is memory, specify the address at which the low-order address is stored.
- If you selected (.W) for the jump distance specifier (.length), control jumps to the start address of the instruction plus the address indicated by *src* (added including the sign bits). If *src* is memory, the required memory capacity is 2 bytes.
- If *src* is memory when you selected (.A) for the jump distance specifier (.length), the required memory capacity is 3 bytes.

[Selectable src]

If you selected (.W) for the jump distance specifier (.length)

src			
~~R0L/~~R0	~~R0H/~~R1	~~R1L/~~R2	~~R1H/~~R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	dsp:16[SB]	abs16
dsp:20[A0]	dsp:20[A1]	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

If you selected (.A) for the jump distance specifier (.length)

src			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	dsp:16[SB]	abs16
dsp:20[A0]	dsp:20[A1]	~~abs20~~	
R2R0	R3R1	A1A0	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JMPI.A    A1A0
JMPI.W    R0
```

[Related Instructions] JMP,JMPS

JMPS

Jump to special page

JuMP Special page

JMPS

[Syntax]

JMPS src

[Instruction Code/Number of Cycles]

Page= 188

[Operation]

$PC_H \leftarrow 0F_{16}$

$PC_{ML} \leftarrow M(FFFFE_{16} - src \times 2)$

[Function]

- This instruction causes control to jump to the address set in each table of the special page vector table plus $F0000_{16}$. The area across which control can jump is from address $F0000_{16}$ to address $FFFFF_{16}$.
- The special page vector table is allocated to an area from address $FFE00_{16}$ to address $FFFDA_{16}$.
- *src* represents a special page number. The special page number is 255 for address $FFE00_{16}$, and 18 for address $FFFDA_{16}$.

[Selectable src]

src
#IMM*1*2

*1 #IMM denotes a special page number.

*2 The range of values that can be taken on is 18 $\leq$ #IMM $\leq$ 255.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JMPS      #20
```

[Related Instructions] JMP,JMPI

JSR *Subroutine call* Jump SubRoutine JSR

[Syntax]

JSR(.length) label

.length: **W , A** (Can be specified)

[Instruction Code/Number of Cycles]

Page= 189

[Operation]

```
SP        ←   SP  –  1
M(SP)     ←   (PC  +  n)H
SP        ←   SP  –  2
M(SP)     ←   (PC  +  n)ML
PC        ←   label
```

*1 n denotes the number of instruction bytes.

[Function]

- This instruction causes control to jump to a subroutine indicated by **label**.

[Selectable label]

.length	label
.W	PC[*1]–32767 ≦ label ≦ PC[*1]+32768
.A	abs20

*1 The PC indicates the start address of the instruction.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	–	–	–	–	–	–

[Description Example]

```
JSR.W    func
JSR.A    func
```

[Related Instructions] JSRI,JSRS

JSRI

Indirect subroutine call

Jump SubRoutine Indirect

JSRI

[Syntax]

JSRI.length src

W , A

[Instruction Code/Number of Cycles]

Page=190

[Operation]

When jump distance specifier (.length) is (.W)

SP ← SP – 1
M(SP) ← $(PC + n)_H$
SP ← SP – 2
M(SP) ← $(PC + n)_{ML}$
PC ← PC ± src

*1 n denotes the number of instruction bytes.

When jump distance specifier (.length) is (.A)

SP ← SP – 1
M(SP) ← $(PC + n)_H$
SP ← SP – 2
M(SP) ← $(PC + n)_H$
PC ← src

[Function]

- This instruction causes control to jump to a subroutine at the address indicated by *src*. If *src* is memory, specify the address at which the low-order address is stored.
- If you selected (.W) for the jump distance specifier (.length), control jumps to a subroutine at the start address of the instruction plus the address indicated by *src* (added including the sign bits). If *src* is memory, the required memory capacity is 2 bytes.
- If *src* is memory when you selected (.A) for the jump distance specifier (.length), the required memory capacity is 3 bytes.

[Selectable src]

If you selected (.W) for the jump distance specifier (.length)

src			
~~R0L/~~R0	~~R0H/~~R1	~~R1L/~~R2	~~R1H/~~R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	dsp:16[SB]	abs16
dsp:20[A0]	dsp:20[A1]	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

If you selected (.A) for the jump distance specifier (.length)

src			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	dsp:16[SB]	abs16
dsp:20[A0]	dsp:20[A1]	~~abs20~~	
R2R0	R3R1	A1A0	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JSRI.A    A1A0
JSRI.W    R0
```

[Related Instructions] JSR,JSRS

JSRS

Special page subroutine call

Jump SubRoutine Special page

JSRS

[Syntax]

JSRS src

[Instruction Code/Number of Cycles]

Page= 191

[Operation]

SP	←	SP – 1
M(SP)	←	$(PC + 2)_H$
SP	←	SP – 2
M(SP)	←	$(PC + 2)_{ML}$
PC_H	←	$0F_{16}$
PC_{ML}	←	M ($FFFFE_{16}$ – src × 2)

[Function]

- This instruction causes control to jump to a subroutine at the address set in each table of the special page vector table plus $F0000_{16}$. The area across which program flow can jump to a subroutine is from address $F0000_{16}$ to address $FFFFF_{16}$.
- The special page vector table is allocated to an area from address $FFE00_{16}$ to address $FFFDA_{16}$.
- *src* represents a special page number. The special page number is 255 for address $FFE00_{16}$, and 18 for address $FFFDA_{16}$.

[Selectable src]

src
#IMM[*1*2]

[*1] #IMM denotes a special page number.

[*2] The range of values that can be taken on is 18 ≤ #IMM ≤ 255.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
JSRS     #18
```

[Related Instructions] JSR,JSRI

LDC

Transfer to control register

LoaD Control register

LDC

[Syntax]

LDC src,dest

[Instruction Code/Number of Cycles]

Page= 191

[Operation]

dest ← src

[Function]

- This instruction transfers *src* to the control register indicated by *dest*. If *src* is memory, the required memory capacity is 2 bytes.
- If the destination is INTBL or INTBH, make sure that bytes are transferred in succession.
- No interrupt requests are accepted immediately after this instruction.

[Selectable src/dest]

src				dest			
~~R0L/~~R0	~~R0H/~~R1	~~R1L/~~R2	~~R1H/~~R3	FB	SB	SP*1	ISP
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]	FLG	INTBH	INTBL	
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]				
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16				
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM				
~~R2R0~~	~~R3R1~~	~~A1A0~~					

*1 Operation is performed on the stack pointer indicated by the U flag.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*2	*2	*2	*2	*2	*2	*2	*2

*2 The flag changes only when *dest* is FLG.

[Description Example]

```
LDC     R0,SB
LDC     A0,FB
```

[Related Instructions] POPC,PUSHC,STC,LDINTB

LDCTX

Restore context

LoaD ConTeXt

LDCTX

[Syntax]

LDCTX abs16,abs20

[Instruction Code/Number of Cycles]

Page=192

[Function]

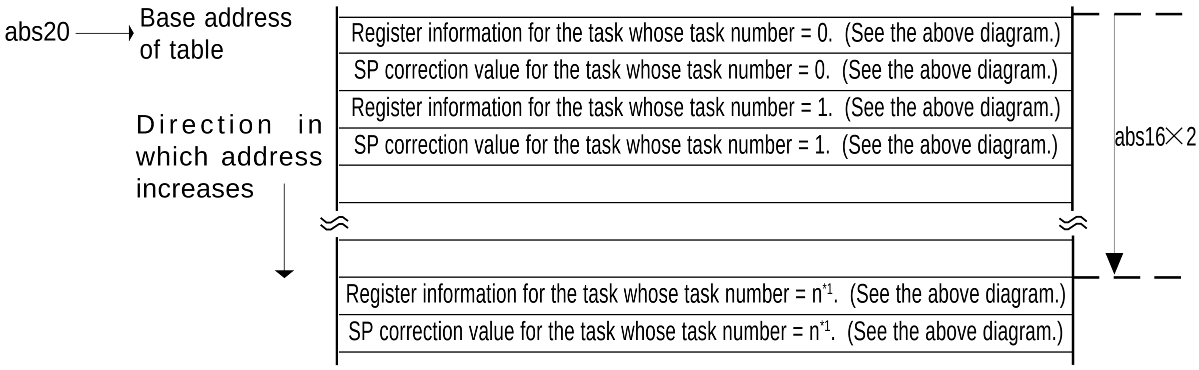
- This instruction restores task context from the stack area.
- Set the RAM address that contains the task number in abs16 and the start address of table data in abs20.
- The required register information is specified from table data by the task number and the data in the stack area is transferred to each register according to the specified register information. Then the SP correction value is added to the stack pointer (SP). For this SP correction value, set the number of bytes you want to the transferred.
- Information on transferred registers is configured as shown below. Logic 1 indicates a register to be transferred and logic 0 indicates a register that is not transferred.

MSB LSB

FB	SB	A1	A0	R3	R2	R1	R0

Transferred sequentially beginning with R0

- The table data is comprised as shown below.The address indicated by abs20 is the base address of the table. The data stored at an address apart from the base address as much as twice the content of abs16 indicates register information, and the next address contains the stack pointer correction value.



abs20 → Base address of table

Direction in which address increases

Register information for the task whose task number = 0. (See the above diagram.)

SP correction value for the task whose task number = 0. (See the above diagram.)

Register information for the task whose task number = 1. (See the above diagram.)

SP correction value for the task whose task number = 1. (See the above diagram.)

Register information for the task whose task number = n*1. (See the above diagram.)

SP correction value for the task whose task number = n*1. (See the above diagram.)

*1 n=0 to 255

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

LDCTX Ram,Rom_TBL

[Related Instructions] STCTX

LDE

Transfer from extended data area

LoaD from EXtra far data area

LDE

[Syntax]

LDE.size **src,dest**

B , W

[Instruction Code/Number of Cycles]

Page=193

[Operation]

dest ← src

[Function]

- This instruction transfers *src* from extended area to *dest.*
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to transfer data in 16 bits.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0	A1/A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
dsp:20[A0]	~~dsp:20[A1]~~	abs20	~~#IMM~~	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	[A1A0]	~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the transfer resulted in MSB of *dest* = 1; otherwise cleared.

Z : The flag is set when the transfer resulted in *dest* = 0; otherwise cleared.

[Description Example]

```
LDE.W     [A1A0],R0
LDE.B     Rom_TBL,A0
```

[Related Instructions] STE,MOV,XCHG

LDINTB

Transfer to INTB register

LoaD INTB register

LDINTB

[Syntax]

LDINTB src

[Instruction Code/Number of Cycles]

Page=194

[Operation]

INTBHL ← src

[Function]

- This instruction transfers *src* to INTB.
- The LDINTB instruction is a macro-instruction consisting of the following:

```
LDC     #IMM, INTBH
LDC     #IMM, INTBL
```

[Selectable src]

src
#IMM20

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
LDINTB     #0F0000H
```

[Related Instructions] LDC,STC,PUSHC,POPC

LDIPL

Set interrupt enable level

LoaD Interrupt Permission Level

LDIPL

[Syntax]

LDIPL src

[Instruction Code/Number of Cycles]

Page= 195

[Operation]

IPL ← src

[Function]

- This instruction transfers *src* to IPL.

[Selectable src]

src
#IMM[*1]

*1 The range of values that can be taken on is 0 ≤ #IMM ≤ 7

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
LDIPL       #2
```

MOV *Transfer* MOVe MOV

[Syntax]

MOV.size (:format) src,dest

(:format) — **G , Q , Z , S** (Can be specified)

size — **B , W**

[Instruction Code/Number of Cycles]

Page= 195

[Operation]

dest ← src

[Function]

- This instruction transfers *src* to *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to transfer data in 16 bits. If *src* is an A0 or A1, data is transferred from the 8 low-order bits of A0 or A1.

[Selectable src/dest] (See the next page for *src*/*dest* classified by format.)

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM[*2]	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP][*3]	~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP][*2 *3]

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

*2 If *src* is #IMM, you cannot choose dsp:8 [SP] for *dest*.

*3 Operation is performed on the stack pointer indicated by the U flag. You cannot choose dsp:8 [SP] for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the transfer resulted in MSB of *dest* = 1; otherwise cleared.

Z : The flag is set when the transfer resulted in 0; otherwise cleared.

[Description Example]

```
MOV.B:S    #0ABH,R0L
MOV.W      #–1,R2
```

[Related Instructions] LDE,STE,XCHG

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*2	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]*3	~~R2R0~~	~~R3R1~~	~~A1A0~~	dsp:8[SP]*2*3

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.
*2 If *src* is #IMM, you cannot choose dsp:8 [SP] for *dest*.
*3 Operation is performed on the stack pointer indicated by the U flag. You cannot choose dsp:8 [SP] for *src* and *dest* simultaneously.

Q format

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0	A1/A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*4	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*4 The range of values that can be taken on is –8 ≤ #IMM ≤ +7.

S format

src				dest			
R0L*5*6*7	R0H*5*6*8	dsp:8[SB]*5	dsp:8[FB]*5	R0L*5*6	R0H*5*6		
abs16*5	~~#IMM~~			~~abs16~~	A0*5*8	A1*5*7	
R0L*5*6	R0H*5*6	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L*5*6	R0H*5*6	dsp:8[SB]*5	dsp:8[FB]*5
~~abs16~~	~~#IMM~~			abs16*5	~~A0~~	~~A1~~	
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L*5	R0H*5	dsp:8[SB]*5	dsp:8[FB]*5
~~abs16~~	#IMM*9			abs16*5	A0*9	A1*9	

*5 You can only specify (.B) for the size specifier (.size).
*6 You cannot choose the same register for *src* and *dest*.
*7 If *src* is R0L, you can only choose A1 for *dest* as the address register.
*8 If *src* is R0H, you can only choose A0 for *dest* as the address register.
*9 You can specify (.B) and (.W) for the size specifier (.size).

Z format

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#0			abs16	~~A0~~	~~A1~~	

MOVA

Transfer effective address

MOVe effective Address

MOVA

[Syntax]

MOVA src,dest

[Instruction Code/Number of Cycles]

Page= 202

[Operation]

dest ← EVA(src)

[Function]

- This instruction transfers the affective address of *src* to *dest*.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	~~R0L/~~R0	~~R0H/~~R1	~~R1L/~~R2	~~R1H/~~R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0	~~A1/~~A1	~~[A0]~~	~~[A1]~~
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~#IMM~~	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

MOVA Ram:16[SB],A0

[Related Instructions] PUSHA

MOV*Dir*

Transfer 4-bit data

MOVe nibble

MOV*Dir*

[Syntax]

MOV*Dir* src,dest

[Instruction Code/Number of Cycles]

Page= 203

[Operation]

Dir	Operation		
HH	H4:dest	←	H4:src
HL	L4:dest	←	H4:src
LH	H4:dest	←	L4:src
LL	L4:dest	←	L4:src

[Function]

- Be sure to choose R0L for either *src* or *dest*.

Dir	Function
HH	Transfers src’s 4 high-order bits to dest’s 4 high-order bits.
HL	Transfers src’s 4 high-order bits to dest’s 4 low-order bits.
LH	Transfers src’s 4 low-order bits to dest’s 4 high-order bits.
LL	Transfers src’s 4 low-order bits to dest’s 4 low-order bits.

[Selectable src/dest]

src				dest			
R0L~~/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~	R0L~~/R0~~	R0H~~/R1~~	R1L~~/R2~~	R1H~~/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0~~	~~A1/A1~~	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~#IMM~~	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	
R0L~~/R0~~	R0H~~/R1~~	R1L~~/R2~~	R1H~~/R3~~	R0L~~/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	[A0]	[A1]	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~#IMM~~	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
MOVHH    R0L,[A0]
MOVHL    R0L,[A0]
```

MUL

Signed multiply
MULtiple

MUL

[Syntax]

MUL.size src,dest
B , W

[Instruction Code/Number of Cycles]

Page= 205

[Operation]

dest ← dest × src

[Function]

- This instruction multiplies *src* and *dest* together including the sign bits and stores the result in *dest*.
- If you selected (.B) for the size specifier (.size), *src* and *dest* both are operated on in 8 bits and the result is stored in 16 bits. If you specified an A0 or A1 for either *src* or *dest*, operation is performed on the 8 low-order bits of A0 or A1.
- If you selected (.W) for the size specifier (.size), *src* and *dest* both are operated on in 16 bits and the result is stored in 32 bits. If you specified R0, R1, or A0 for *dest*, the result is stored in R2R0, R3R1, or A1A0 accordingly.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	~~R0H/~~R1	R1L~~/R2~~	~~R1H/R3~~
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	~~A1/A1~~	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[*1] If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
MUL.B    A0,R0L          ; R0L and A0's 8 low-order bits are multiplied.
MUL.W    #3,R0
MUL.B    R0L,R1L
MUL.W    A0,Ram
```

[Related Instructions] DIV,DIVU,DIVX,MULU

MULU *Unsigned multiply* MULtiple Unsigned MULU

[Syntax]

MULU.size src,dest

└ **B , W**

[Instruction Code/Number of Cycles]

Page= 207

[Operation]

dest ← dest × src

[Function]

- This instruction multiplies *src* and *dest* together not including the sign bits and stores the result in *dest*.
- If you selected (.B) for the size specifier (.size), *src* and *dest* both are operated on in 8 bits and the result is stored in 16 bits. If you specified an A0 or A1 for either *src* or *dest*, operation is performed on the 8 low-order bits of A0 or A1.
- If you selected (.W) for the size specifier (.size), *src* and *dest* both are operated on in 16 bits and the result is stored in 32 bits. If you specified R0, R1, or A0 for *dest*, the result is stored in R2R0, R3R1, or A1A0 accordingly.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	~~R0H/~~R1	R1L~~/R2~~	~~R1H/R3~~
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	~~A1/A1~~	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
MULU.B    A0,R0L                    ; R0L and A0's 8 low-order bits are multiplied.
MULU.W    #3,R0
MULU.B    R0L,R1L
MULU.W    A0,Ram
```

[Related Instructions] DIV,DIVU,DIVX,MUL

NEG *Two's complement* **NEGate** NEG

[Syntax]

NEG.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 209

[Operation]

dest ← 0 – dest

[Function]

- This instruction takes the 2's complement of *dest* and stores the result in *dest.*

[Selectable dest]

dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	○	○	—	○

Conditions

O : The flag is set when *dest* before the operation is –128 (.B) or –32768 (.W); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
NEG.B     R0L
NEG.W     A1
```

[Related Instructions] NOT

NOP

No operation

No OPeration

NOP

[Syntax]

NOP

[Instruction Code/Number of Cycles]

Page= 209

[Operation]

PC ← PC + 1

[Function]

- This instruction adds 1 to PC.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

NOP

NOT

Invert all bits

NOT

NOT

[Syntax]

NOT.size (:format) dest

G , S (Can be specified)

B , W

[Instruction Code/Number of Cycles]

Page= 210

[Operation]

dest ← $\overline{\text{dest}}$

[Function]

- This instruction inverts *dest* and stores the result in *dest*.

[Selectable dest]

dest			
R0L[*1]/R0	R0H[*1]/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB][*1]	dsp:8[FB][*1]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16[*1]
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 Can be selected in G and S formats.
In other cases, *dest* can be selected in G format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
NOT.B    R0L
NOT.W    A1
```

[Related Instructions] NEG

OR *Logically OR* OR OR

[Syntax]

OR.size (:format) src,dest

G , S (Can be specified)

B , W

[Instruction Code/Number of Cycles]

Page= 211

[Operation]

dest ← src ∨ dest

[Function]

- This instruction logically ORs *dest* and *src* together and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest] (See the next page for *src*/*dest* classified by format.)

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
OR.B      Ram:8[SB],R0L
OR.B:G    A0,R0L                 ; A0's 8 low-order bits and R0L are ORed.
OR.B:G    R0L,A0                 ; R0L is zero-expanded and ORed with A0.
OR.B:S    #3,R0L
```

[Related Instructions] AND,XOR,TST

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

S format*2

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#IMM			abs16	~~A0~~	~~A1~~	
R0L*3	R0H*3	dsp:8[SB]	dsp:8[FB]	R0L*3	R0H*3	~~dsp:8[SB]~~	~~dsp:8[FB]~~
abs16	~~#IMM~~			~~abs16~~	~~A0~~	~~A1~~	

*2 You can only specify (.B) for the size specifier (.size).

*3 You cannot choose the same register for *src* and *dest*.

POP *Restore register/memory* POP POP

[Syntax] **[Instruction Code/Number of Cycles]**

POP.size (:format) dest Page= 213

G , S (Can be specified)

B , W

[Operation]

If the size specifier (.size) is (.B)			If the size specifier (.size) is (.W)		
dest	←	M(SP)	dest	←	M(SP)
SP	←	SP + 1	SP	←	SP + 2

[Function]

- This instruction restores *dest* from the stack area.

[Selectable dest]

dest			
R0L[*1]/R0	R0H[*1]/R1	R1L/R2	R1H/R3
~~A0/~~A0[*1]	~~A1/~~A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[*1] Can be selected in G and S formats.

In other cases, *dest* can be selected in G format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
POP.B    R0L
POP.W    A0
```

[Related Instructions] PUSH,POPM,PUSHM

POPC *Restore control register* POP Control register POPC

[Syntax]

POPC dest

[Instruction Code/Number of Cycles]

Page= 215

[Operation]

dest ← M(SP)

SP[*1] ← SP + 2

*1 When *dest* is SP or when the U flag = “0” and *dest* is ISP, the value 2 is not added to SP.

[Function]

- This instruction restores from the stack area to the control register indicated by *dest*.
- When restoring the interrupt table register, always be sure to restore INTBH and INTBL in succession.
- No interrupt requests are accepted immediately after this instruction.

[Selectable dest]

dest
FB SB SP[*2] ISP FLG INTBH INTBL

*2 Operation is performed on the stack pointer indicated by the U flag.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*3	*3	*3	*3	*3	*3	*3	*3

*3 The flag changes only when *dest* is FLG.

[Description Example]

```
POPC        SB
```

[Related Instructions] PUSHC,LDC,STC,LDINTB

POPM

Restore multiple registers

POP Multiple

POPM

[Syntax]

POPM dest

[Instruction Code/Number of Cycles]

Page= 215

[Operation]

dest ← M(SP)

SP ← SP + N*1 × 2

*1 Number of registers to be restored

[Function]

- This instruction restores the registers selected by *dest* collectively from the stack area.
- Registers are restored from the stack area in the following order:

FB	**SB**	**A1**	**A0**	**R3**	**R2**	**R1**	**R0**

◄———

Restored sequentially beginning with R0

[Selectable dest]

dest*2
R0 R1 R2 R3 A0 A1 SB FB

*2 You can choose multiple *dest*.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

POPM R0,R1,A0,SB,FB

[Related Instructions] POP,PUSH,PUSHM

PUSH

Save register/memory/immediate data

PUSH

PUSH

[Syntax]

PUSH.size (:format) src

G , S (Can be specified)

B , W

[Instruction Code/Number of Cycles]

Page= 216

[Operation]

If the size specifier (.size) is (.B)

SP ← SP – 1

M(SP) ← src

If the size specifier (.size) is (.W)

SP ← SP – 2

M(SP) ← src

[Function]

- This instruction saves *src* to the stack area.

[Selectable src]

src			
R0L[*1]/R0	R0H[*1]/R1	R1L/R2	R1H/R3
~~A0/~~A0[*1]	~~A1/~~A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM
~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 Can be selected in G and S formats.

In other cases, *dest* can be selected in G format.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
PUSH.B    #5
PUSH.W    #100H
PUSH.B    R0L
PUSH.W    A0
```

[Related Instructions] POP,POPM,PUSHM

PUSHA

Save effective address

PUSH effective Address

PUSHA

[Syntax]

PUSHA src

[Instruction Code/Number of Cycles]

Page= 218

[Operation]

SP ← SP – 2

M(SP) ← EVA(src)

[Function]

- This instruction saves the effective address of *src* to the stack area.

[Selectable src]

src			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
PUSHA   Ram:8[FB]
PUSHA   Ram:16[SB]
```

[Related Instructions] MOVA

PUSHC *Save control register* **PUSH Control register** PUSHC

[Syntax]

PUSHC src

[Instruction Code/Number of Cycles]

Page= 218

[Operation]

SP ← SP – 2

M(SP) ← src*1

*1 When *src* is SP or when the U flag = “0” and *src* is ISP, the SP before being subtracted by 2 is saved.

[Function]

- This instruction saves the control register indicated by *src* to the stack area.

[Selectable src]

src						
FB	SB	SP*2	ISP	FLG	INTBH	INTBL

*2 Operation is performed on the stack pointer indicated by the U flag.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

PUSHC SB

[Related Instructions] POPC,LDC,STC,LDINTB

PUSHM

Save multiple registers

PUSH Multiple

PUSHM

[Syntax]

PUSHM src

[Instruction Code/Number of Cycles]

Page= 219

[Operation]

SP ← SP – N*1 × 2

M(SP) ← src

*1 Number of registers saved.

[Function]

- This instruction saves the registers selected by *src* collectively to the stack area.
- The registers are saved to the stack area in the following order:

R0	**R1**	**R2**	**R3**	**A0**	**A1**	**SB**	**FB**

Saved sequentially beginning with FB

[Selectable src]

src*2
R0 R1 R2 R3 A0 A1 SB FB

*2 You can choose multiple *src*.

[Flag Change]

Flag	**U**	**I**	**O**	**B**	**S**	**Z**	**D**	**C**
Change	—	—	—	—	—	—	—	—

[Description Example]

```
PUSHM    R0,R1,A0,SB,FB
```

[Related Instructions] POP,PUSH,POPM

REIT

Return from interrupt

REturn from InTerrupt

REIT

[Syntax]

REIT

[Instruction Code/Number of Cycles]

Page= 219

[Operation]

```
PCML      ←   M(SP)
SP        ←   SP  +  2
PCH, FLG  ←   M(SP)
SP        ←   SP  +  2
```

[Function]

- This instruction restores the PC and FLG that were saved when an interrupt request was accepted to return from the interrupt handler routine.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*1	*1	*1	*1	*1	*1	*1	*1

*1 The flags are reset to the previous FLG state before the interrupt request was accepted.

[Description Example]

REIT

RMPA *Calculate sum-of-products* Repeat MultiPle & Addition RMPA

[Syntax]

RMPA.size — **B , W**

[Instruction Code/Number of Cycles]

Page= 220

[Operation][*1]

```
Repeat
            R2R0(R0)*2  ←   R2R0(R0)*2  +  M(A0)  ×  M(A1)
            A0          ←   A0  +  2 (1)*2
            A1          ←   A1  +  2 (1)*2
            R3          ←   R3  –  1
Until       R3 = 0
```

*1 If you set a value 0 in R3, this instruction is ingored.

*2 Shown in ( )*2 applies when (.B) is selected for the size specifier (.size).

[Function]

- This instruction performs sum-of-product calculations, with the multiplicand address indicated by A0, the multiplier address indicated by A1, and the count of operation indicated by R3. Calculations are performed including the sign bits and the result is stored in R2R0 (R0)*1.
- If an overflow occurs during operation, the O flag is set to terminate the operation. R2R0 (R0)*1 contains the result of the addition performed last. A0, A1 and R3 are indeterminate.
- The content of the A0 or A1 when the instruction is completed indicates the next address of the last-read data.
- If an interrupt request is received during instruction execution, the interrupt is acknowledged after a sum-of-product addition is completed (i.e., after the content of R3 is decremented by 1).
- Make sure that R2R0 (R0)*1 has the initial value set.

Shown in ( )*1 applies when (.B) is selected for the size specifier (.size).

[Fl ag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	—	—	—	—

Conditions

O : The flag is set when +2147483647 (.W) or –2147483648 (.W), or +32767 (.B) or –32768 (.B) is exceeded during operation; otherwise cleared.

[Description Example]

```
RMPA.B
```

ROLC

Rotate left with carry

ROtate to Left with Carry

ROLC

[Syntax]

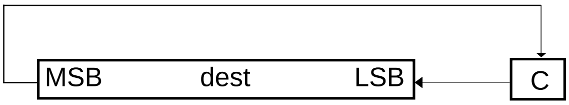
ROLC.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 220

[Operation]



[Function]

- This instruction rotates *dest* one bit to the left including the C flag.

[Selectable dest]

dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in *dest* = 0; otherwise cleared.

C : The flag is set when the shifted-out bit is 1; otherwise cleared.

[Description Example]

```
ROLC.B     R0L
ROLC.W     R0
```

[Related Instructions] RORC,ROT,SHA,SHL

RORC *Rotate right with carry* ROtate to Right with Carry RORC

[Syntax]

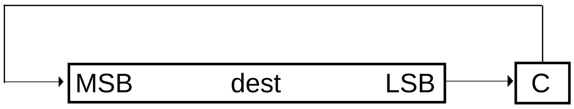
RORC.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 221

[Operation]



[Function]

- This instruction rotates *dest* one bit to the right including the C flag.

[Selectable dest]

dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in *dest* = 0; otherwise cleared.

C : The flag is set when the shifted-out bit is 1; otherwise cleared.

[Description Example]

```
RORC.B     R0L
RORC.W     R0
```

[Related Instructions] ROLC,ROT,SHA,SHL

ROT

Rotate
ROTate

ROT

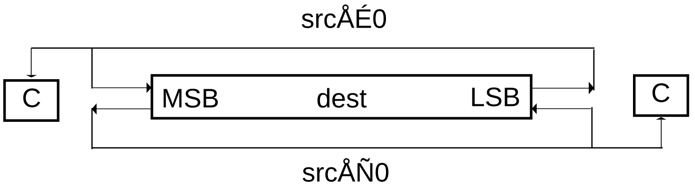
[Syntax]

ROT.size src,dest
└── **B , W**

[Instruction Code/Number of Cycles]

Page= 222

[Operation]



[Function]

- This instruction rotates *dest* left or right the number of bits indicated by *src*. The bit overflowing from LSB (MSB) is transferred to MSB(LSB) and the C flag.
- The direction of rotate is determined by the sign of *src*. If *src* is positive, bits are rotated left; if negative, bits are rotated right.
- If *src* is an immediate, the number of rotates is –8 to –1 and +1 to +8. You cannot set values less than –8, equal to 0, or greater than +8.
- If *src* is a register and you selected (.B) for the size specifier (.size), the number of rotates is –8 to +8. Although you can set 0, no bits are rotated and no flags are changed. If you set a value less than –8 or greater than +8, the result of rotation is indeterminate.
- If *src* is a register and you selected (.W) for the size specifier (.size), the number of rotates is –16 to +16. Although you can set 0, no bits are rotated and no flags are changed. If you set a value less than –16 or greater than +16, the result of rotation is indeterminate.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	R1H*1~~/R3~~	R0L/R0	R0H/R1*1	R1L/R2	R1H/R3*1
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*2	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If *src* is R1H, you cannot choose R1 or R1H for *dest*.

*2 The range of values that can be taken on is –8 ≤ #IMM ≤ +8. However, you cannot set 0.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

*1 If the number of rotates is 0, no flags are changed.

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when the bit shifted out last is 1; otherwise cleared.

[Description Example]

```
ROT.B    #1,R0L          ; Rotated left
ROT.B    #–1,R0L         ; Rotated right
ROT.W    R1H,R2
```

[Related Instructions] ROLC,RORC,SHA,SHL

RTS *Return from subroutine* **ReTurn from Subroutine** RTS

[Syntax]

RTS

[Instruction Code/Number of Cycles]

Page= 223

[Operation]

PC_{ML}	←	M(SP)
SP	←	SP + 2
PC_H	←	M(SP)
SP	←	SP + 1

[Function]

- This instruction causes control to return from a subroutine.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

RTS

SBB *Subtract with borrow* SuBtract with Borrow SBB

[Syntax]

SBB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 224

[Operation]

dest ← dest – src – $\overline{C}$

[Function]

- This instruction subtracts *src* and inverted C flag from *dest* and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[*1] If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
	–	–	○	–	○	○	–	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W), or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

```
SBB.B   #2,R0L
SBB.W   A0,R0
SBB.B   A0,R0L      ; A0’s 8 low-order bits and R0L are operated on.
SBB.B   R0L,A0      ; R0L is zero-expanded and operated with A0.
```

[Related Instructions] ADC,ADCF,ADD,SUB

SBJNZ

Subtract & conditional jump

SuBtract then Jump on Not Zero

SBJNZ

[Syntax]

SBJNZ.size src,dest,label

B , W

[Instruction Code/Number of Cycles]

Page= 226

[Operation]

dest ← dest – src

if dest ≠ 0 **then** jump label

[Function]

- This instruction subtracts *src* from *dest* and stores the result in *dest*.
- If the operation resulted in any value other than 0, control jumps to **label**. If the operation resulted in 0, the next instruction is executed.
- The op-code of this instruction is the same as that of ADJNZ.

[Selectable src/dest/label]

src	dest			label
#IMM*1	R0L/R0	R0H/R1	R1L/R2	PC*2–126 ≤ label ≤ PC*2+129
	R1H/R3	~~A0/~~A0	~~A1/~~A1	
	[A0]	[A1]	dsp:8[A0]	
	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	
	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	
	abs16			

*1 The range of values that can be taken on is –7 ≤ #IMM ≤ +8.

*2 The PC indicates the start address of the instruction.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
SBJNZ.W    #1,R0,label
```

[Related Instructions] ADJNZ

SHA

Shift arithmetic
SHift Arithmetic

SHA

[Syntax]

[Instruction Code/Number of Cycles]

SHA.size src,dest

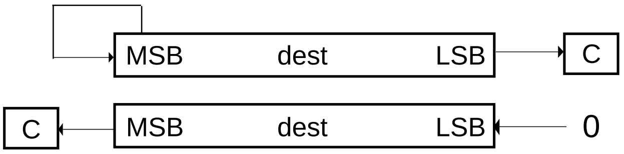
Page= 227

B , W , L

[Operation]

When *src* < 0

When *src* > 0



[Function]

overflowing from LSB (MSB) is transferred to the C flag.

- The direction of shift is determined by the sign of *src*. If *src* is positive, bits are shifted left; if negative, bits are shifted right.
- If *src* is an immediate, the number of shifts is –8 to –1 and +1 to +8. You cannot set values less than –8, equal to 0, or greater than +8.
- If *src* is a register and you selected (.B) for the size specifier (.size), the number of shifts is –8 to +8. Although you can set 0, no bits are shifted and no flags are changed. If you set a value less than –8 or greater than +8, the result of shift is indeterminate.
- If *src* is a register and you selected (.W) or (.L) for the size specifier (.size), the number of shifts is –16 to +16. Although you can set 0, no bits are shifted and no flags are changed. If you set a value less than –16 or greater than +16, the result of shift is indeterminate.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	R1H*1~~/R3~~	R0L/R0	R0H/R1*1	R1L/R2	R1H/R3*1
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM*2	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		R2R0*3	R3R1*3	~~A1A0~~	

*1 If *src* is R1H, you cannot choose R1 or R1H for *dest*.
*2 The range of values that can be taken on is –8 ≤ #IMM ≤ +8. However, you cannot set 0.
*3 You can only specify (.L) for the size specifier (.size). For other *dest*, you can specify (.B) or (.W).

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	○	○	—	○

*1 If the number of shifts is 0, no flags are changed.

Conditions

O : The flag is set when the operation resulted in MSB changing its state from 1 to 0 or from 0 to 1; otherwise cleared. However, the flag does not change if you selected (.L) for the size specifier (.size).
S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z : The flag is set when the operation resulted in 0; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).
C : The flag is set when the bit shifted out last is 1; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).

[Description Example]

```
SHA.B    #3,R0L          ; Arithmetically shifted left
SHA.B    #–3,R0L         ; Arithmetically shifted right
SHA.L    R1H,R2R0
```

[Related Instructions] ROLC,RORC,ROT,SHL

SHL *Shift logical* SHift Logical SHL

[Syntax]

SHL.size src,dest

B , W , L

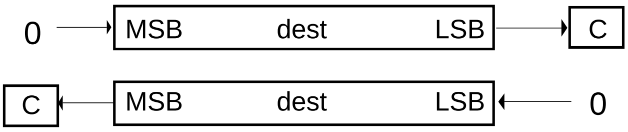
[Instruction Code/Number of Cycles]

Page= 230

[Operation]

When *src* < 0

When *src* > 0



[Function]

- This instruction logically shifts *dest* left or right the number of bits indicated by *src*. The bit overflowing from LSB (MSB) is transferred to the C flag.
- The direction of shift is determined by the sign of *src*. If *src* is positive, bits are shifted left; if negative, bits are shifted right.
- If *src* is an immediate, the number of shifts is –8 to –1 and +1 to +8. You cannot set values less than –8, equal to 0, or greater than +8.
- If *src* is a register and you selected (.B) for the size specifier (.size), the number of shifts is –8 to +8. Although you can set 0, no bits are shifted and no flags are changed. If you set a value less than –8 or greater than +8, the result of shift is indeterminate.
- If *src* is a register and you selected (.W) or (.L) for the size specifier (.size), the number of shifts is –16 to +16. Although you can set 0, no bits are shifted and no flags are changed. If you set a value less than –16 or greater than +16, the result of shift is indeterminate.

[Selectable src/dest]

src				dest			
~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	R1H[*1]~~/R3~~	R0L/R0	R0H/R1[*1]	R1L/R2	R1H/R3[*1]
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM[*2]	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		R2R0[*3]	R3R1[*3]	~~A1A0~~	

[*1] If *src* is R1H, you cannot choose R1 or R1H for *dest*.

[*2] The range of values that can be taken on is –8 ≤ #IMM ≤ +8. However, you cannot set 0.

[*3] You can only specify (.L) for the size specifier (.size). For other *dest*, you can specify (.B) or (.W).

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	○

[*1] If the number of shifts is 0, no flags are changed.

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).

C : The flag is set when the bit shifted out last is 1; otherwise cleared. However, the flag is indeterminate if you selected (.L) for the size specifier (.size).

[Description Example]

```
SHL.B     #3,R0L                          ; Logically shifted left
SHL.B     #–3,R0L                         ; Logically shifted right
SHL.L     R1H,R2R0
```

[Related Instructions] ROLC,RORC,ROT,SHA

SMOVB

Transfer string backward

String MOVe Backward

[Syntax]

SMOVB.size

└ **B , W**

[Instruction Code/Number of Cycles]

Page= 232

[Operation][*1]

When size specifier (.size) is (.B)

```
Repeat
    M(A1) ←  M(2^16 × R1H + A0)
    A0*2  ←  A0 – 1
    A1    ←  A1 – 1
    R3    ←  R3 – 1
Until        R3 = 0
```

When size specifier (.size) is (.W)

```
Repeat
    M(A1) ←  M(2^16 × R1H + A0)
    A0*2  ←  A0 – 2
    A1    ←  A1 – 2
    R3    ←  R3 – 1
Until        R3 = 0
```

*1 If you set a value 0 in R3, this instruction is ingored.

*2 If A0 underflows, the content of R1H is decremented by 1.

[Function]

- This instruction transfers string in successively address decrementing direction from the source address indicated by 20 bits to the destination address indicated by 16 bits.
- Set the 4 high-order bits of the source address in R1H, the 16 low-order bits of the source address in A0, the destination address in A1, and the transfer count in R3.
- The A0 or A1 when the instruction is completed contains the next address of the last-read data.
- If an interrupt request is received during instruction execution, the interrupt is acknowledged after one data transfer is completed.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	–	–	–	–	–	–	–	–

[Description Example]

SMOVB.B

[Related Instructions] SMOVF,SSTR

SMOVF *Transfer string forward* String MOVe Forward SMOVF

[Syntax]

SMOVF.size

B , W

[Instruction Code/Number of Cycles]

Page= 233

[Operation][*1]

When size specifier (.size) is (.B)

```
Repeat
    M(A1)    ←  M($2^{16}$ × R1H + A0)
    A0[*2*]  ←  A0  +  1
    A1       ←  A1  +  1
    R3       ←  R3  –  1
Until           R3 = 0
```

When size specifier (.size) is (.W)

```
Repeat
    M(A1)    ←  M($2^{16}$ × R1H + A0)
    A0[*2]   ←  A0  +  2
    A1       ←  A1  +  2
    R3       ←  R3  –  1
Until           R3 = 0
```

[*1] If you set a value 0 in R3, this instruction is ingored.

[*2] If A0 overflows, the content of R1H is incremented by 1.

[Function]

- This instruction transfers string in successively address incrementing direction from the source address indicated by 20 bits to the destination address indicated by 16 bits.
- Set the 4 high-order bits of the source address in R1H, the 16 low-order bits of the source address in A0, the destination address in A1, and the transfer count in R3.
- The A0 or A1 when the instruction is completed contains the next address of the last-read data.
- If an interrupt request is received during instruction execution, the interrupt is acknowledged after one data transfer is completed.
- This instruction arithmetically shifts *dest* left or right the number of bits indicated by *src*. The bit

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

SMOVF.W

[Related Instructions] SMOVB,SSTR

SSTR

Store string

String SToRe

SSTR

[Syntax]

SSTR.size

B , W

[Instruction Code/Number of Cycles]

Page= 233

[Operation][*1]

When size specifier (.size) is (.B)

```
Repeat
        M(A1) ←  R0L
        A1    ←  A1  +  1
        R3    ←  R3  –  1
Until            R3  =  0
```

When size specifier (.size) is (.W)

```
Repeat
        M(A1) ←  R0
        A1    ←  A1  +  2
        R3    ←  R3  –  1
Until            R3  =  0
```

*1 If you set a value 0 in R3, this instruction is ingored.

[Function]

- This instruction stores string, with the store data indicated by R0, the transfer address indicated by A1, and the transfer count indicated by R3.
- The A0 or A1 when the instruction is completed contains the next address of the last-written data.
- If an interrupt request is received during instruction execution, the interrupt is acknowledged after one data transfer is completed.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

SSTR.B

[Related Instructions] SMOVB,SMOVF

STC

Transfer from control register

STore from Control register

STC

[Syntax]

STC src,dest

[Instruction Code/Number of Cycles]

Page= 234

[Operation]

dest ← src

[Function]

- This instruction transfers the control register indicated by *src* to *dest*. If *dest* is memory, specify the address in which to store the low-order address.
- If *dest* is memory while src is PC, the required memory capacity is 3 bytes. If *src* is not PC, the required memory capacity is 2 bytes.

[Selectable src/dest]

src				dest			
FB	SB	SP*1	ISP	~~R0L/~~R0	~~R0H/~~R1	~~R1L/~~R2	~~R1H/~~R3
FLG	INTBH	INTBL		~~A0/~~A0	~~A1/~~A1	[A0]	[A1]
				dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
				dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
				~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
				~~R2R0~~	~~R3R1~~	~~A1A0~~	
PC				~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
				~~A0/A0~~	~~A1/A1~~	[A0]	[A1]
				dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
				dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
				~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
				R2R0	R3R1	A1A0	

*1 Operation is performed on the stack pointer indicated by the U flag.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
STC        SB,R0
STC        FB,A0
```

[Related Instructions] POPC,PUSHC,LDC,LDINTB

STCTX

Save context

STore ConTeXt

STCTX

[Syntax]

STCTX abs16,abs20

[Instruction Code/Number of Cycles]

Page= 235

[Operation]

[Function]

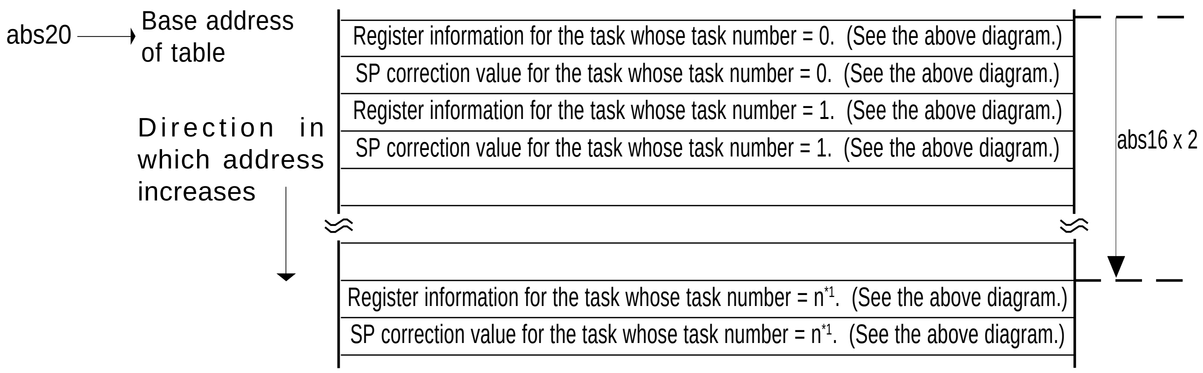
- This instruction saves task context to the stack area.
- Set the RAM address that contains the task number in abs16 and the start address of table data in abs20.
- The required register information is specified from table data by the task number and the data in the stack area is transferred to each register according to the specified register information. Then the SP correction value is subtracted to the stack pointer (SP). For this SP correction value, set the number of bytes you want to the transferred.
- Information on transferred registers is configured as shown below. Logic 1 indicates a register to be transferred and logic 0 indicates a register that is not transferred.

MSB LSB

FB	SB	A1	A0	R3	R2	R1	R0

Transferred sequentially beginning with FB

- The table data is comprised as shown below. The address indicated by abs20 is the base address of the table. The data stored at an address apart from the base address as much as twice the content of abs16 indicates register information, and the next address contains the stack pointer correction value.



abs20 → Base address of table

Direction in which address increases

Register information for the task whose task number = 0. (See the above diagram.)

SP correction value for the task whose task number = 0. (See the above diagram.)

Register information for the task whose task number = 1. (See the above diagram.)

SP correction value for the task whose task number = 1. (See the above diagram.)

Register information for the task whose task number = n*1. (See the above diagram.)

SP correction value for the task whose task number = n*1. (See the above diagram.)

*1 n=0 to 255

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
STCTX     Ram,Rom_TBL
```

[Related Instructions] LDCTX

STE

Transfer to extended data area

STore to EXtra far data area

STE

[Syntax]

STE.size **src,dest**

B , W

[Instruction Code/Number of Cycles]

Page= 235

[Operation]

dest ← src

[Function]

- This instruction transfers *src* to *dest* in an extended area.
- If *src* is an A0 or A1 when the size specifier (.size) you selected is (.B), operation is performed on the 8 low-order bits of A0 or A1. However, the flag changes depending on the A0 or A1 status (16 bits) before the operation is performed.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	~~R0L/R0~~	~~R0H/R1~~	~~R1L/R2~~	~~R1H/R3~~
A0/A0	A1/A1	[A0]	[A1]	~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~#IMM~~	dsp:20[A0]	~~dsp:20[A1]~~	abs20	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	[A1A0]	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
STE.B     R0L,[A1A0]
STE.W     R0,10000H[A0]
```

[Related Instructions] MOV,LDE,XCHG

STNZ

Conditional transfer

STore on Not Zero

STNZ

[Syntax]

STNZ src,dest

[Instruction Code/Number of Cycles]

Page= 237

[Operation]

if Z = 0 **then** dest ← src

[Function]

- This instruction transfers *src* to *dest* when the Z flag is 0.

[Selectable src/dest]

src	dest			
#IMM8	R0L	R0H	dsp:8[SB]	dsp:8[FB]
	abs16	~~A0~~	~~A1~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
STNZ        #5,Ram:8[SB]
```

[Related Instructions]

STZ,STZX

STZ

Conditional transfer

STore on Zero

STZ

[Syntax]

STZ src,dest

[Instruction Code/Number of Cycles]

Page= 237

[Operation]

if Z = 1 **then** dest ← src

[Function]

- This instruction transfers *src* to *dest* when the Z flag is 1.

[Selectable src/dest]

src	dest
#IMM8	R0L R0H dsp:8[SB] dsp:8[FB] abs16 ~~A0~~ ~~A1~~

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
STZ          #5,Ram:8[SB]
```

[Related Instructions] STNZ,STZX

STZX *Conditional transfer* STore on Zero eXtention STZX

[Syntax]

STZX src1,src2,dest

[Instruction Code/Number of Cycles]

Page= 238

[Operation]

```
If Z = 1 then
        dest ← src1
else
        dest ← src2
```

[Function]

- This instruction transfers *src1* to *dest* when the Z flag is 1. When the Z flag is 0, it transfers *src2* to *dest*.

[Selectable src/dest]

src	dest			
#IMM8	R0L	R0H	dsp:8[SB]	dsp:8[FB]
	abs16	~~A0~~	~~A1~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
STZX        #1,#2,Ram:8[SB]
```

[Related Instructions] STZ,STNZ

SUB *Subtract without borrow* **SUBtract** SUB

[Syntax]

SUB.size (:format) src,dest

- :format — **G , S** (Can be specified)
- .size — **B , W**

[Instruction Code/Number of Cycles]

Page= 238

[Operation]

dest ← dest – src

[Function]

- This instruction subtracts *src* from *dest* and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest] (See the next page for *src/dest* classified by format.)

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

[*1] If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	○	—	○	○	—	○

Conditions

O : The flag is set when a signed operation resulted in exceeding +32767 (.W) or –32768 (.W), or +127 (.B) or –128 (.B); otherwise cleared.

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

C : The flag is set when an unsigned operation resulted in any value equal to or greater than 0; otherwise cleared.

[Description Example]

```
SUB.B     A0,R0L                    ; A0's 8 low-order bits and R0L are operated on.
SUB.B     R0L,A0                    ; R0L is zero-expanded and operated with A0.
SUB.B     Ram:8[SB],R0L
SUB.W     #2,[A0]
```

[Related Instructions] ADC,ADCF,ADD,SBB

[src/dest Classified by Format]

G format

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~SP/SP~~
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

S format[*2]

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L	R0H	dsp:8[SB]	dsp:8[FB]
~~abs16~~	#IMM			abs16	~~A0~~	~~A1~~	
R0L[*3]	R0H[*3]	dsp:8[SB]	dsp:8[FB]	R0L[*3]	R0H[*3]	~~dsp:8[SB]~~	~~dsp:8[FB]~~
abs16	~~#IMM~~			~~abs16~~	~~A0~~	~~A1~~	

*2 You can only specify (.B) for the size specifier (.size).

*3 You cannot choose the same register for *src* and *dest*.

TST *Test* TeST TST

[Syntax]

TST.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 241

[Operation]

dest ∧ src

[Function]

- Each flag in the flag register changes state depending on the result of logical AND of *src* and *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0[*1]	A1/A1[*1]	[A0]	[A1]	A0/A0[*1]	A1/A1[*1]	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
TST.B    #3,R0L
TST.B    A0,R0L        ; A0's 8 low-order bits and ROL are operated on.
TST.B    R0L,A0        ; R0L is zero-expanded and operated on with A0.
```

[Related Instructions] AND,OR,XOR

UND

Interrupt for undefined instruction

UNDefined instruction

UND

[Syntax]

UND

[Instruction Code/Number of Cycles]

Page= 243

[Operation]

SP	←	SP – 2
M(SP)	←	$(PC + 1)_H$, FLG
SP	←	SP – 2
M(SP)	←	$(PC + 1)_{ML}$
PC	←	$M(FFFDC_{16})$

[Function]

- This instruction generates an undefined instruction interrupt.
- The undefined instruction interrupt is a nonmaskable interrupt.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	○	○	–	–	–	–	○	–

Conditions

U : The flag is cleared.

I : The flag is cleared.

D : The flag is cleared.

*1 The flags are saved to the stack area before the UND instruction is executed. After the interrupt, the flag status becomes as shown on the left.

[Description Example]

UND

WAIT *Wait* WAIT WAIT

[Syntax]

WAIT

[Instruction Code/Number of Cycles]

Page= 243

[Operation]

[Function]

- This instruction halts program execution. Program execution is restarted when an interrupt of a higher priority level than IPL is acknowledged or a reset is generated.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

WAIT

XCHG

Exchange
eXCHanGe

XCHG

[Syntax]

XCHG.size src,dest
B , W

[Instruction Code/Number of Cycles]

Page= 244

[Operation]

dest ←→ src

[Function]

- This instruction exchanges contents between *src* and *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), 16 bits of zero- expanded *src* data are placed in the A0 or A1 and the 8 low-order bits of the A0 or A1 are placed in *src*.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
~~A0/A0~~	~~A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0	A1/A1	[A0]	[A1]
~~dsp:8[A0]~~	~~dsp:8[A1]~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~abs16~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	~~#IMM~~	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~	~~[A1A0]~~	~~R2R0~~	~~R3R1~~	~~A1A0~~	

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	—	—	—	—

[Description Example]

```
XCHG.B    R0L,A0          ; A0’s 8 low-order bits and R0L’s zero-expanded value are exchanged.
XCHG.W    R0,A1
XCHG.B    R0L,[A0]
```

[Related Instructions] MOV,LDE,STE

XOR *Exclusive OR* eXclusive OR XOR

[Syntax]

XOR.size src,dest
└ B , W

[Instruction Code/Number of Cycles]

Page= 245

[Operation]

dest ← dest ∀ src

[Function]

- This instruction exclusive ORs *src* and *dest* together and stores the result in *dest*.
- If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

[Selectable src/dest]

src				dest			
R0L/R0	R0H/R1	R1L/R2	R1H/R3	R0L/R0	R0H/R1	R1L/R2	R1H/R3
A0/A0*1	A1/A1*1	[A0]	[A1]	A0/A0*1	A1/A1*1	[A0]	[A1]
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	abs16
~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	#IMM	~~dsp:20[A0]~~	~~dsp:20[A1]~~	~~abs20~~	
~~R2R0~~	~~R3R1~~	~~A1A0~~		~~R2R0~~	~~R3R1~~	~~A1A0~~	

*1 If you specify (.B) for the size specifier (.size), you cannot choose A0 or A1 for *src* and *dest* simultaneously.

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	—	—	—	—	○	○	—	—

Conditions

S : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z : The flag is set when the operation resulted in 0; otherwise cleared.

[Description Example]

```
XOR.B    A0,R0L             ; A0's 8 low-order bits and R0L are exclusive ORed.
XOR.B    R0L,A0             ; R0L is zero-expanded and exclusive ORed with A0.
XOR.B    #3,R0L
XOR.W    A0,A1
```

[Related Instructions] AND,OR,TST

Chapter 4

Instruction Code/Number of Cycles

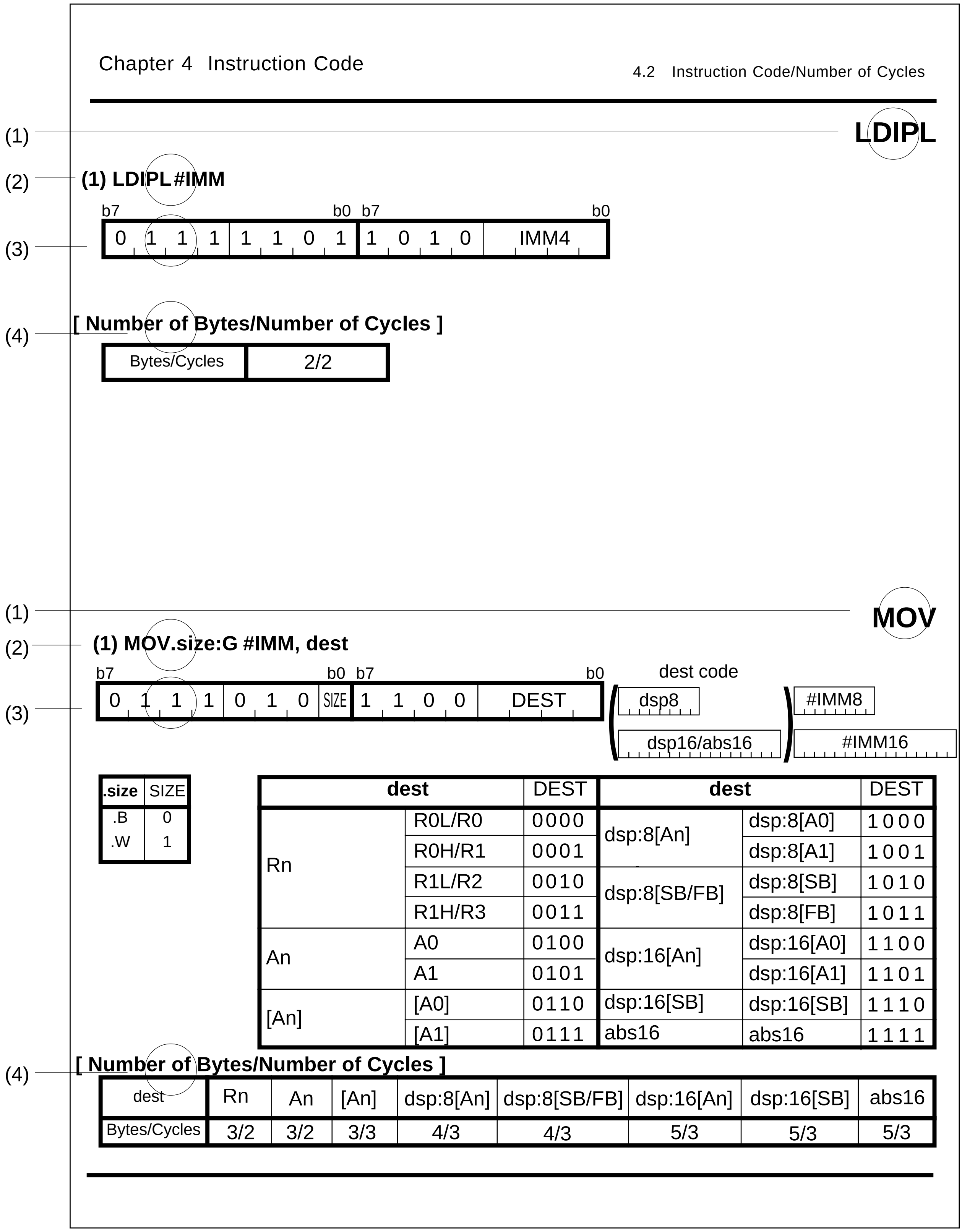
4.1 Guide to This Chapter

4.2 Instruction Code/Number of Cycles

4.1 Guide to This Chapter

This chapter describes instruction code and number of cycles for each op-code.

The following shows how to read this chapter by using an actual page as an example.



Chapter 4 Instruction Code

4.2 Instruction Code/Number of Cycles

(1) LDIPL

(2) **(1) LDIPL #IMM**

(3)

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	1	0	IMM4			

(4) **[Number of Bytes/Number of Cycles]**

Bytes/Cycles	2/2

(1) MOV

(2) **(1) MOV.size:G #IMM, dest**

(3)

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	1	1	0	0	DEST			

dest code

(dsp8 / dsp16/abs16) (#IMM8 / #IMM16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0000	dsp:8[An]	dsp:8[A0]	1000
	R0H/R1	0001		dsp:8[A1]	1001
	R1L/R2	0010	dsp:8[SB/FB]	dsp:8[SB]	1010
	R1H/R3	0011		dsp:8[FB]	1011
An	A0	0100	dsp:16[An]	dsp:16[A0]	1100
	A1	0101		dsp:16[A1]	1101
[An]	[A0]	0110	dsp:16[SB]	dsp:16[SB]	1110
	[A1]	0111	abs16	abs16	1111

(4) **[Number of Bytes/Number of Cycles]**

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/3	4/3	4/3	5/3	5/3	5/3

(1) Mnemonic

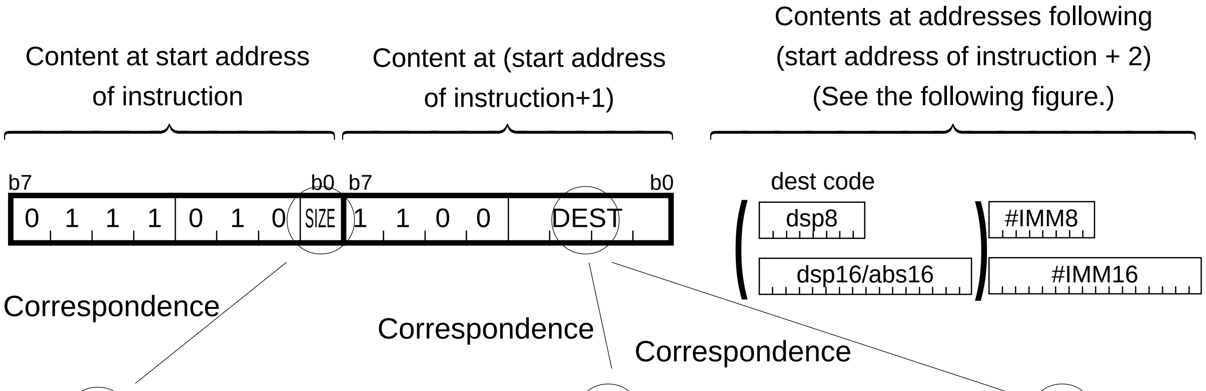
Shows the mnemonic explained in this page.

(2) Syntax

Shows an instruction syntax using symbols.

(3) Instruction code

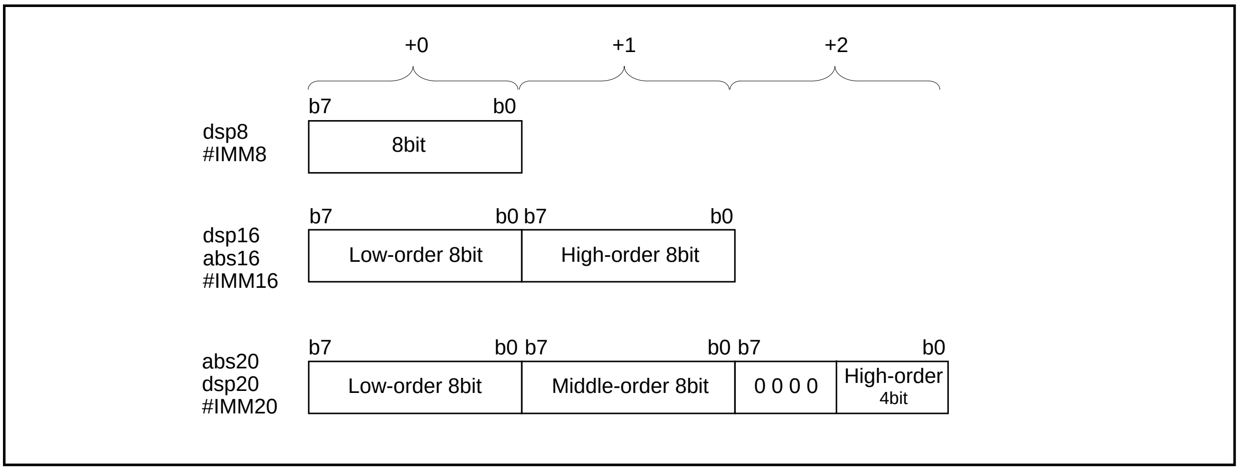
Shows instruction code. Entered in ( ) are omitted depending on src/dest you selected.


Content at start address of instruction
Content at (start address of instruction+1)
Contents at addresses following (start address of instruction + 2) (See the following figure.)
b7
b0
b7
b0
0 1 1 1 0 1 0 SIZE
1 1 0 0 DEST
dest code
dsp8
dsp16/abs16
#IMM8
#IMM16
Correspondence
Correspondence
Correspondence

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0000	dsp:8[An]	dsp:8[A0]	1000
	R0H/R1	0001		dsp:8[A1]	1001
	R1L/R2	0010	dsp:8[SB/FB]	dsp:8[SB]	1010
	R1H/R3	0011		dsp:8[FB]	1011
An	A0	0100	dsp:16[An]	dsp:16[A0]	1100
	A1	0101		dsp:16[A1]	1101
[An]	[A0]	0110	dsp:16[SB]	dsp:16[SB]	1110
	[A1]	0111	abs16	abs16	1111

Contents at addresses following (start address of instruction + 2) are arranged as follows:



(4) Table of cycles

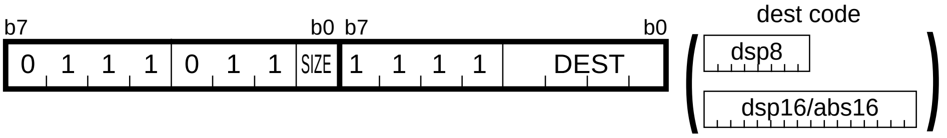
Shows the number of cycles required to execute this instruction and the number of instruction bytes.

There is a chance that the number of cycles increases due to an effect of software wait.

Instruction bytes are indicated on the left side of the slash and execution cycles are indicated on the right side.

ABS

(1) ABS.size dest



.size	SIZE
.B	0
.W	1

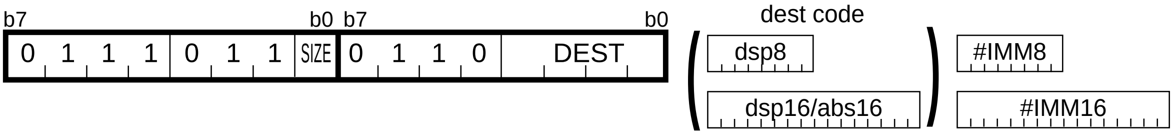
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/3	2/3	2/5	3/5	3/5	4/5	4/5	4/5

ADC

(1) ADC.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

ADC

(2) ADC.size src, dest

b7							b0	b7	b0
1	0	1	1	0	0	0	SIZE	SRC	DEST

src code: (dsp8 / dsp16/abs16) dest code: (dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

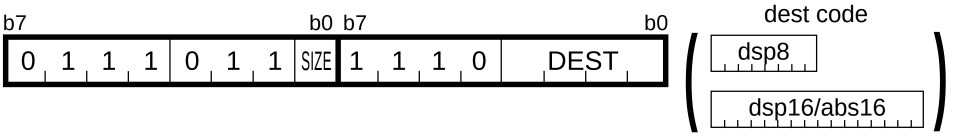
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

ADCF

(1) ADCF.size dest



.size	SIZE
.B	0
.W	1

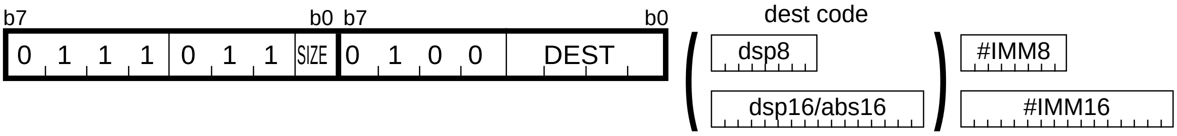
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

ADD

(1) ADD.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

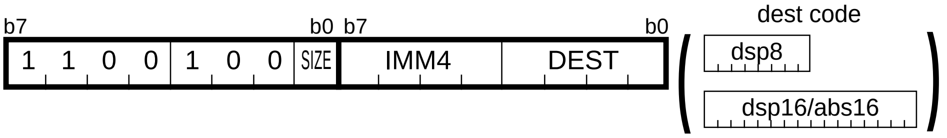
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

ADD

(2) ADD.size:Q #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

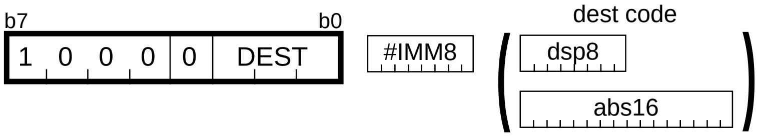
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

ADD

(3) ADD.B:S #IMM8, dest



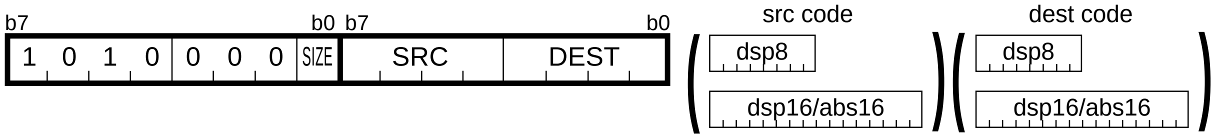
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

ADD

(4) ADD.size:G src, dest



.size	SIZE
.B	0
.W	1

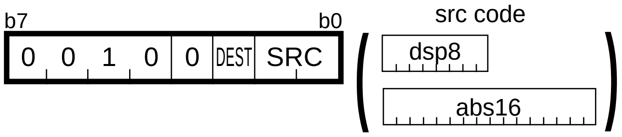
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

ADD

(5) ADD.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

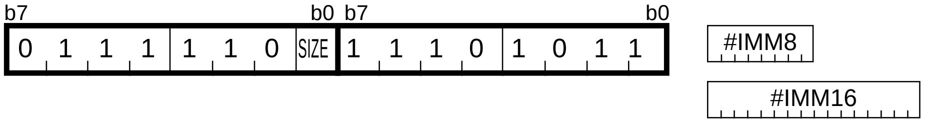
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

ADD

(6) ADD.size:G #IMM, SP



.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

(7) ADD.size:Q #IMM, SP

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	1	1	IMM4			

*1 The instruction code is the same regardless of whether you selected (.B) or (.W) for the size specifier (.size).

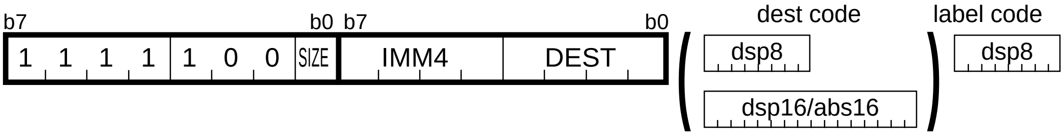
#IMM	IMM4	**#IMM**	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

ADJNZ

(1) ADJNZ.size #IMM, dest, label



dsp8 (label code)= address indicated by label –(start address of instruction + 2)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

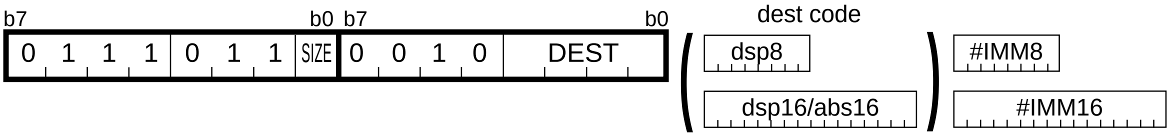
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/3	3/3	3/5	4/5	4/5	5/5	5/5	5/5

*1 If branched to label, the number of cycles above is increased by 4.

AND

(1) AND.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

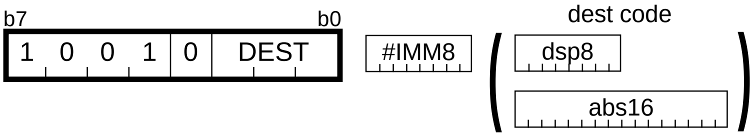
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

AND

(2) AND.B:S #IMM8, dest



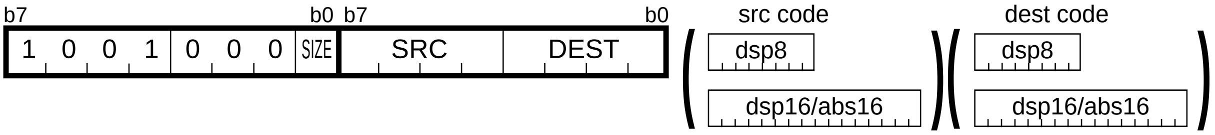
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

AND

(3) AND.size:G src, dest



.size	SIZE
.B	0
.W	1

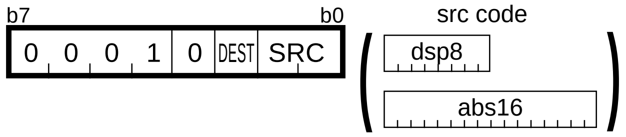
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

AND

(4) AND.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

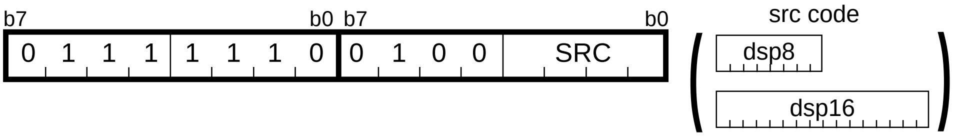
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

BAND

(1) BAND src



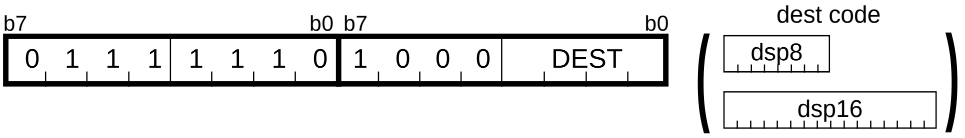
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BCLR

(1) BCLR:G dest



dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BCLR

(2) BCLR:S bit, base:11[SB]

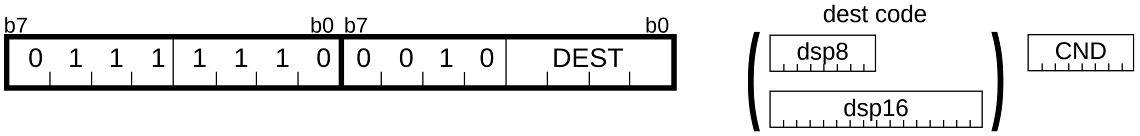
b7					b0	dest code
0	1	0	0	0	BIT	dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BM*Cnd*

(1) BM*Cnd* dest



dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

Cnd	CND	***Cnd***	CND
GEU/C	0 0 0 0 0 0 0 0	LTU/NC	1 1 1 1 1 0 0 0
GTU	0 0 0 0 0 0 0 1	LEU	1 1 1 1 1 0 0 1
EQ/Z	0 0 0 0 0 0 1 0	NE/NZ	1 1 1 1 1 0 1 0
N	0 0 0 0 0 0 1 1	PZ	1 1 1 1 1 0 1 1
LE	0 0 0 0 0 1 0 0	GT	1 1 1 1 1 1 0 0
O	0 0 0 0 0 1 0 1	NO	1 1 1 1 1 1 0 1
GE	0 0 0 0 0 1 1 0	LT	1 1 1 1 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	4/6	4/6	3/10	4/10	4/7	5/10	5/7	5/7

BM*Cnd*

(2) BM*Cnd* C

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	0	1	CND			

Cnd	CND	***Cnd***	CND
GEU/C	0 0 0 0	PZ	0 1 1 1
GTU	0 0 0 1	LE	1 0 0 0
EQ/Z	0 0 1 0	O	1 0 0 1
N	0 0 1 1	GE	1 0 1 0
LTU/NC	0 1 0 0	GT	1 1 0 0
LEU	0 1 0 1	NO	1 1 0 1
NE/NZ	0 1 1 0	LT	1 1 1 0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 If the condition is true, the number of cycles above is increased by 1.

BNAND

(1) BNAND src

b7							b0	b7							b0
0	1	1	1	1	1	1	0	0	1	0	1	SRC			

src code

(dsp8 <br> dsp16)

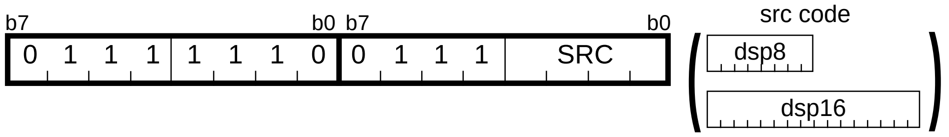
src		SRC	**src**		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNOR

(1) BNOR src



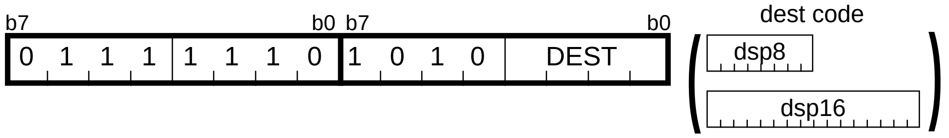
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNOT

(1) BNOT:G dest



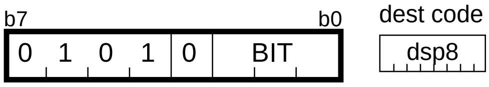
dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BNOT

(2) BNOT:S bit, base:11[SB]

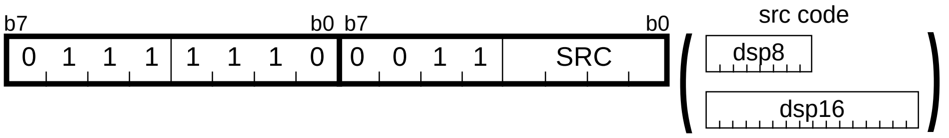


[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BNTST

(1) BNTST src



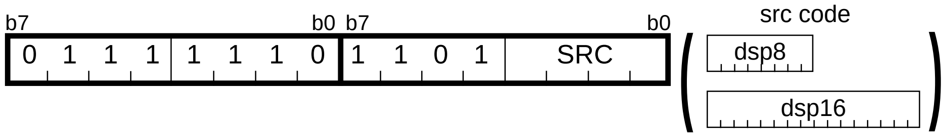
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BNXOR

(1) BNXOR src



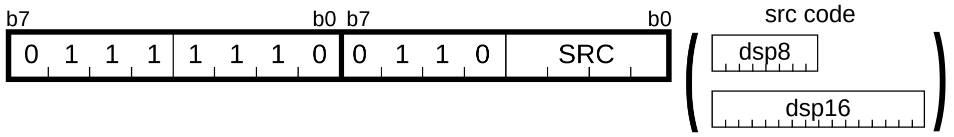
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BOR

(1) BOR src



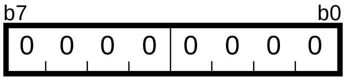
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BRK

(1) BRK



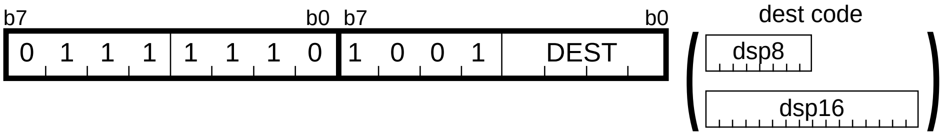
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/27

*1 If you specify the target address of the BRK interrupt by use of the interrupt table register (INTB), the number of cycles shown in the table increases by two. At this time, set FF_{16} in addresses $FFFE4_{16}$ through $FFFE7_{16}$.

BSET

(1) BSET:G dest



dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BSET

(2) BSET:S bit, base:11[SB]

b7					b0	dest code
0	1	0	0	1	BIT	dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BTST

(1) BTST:G src

b7							b0	b7				b0	src code
0	1	1	1	1	1	1	0	1	0	1	1	SRC	(dsp8 dsp16)

src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/2	3/2	2/6	3/6	3/3	4/6	4/3	4/3

BTST

(2) BTST:S bit, base:11[SB]

b7						b0	src code
0	1	0	1	1	BIT		dsp8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

BTSTC

(1) BTSTC dest

b7							b0	b7				b0	dest code
0	1	1	1	1	1	1	0	0	0	0	0	DEST	(dsp8 dsp16)

dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BTSTS

(1) BTSTS dest

b7							b0	b7				b0	dest code
0	1	1	1	1	1	1	0	0	0	0	1	DEST	(dsp8 dsp16)

dest		DEST	dest		DEST
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

BXOR

(1) BXOR src

b7							b0	b7				b0	src code
0	1	1	1	1	1	1	0	1	1	0	0	SRC	(dsp8 dsp16)

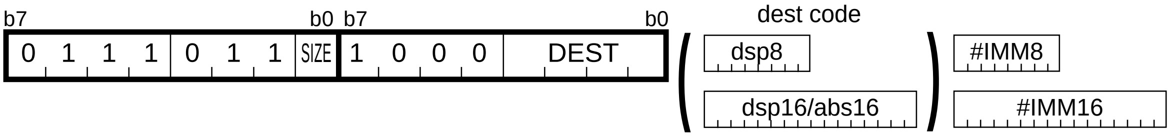
src		SRC	src		SRC
bit,Rn	bit,R0	0 0 0 0	base:8[An]	base:8[A0]	1 0 0 0
	bit,R1	0 0 0 1		base:8[A1]	1 0 0 1
	bit,R2	0 0 1 0	bit,base:8 [SB/FB]	bit,base:8[SB]	1 0 1 0
	bit,R3	0 0 1 1		bit,base:8[FB]	1 0 1 1
bit,An	bit,A0	0 1 0 0	base:16[An]	base:16[A0]	1 1 0 0
	bit,A1	0 1 0 1		base:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	bit,base:16[SB]	bit,base:16[SB]	1 1 1 0
	[A1]	0 1 1 1	bit,base:16	bit,base:16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	[An]	base:8 [An]	bit,base:8 [SB/FB]	base:16 [An]	bit,base:16 [SB]	bit,base:16
Bytes/Cycles	3/3	3/3	2/7	3/7	3/4	4/7	4/4	4/4

CMP

(1) CMP.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

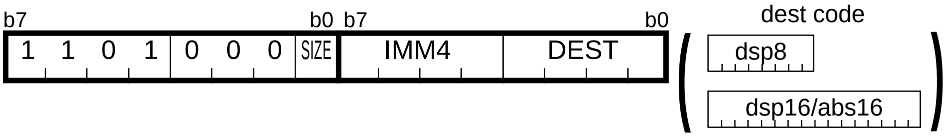
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

CMP

(2) CMP.size:Q #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

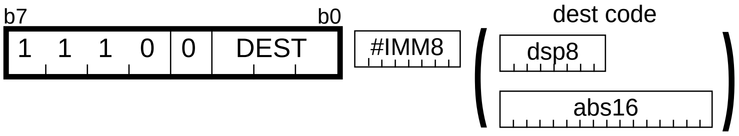
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

CMP

(3) CMP.B:S #IMM8, dest



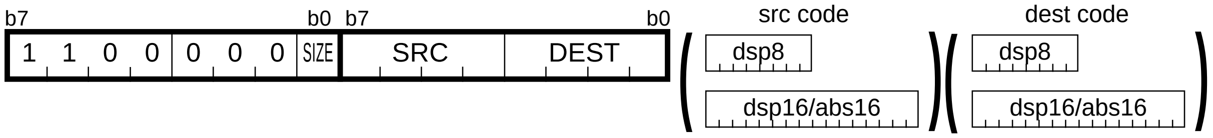
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

CMP

(4) CMP.size:G src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

CMP

(5) CMP.B:S src, R0L/R0H

b7						b0
0	0	1	1	1	DEST	SRC

src code

(dsp8

abs16)

src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

DADC

(1) DADC.B #IMM8, R0L

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	0	1	1	1	0	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/5

DADC

(2) DADC.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	1	0	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/5

DADC

(3) DADC.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADC

(4) DADC.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADD

(1) DADD.B #IMM8, R0L

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	0	1	1	0	0	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/5

DADD

(2) DADD.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	0	0	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/5

DADD

(3) DADD.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DADD

(4) DADD.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	0	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5

DEC

(1) DEC.B dest

b7							b0
1	0	1	0	1	DEST		

dest code

(dsp8
abs16)

dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

DEC

(2) DEC.W dest

b7							b0
1	1	1	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

DIV

(1) DIV.size #IMM

b7							b0	b7							b0	
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	0	1	#IMM8 #IMM16

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

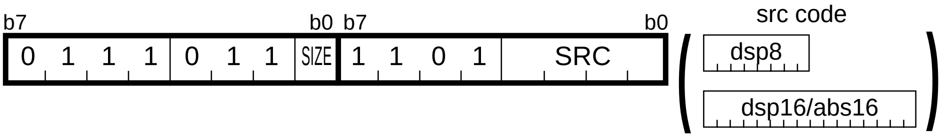
Bytes/Cycles	3/22

*1 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 6, respectively.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIV

(2) DIV.size src



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

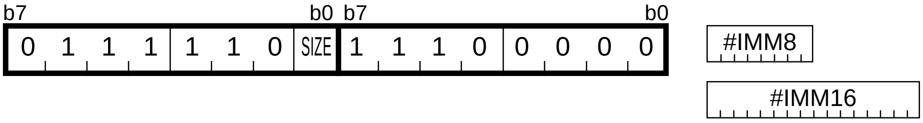
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/22	2/22	2/24	3/24	3/24	4/24	4/24	4/24

*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 6.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIVU

(1) DIVU.size #IMM



.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/18

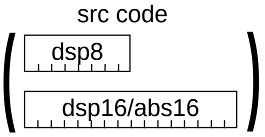
*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

*3 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 7, respectively.

DIVU

(2) DIVU.size src

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	1	0	0	SRC			



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/18	2/18	2/20	3/20	3/20	4/20	4/20	4/20

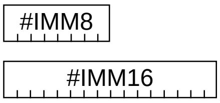
*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 7.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DIVX

(1) DIVX.size #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	0	0	1	1



.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

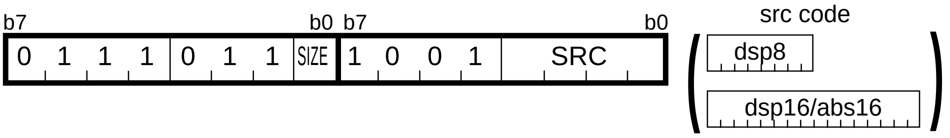
Bytes/Cycles	3/22

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

*3 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 6, respectively.

DIVX

(2) DIVX.size src



.size	SIZE
.B	0
.W	1

src		SRC	**src**		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

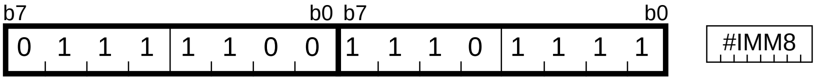
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/22	2/22	2/24	3/24	3/24	4/24	4/24	4/24

*1 If the size specifier (.size) is (.W), the number of cycles above is increased by 6.

*2 The number of cycles may decrease when an overflow occurs or depending on the value of the divisor or dividend.

DSBB

(1) DSBB.B #IMM8, R0L



[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

DSBB

(2) DSBB.W #IMM16, R0

b7							b0	b7							b0		#IMM16
0	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1		

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

DSBB

(3) DSBB.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSBB

(4) DSBB.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSUB

(1) DSUB.B #IMM8, R0L

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	0	1	1	0	1	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

DSUB

(2) DSUB.W #IMM16, R0

b7							b0	b7							b0	
0	1	1	1	1	1	0	1	1	1	1	0	1	1	0	1	#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

DSUB

(3) DSUB.B R0H, R0L

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

DSUB

(4) DSUB.W R1, R0

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	0	0	1	0	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

ENTER

(1) ENTER #IMM8

b7							b0	b7							b0	
0	1	1	1	1	1	0	0	1	1	1	1	0	0	1	0	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

EXITD

(1) EXITD

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	1	0	0	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/9

EXTS

(1) EXTS.B dest

b7							b0	b7				b0
0	1	1	1	1	1	0	0	0	1	1	0	DEST

dest code

(dsp8
dsp16/abs16)

dest		DEST	dest		DEST
Rn	R0L	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
---	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/3	2/5	3/5	3/5	4/5	4/5	4/5

EXTS

(2) EXTS.W R0

b7							b0	b7							b0
0	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

FCLR

(1) FCLR dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	DEST			0	1	0	1

dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

FSET

(1) FSET dest

b7							b0	b7					b0
1	1	1	0	1	0	1	1	0	DEST	0	1	0	0

dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

INC

(1) INC.B dest

b7					b0
1	0	1	0	0	DEST

dest code

(dsp8
abs16)

dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

INC

(2) INC.W dest

b7							b0
1	0	1	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

INT

(1) INT #IMM

b7							b0
1	1	1	0	1	0	1	1

1 1	#IMM

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/19

INTO

(1) INTO

b7							b0
1	1	1	1	0	1	1	0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

*1 If the O flag = 1, the number of cycles above is increased by 19.

J*Cnd*

(1) J*Cnd* label

b7							b0	label code
0	1	1	0	1	CND			dsp8

dsp8 = address indicated by label – (start address of instruction + 1)

Cnd	CND	***Cnd***	CND
GEU/C	0 0 0	LTU/NC	1 0 0
GTU	0 0 1	LEU	1 0 1
EQ/Z	0 1 0	NE/NZ	1 1 0
N	0 1 1	PZ	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

*2 If branched to label, the number of cycles above is increased by 2.

J*Cnd*

(2) J*Cnd* label

b7							b0	b7							b0	label code
0	1	1	1	1	1	0	1	1	1	0	0	CND				dsp8

dsp8 =address indicated by label – (start address of instruction + 2)

Cnd	CND	***Cnd***	CND
LE	1 0 0 0	GT	1 1 0 0
O	1 0 0 1	NO	1 1 0 1
GE	1 0 1 0	LT	1 1 1 0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If branched to label, the number of cycles above is increased by 2.

JMP

(1) JMP.S label

b7							b0
0	1	1	0	0	dsp		

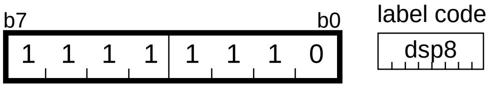
dsp = address indicated by label – (start address of instruction + 2)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/5

JMP

(2) JMP.B label



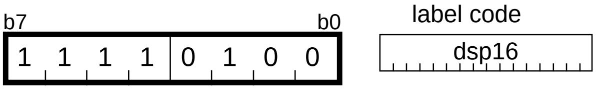
dsp8 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

JMP

(3) JMP.W label



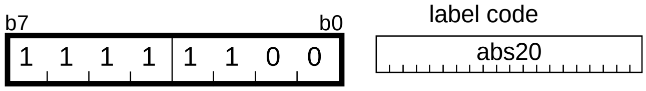
dsp16 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/4

JMP

(4) JMP.A label

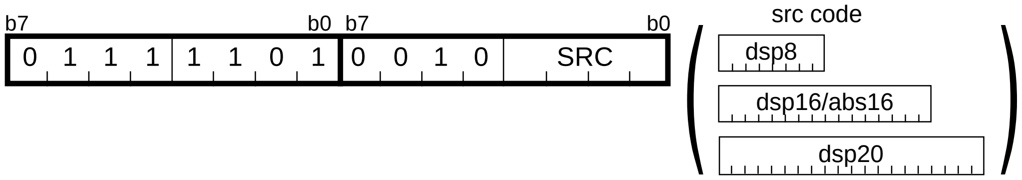


[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/4

JMPI

(1) JMPI.W src



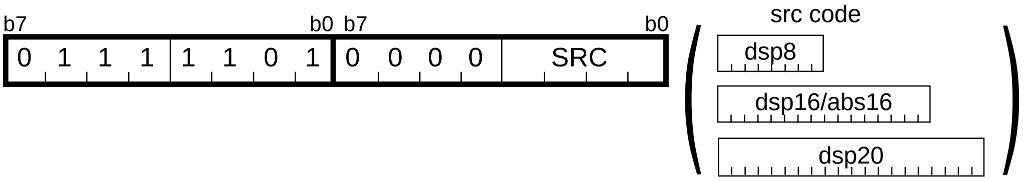
src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	A1	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/7	2/7	2/11	3/11	3/11	5/11	4/11	4/11

JMPI

(2) JMPI.A src



src		SRC	src		SRC
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	---	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

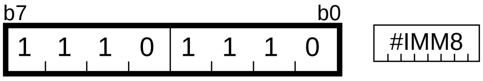
*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/6	2/6	2/10	3/10	3/10	5/10	4/10	4/10

JMPS

(1) JMPS #IMM8



[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/9

JSR

(1) JSR.W label

b7							b0	label code
1	1	1	1	0	1	0	1	dsp16

dsp16 = address indicated by label – (start address of instruction + 1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/8

JSR

(2) JSR.A label

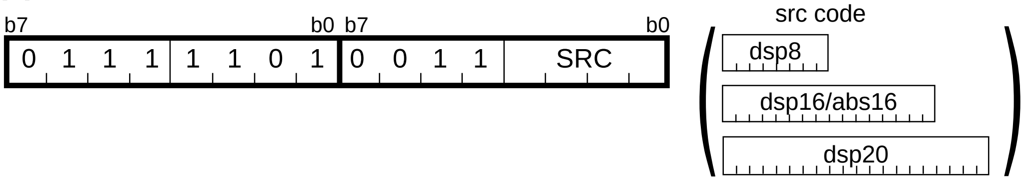
b7							b0	label code
1	1	1	1	1	1	0	1	abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/9

JSRI

(1) JSRI.W src



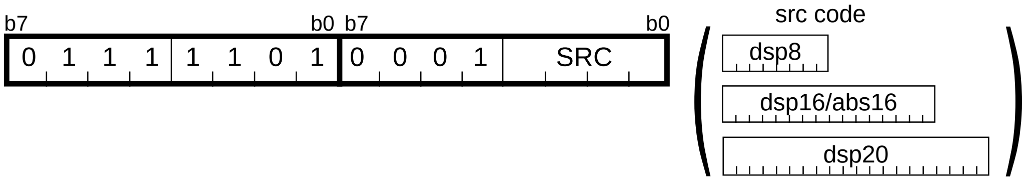
src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	A1	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/11	2/11	2/15	3/15	3/15	5/15	4/15	4/15

JSRI

(2) JSRI.A src



src		SRC	src		SRC
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:20[An]	dsp:20[A0]	1 1 0 0
	---	0 1 0 1		dsp:20[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:20[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/11	2/11	2/15	3/15	3/15	5/15	4/15	4/15

JSRS

(1) JSRS #IMM8

b7							b0	
1	1	1	0	1	1	1	1	#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/13

LDC

(1) LDC #IMM16, dest

b7							b0	b7					b0	
1	1	1	0	1	0	1	1	0	DEST	0	0	0	0	#IMM16

dest	DEST
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

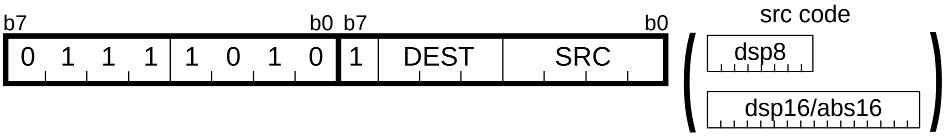
*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/2

LDC

(2) LDC src, dest



src		SRC	src		SRC
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest	DEST
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

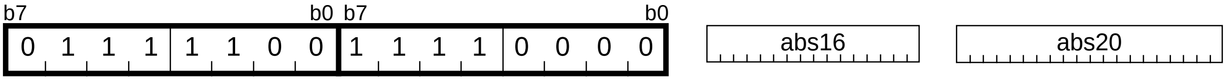
*1 Marked by **---** cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

LDCTX

(1) LDCTX abs16, abs20



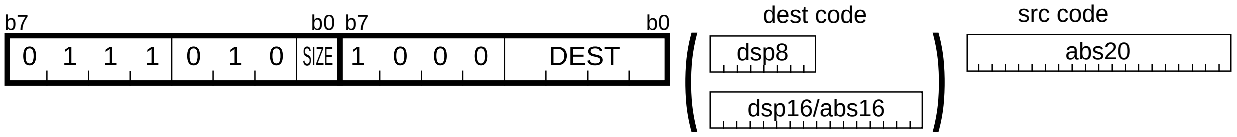
[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/11+2×m

*2 m denotes the number of transfers performed.

LDE

(1) LDE.size abs20, dest



.size	SIZE
.B	0
.W	1

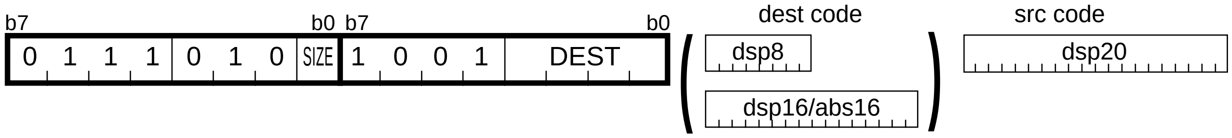
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	5/4	5/4	5/5	6/5	6/5	7/5	7/5	7/5

LDE

(2) LDE.size dsp:20[A0], dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	5/4	5/4	5/5	6/5	6/5	7/5	7/5	7/5

LDE

(3) LDE.size [A1A0], dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	1	0	1	0	DEST			

dest code

(dsp8 / dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5

LDINTB

(1) LDINTB #IMM

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	0
0	0	0	0	#IMM1				0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	1	0	0	0	1	0	0	0	0
#IMM2															

*1 #IMM1 indicates the 4 high-order bits of #IMM.
#IMM2 indicates the 16 low-order bits of #IMM.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	8/4

LDIPL

(1) LDIPL #IMM

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	1	0	0	#IMM		

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

MOV

(1) MOV.size:G #IMM, dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	1	1	0	0	DEST			

dest code
(dsp8 / dsp16/abs16) #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

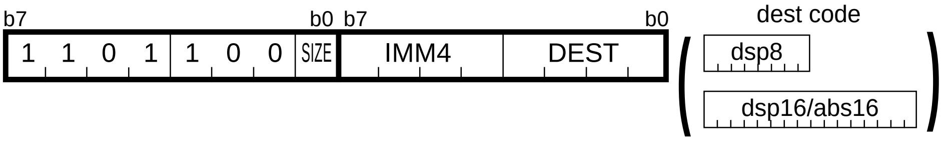
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/3	4/3	4/3	5/3	5/3	5/3

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

MOV

(2) MOV.size:Q #IMM, dest



.size	SIZE
.B	0
.W	1

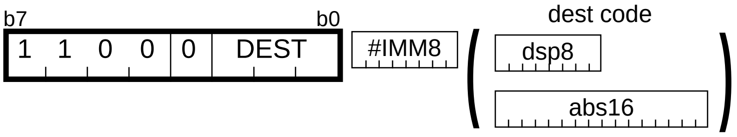
#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	–8	1 0 0 0
+1	0 0 0 1	–7	1 0 0 1
+2	0 0 1 0	–6	1 0 1 0
+3	0 0 1 1	–5	1 0 1 1
+4	0 1 0 0	–4	1 1 0 0
+5	0 1 0 1	–3	1 1 0 1
+6	0 1 1 0	–2	1 1 1 0
+7	0 1 1 1	–1	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/2	3/2	3/2	4/2	4/2	4/2

(3) MOV.B:S #IMM8, dest



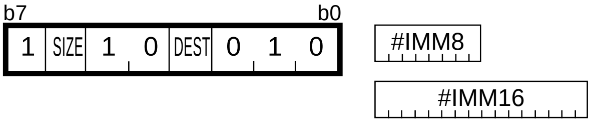
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

MOV

(4) MOV.size:S #IMM, dest



.size	SIZE
.B	1
.W	0

dest	DEST
A0	0
A1	1

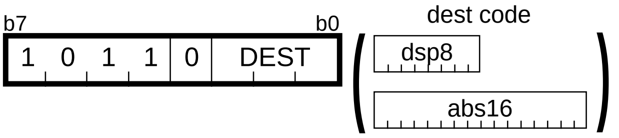
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 If the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 1, respectively.

MOV

(5) MOV.B:Z #0, dest



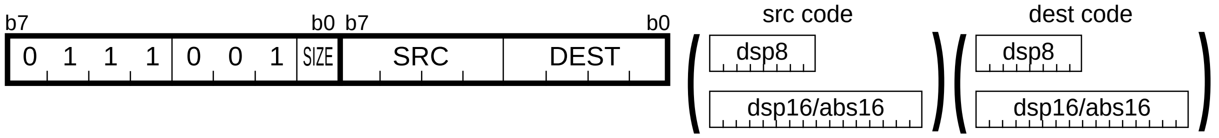
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/2	3/2

MOV

(6) MOV.size:G src, dest



.size	SIZE
.B	0
.W	1

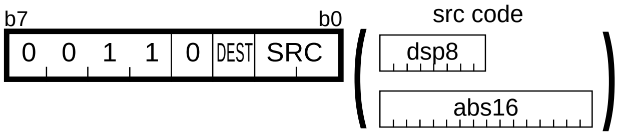
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/2	3/2	3/2	4/2	4/2	4/2
An	2/2	2/2	2/2	3/2	3/2	4/2	4/2	4/2
[An]	2/3	2/3	2/3	3/3	3/3	4/3	4/3	4/3
dsp:8[An]	3/3	3/3	3/3	4/3	4/3	5/3	5/3	5/3
dsp:8[SB/FB]	3/3	3/3	3/3	4/3	4/3	5/3	5/3	5/3
dsp:16[An]	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3
dsp:16[SB]	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3
abs16	4/3	4/3	4/3	5/3	5/3	6/3	6/3	6/3

MOV

(7) MOV.B:S src, dest



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

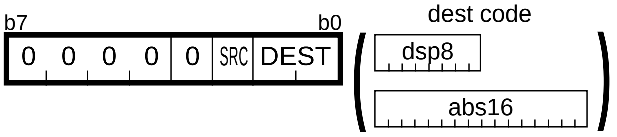
dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

MOV

(8) MOV.B:S R0L/R0H, dest



src	SRC
R0L	0
R0H	1

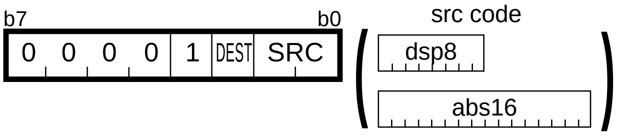
dest		DEST
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

[Number of Bytes/Number of Cycles]

dest	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/2	3/2

MOV

(9) MOV.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

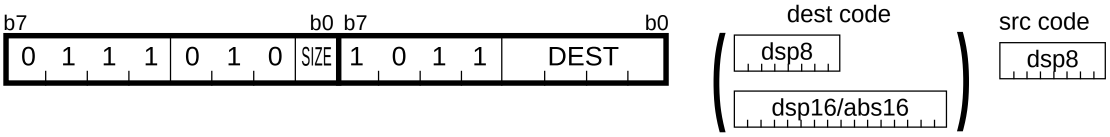
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

MOV

(10) MOV.size:G dsp:8[SP], dest



.size	SIZE
.B	0
.W	1

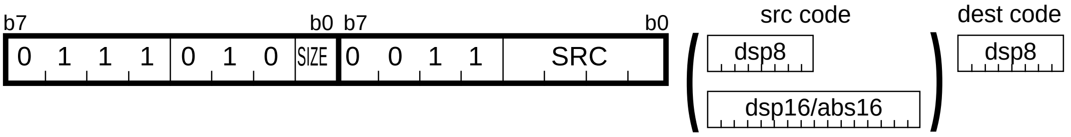
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/3	4/3	4/3	5/3	5/3	5/3

MOV

(11) MOV.size:G src, dsp:8[SP]



.size	SIZE
.B	0
.W	1

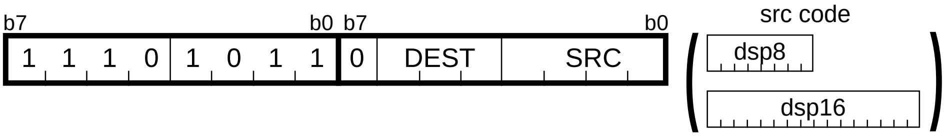
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4

MOVA

(1) MOVA src, dest



src		SRC
dsp:8[An]	dsp:8[A0]	1 0 0 0
	dsp:8[A1]	1 0 0 1
dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	dsp:8[FB]	1 0 1 1
dsp:16[An]	dsp:16[A0]	1 1 0 0
	dsp:16[A1]	1 1 0 1
dsp:16[SB]	dsp:16[SB]	1 1 1 0
abs16	abs16	1 1 1 1

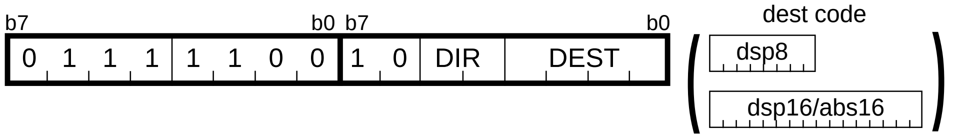
dest	DEST
R0	0 0 0
R1	0 0 1
R2	0 1 0
R3	0 1 1
A0	1 0 0
A1	1 0 1

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	4/2	4/2	4/2

MOV*Dir*

(1) MOV*Dir* R0L, dest



Dir	DIR
LL	0 0
LH	1 0
HL	0 1
HH	1 1

dest		DEST	**dest**		DEST
Rn	---	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H	0 0 1 1		dsp:8[FB]	1 0 1 1
An	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

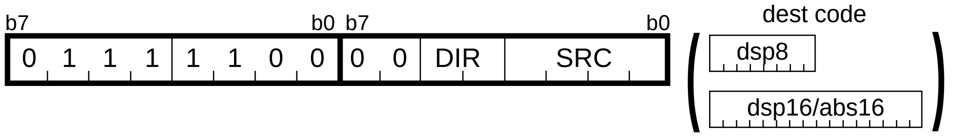
*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
MOVHH, MOVLL	2/4	2/5	3/5	3/5	4/5	4/5	4/5
MOVHL, MOVLH	2/7	2/8	3/8	3/8	4/8	4/8	4/8

MOV*Dir*

(2) MOV*Dir* src, R0L



Dir	DIR
LL	0 0
LH	1 0
HL	0 1
HH	1 1

src		SRC	**src**		SRC
Rn	R0L	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H	0 0 1 1		dsp:8[FB]	1 0 1 1
An	---	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

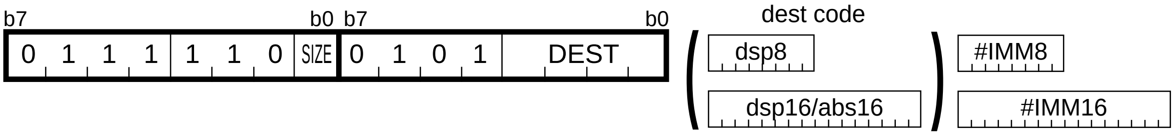
*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

src	Rn	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
MOVHH, MOVLL	2/3	2/5	3/5	3/5	4/5	4/5	4/5
MOVHL, MOVLH	2/6	2/8	3/8	3/8	4/8	4/8	4/8

MUL

(1) MUL.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

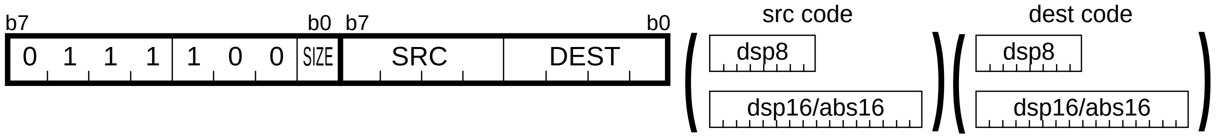
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/4	3/4	3/5	4/5	4/5	5/5	5/5	5/5

*2 If dest is Rn or An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 each.

*3 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 2, respectively.

MUL

(2) MUL.size src, dest



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5
An	2/4	2/5	2/5	3/5	3/5	4/5	4/5	4/5
[An]	2/6	2/6	2/6	3/6	3/6	4/6	4/6	4/6
dsp:8[An]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:8[SB/FB]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:16[An]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
dsp:16[SB]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
abs16	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6

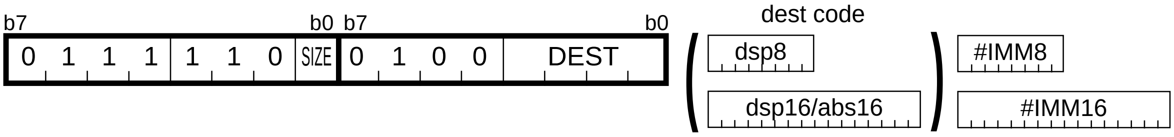
*2 If src is An and dest is Rn while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*3 If src is not An and dest is Rn or An while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*4 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of cycles above is increased by 2.

MULU

(1) MULU.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Numbera of Bytes/Number of Cycles]

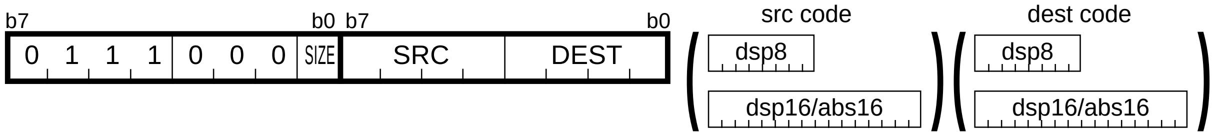
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/4	3/4	3/5	4/5	4/5	5/5	5/5	5/5

*2 If dest is Rn or An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 each.

*3 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of bytes and cycles above are increased by 1 and 2, respectively.

MULU

(2) MULU.size src, dest



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	--- /R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5
An	2/4	2/5	2/5	3/5	3/5	4/5	4/5	4/5
[An]	2/6	2/6	2/6	3/6	3/6	4/6	4/6	4/6
dsp:8[An]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:8[SB/FB]	3/6	3/6	3/6	4/6	4/6	5/6	5/6	5/6
dsp:16[An]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
dsp:16[SB]	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6
abs16	4/6	4/6	4/6	5/6	5/6	6/6	6/6	6/6

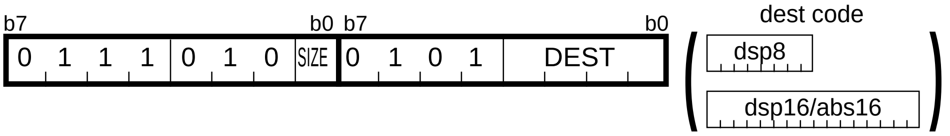
*2 If src is An and dest is Rn while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*3 If src is not An and dest is Rn or An while the size specifier (.size) is (.W), the number of cycles above is increased by 1.

*4 If dest is neither Rn nor An while the size specifier (.size) is (.W), the number of cycles above is increased by 2.

NEG

(1) NEG.size dest



.size	SIZE
.B	0
.W	1

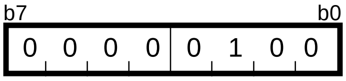
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

NOP

(1) NOP

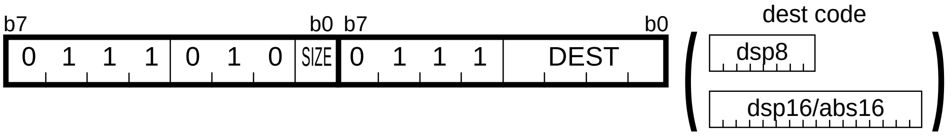


[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

NOT

(1) NOT.size:G dest



.size	SIZE
.B	0
.W	1

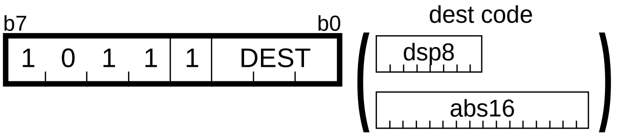
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

NOT

(2) NOT.B:S dest



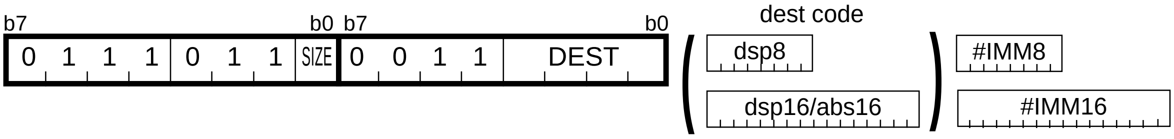
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/3	3/3

OR

(1) OR.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

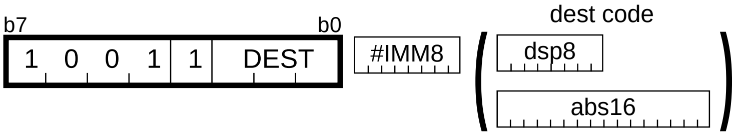
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

OR

(2) OR.B:S #IMM8, dest



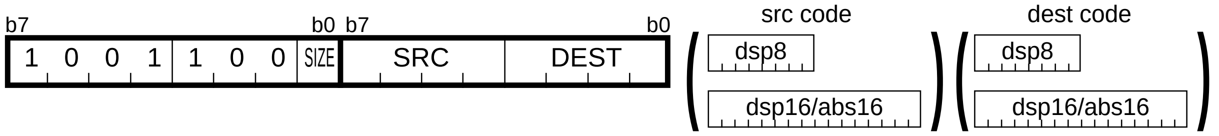
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

OR

(3) OR.size:G src, dest



.size	SIZE
.B	0
.W	1

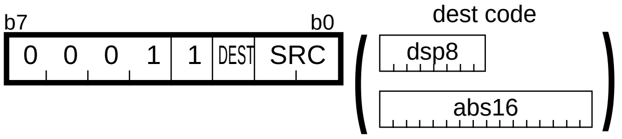
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

OR

(4) OR.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

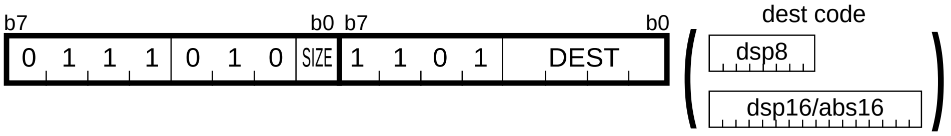
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

POP

(1) POP.size:G dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4

POP

(2) POP.B:S dest

b7							b0
1	0	0	1	DEST	0	1	0

dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

POP

(3) POP.W:S dest

b7							b0
1	1	0	1	DEST	0	1	0

dest	DEST
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

POPC

(1) POPC dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	DEST			0	0	1	1

dest	DEST	**dest**	DEST
---	0 0 0	ISP	1 0 0
INTBL	0 0 1	SP	1 0 1
INTBH	0 1 0	SB	1 1 0
FLG	0 1 1	FB	1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

POPM

(1) POPM dest

b7							b0	
1	1	1	0	1	1	0	1	DEST

dest							
FB	SB	A1	A0	R3	R2	R1	R0
DEST*2							

*2 The bit for a selected register is 1.
The bit for a non-selected register is 0.

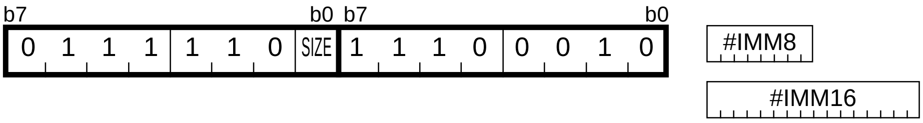
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

*3 If two or more registers need to be restored, the number of required cycles is 2 x m (m: number of registers to be restored).

PUSH

(1) PUSH.size:G #IMM



.size	SIZE
.B	0
.W	1

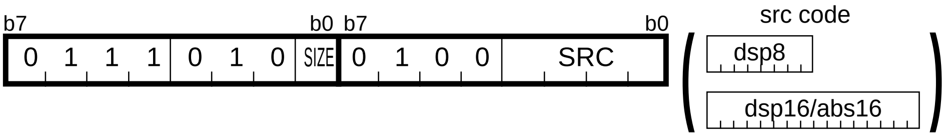
[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

PUSH

(2) PUSH.size:G src



.size	SIZE
.B	0
.W	1

src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	4/4

PUSH

(3) PUSH.B:S src

b7							b0
1	0	0	0	SRC	0	1	0

src	SRC
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/2

PUSH

(4) PUSH.W:S src

b7							b0
1	1	0	0	SRC	0	1	0

src	SRC
A0	0
A1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/2

PUSHA

(1) PUSHA src

b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	0	0	1	SRC			

src code

(dsp8
dsp16/abs16)

src		SRC
dsp:8[An]	dsp:8[A0]	1 0 0 0
	dsp:8[A1]	1 0 0 1
dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	dsp:8[FB]	1 0 1 1
dsp:16[An]	dsp:16[A0]	1 1 0 0
	dsp:16[A1]	1 1 0 1
dsp:16[SB]	dsp:16[SB]	1 1 1 0
abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs:16
Bytes/Cycles	3/2	3/2	4/2	4/2	4/2

PUSHC

(1) PUSHC src

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	SRC			0	0	1	0

src	SRC	src	SRC
---	0 0 0	ISP	1 0 0
INTBL	0 0 1	SP	1 0 1
INTBH	0 1 0	SB	1 1 0
FLG	0 1 1	FB	1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

PUSHM

(1) PUSHM src

b7							b0	
1	1	1	0	1	1	0	0	SRC

src							
R0	R1	R2	R3	A0	A1	SB	FB
SRC*1							

*1 The bit for a selected register is 1.
The bit for a non-selected register is 0.

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2×m

*2 m denotes the number of registers to be saved.

REIT

(1) REIT

b7							b0
1	1	1	1	1	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

RMPA

(1) RMPA.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	1	0	0	0	1

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+7×m

*1 m denotes the number of operation performed.

*2 If the size specifier (.size) is (.W), the number of cycles is (6+9×m).

ROLC

(1) ROLC.size dest

b7							b0	b7							b0
0	1	1	1	0	1	1	SIZE	1	0	1	0	DEST			

dest code

(dsp8

dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

RORC

(1) RORC.size dest

b7							b0	b7							b0	dest code
0	1	1	1	0	1	1	SIZE	1	0	1	1	DEST				(dsp8 dsp16/abs16)

.size	SIZE
.B	0
.W	1

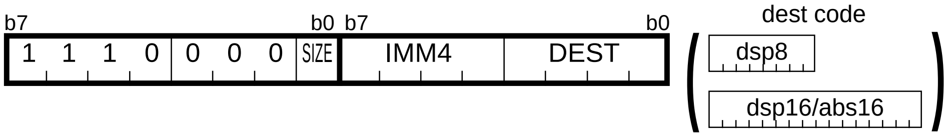
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	4/3

ROT

(1) ROT.size #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of rotates performed.

ROT

(2) ROT.size R1H, dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	0	1	1	0	DEST			

dest code

(dsp8
dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of rotates performed.

RTS

(1) RTS

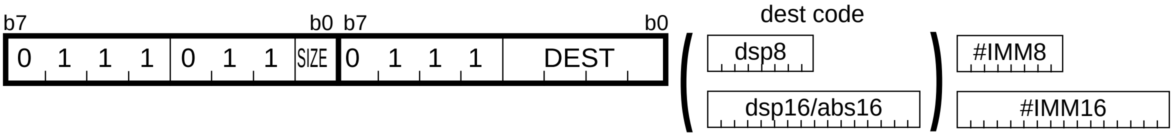
b7							b0
1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

SBB

(1) SBB.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

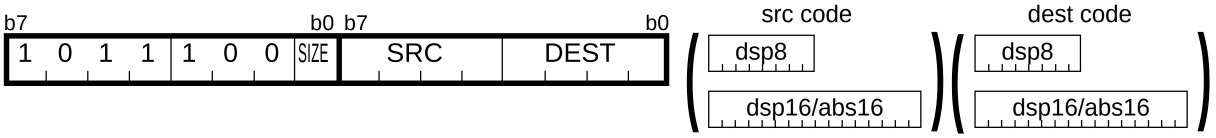
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

SBB

(2) SBB.size src, dest



.size	SIZE
.B	0
.W	1

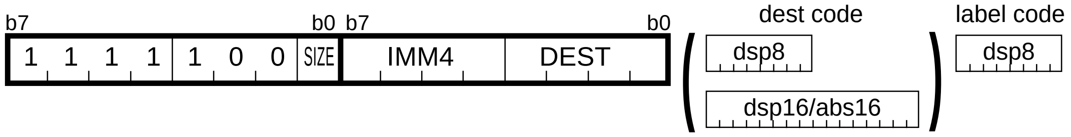
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

SBJNZ

(1) SBJNZ.size #IMM, dest, label



dsp8(label code) = address indicated by label – (start address of instruction + 2)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
0	0 0 0 0	+8	1 0 0 0
–1	0 0 0 1	+7	1 0 0 1
–2	0 0 1 0	+6	1 0 1 0
–3	0 0 1 1	+5	1 0 1 1
–4	0 1 0 0	+4	1 1 0 0
–5	0 1 0 1	+3	1 1 0 1
–6	0 1 1 0	+2	1 1 1 0
–7	0 1 1 1	+1	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/3	3/3	3/5	4/5	4/5	5/5	5/5	5/5

*1 If branched to label, the number of cycles above is increased by 4.

SHA

(1) SHA.size #IMM, dest

b7							b0	b7	b0	dest code
1	1	1	1	0	0	0	SIZE	IMM4	DEST	(dsp8 dsp16/abs16)

.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of shifts performed.

SHA

(2) SHA.size R1H, dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	1	1	1	1	DEST			

dest code

(dsp8)
(dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of shifts performed.

SHA

(3) SHA.L #IMM, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	1	0	1	DEST	IMM4			

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+m

*2 m denotes the number of shifts performed.

SHA

(4) SHA.L R1H, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	1	DEST	0	0	0	1

dest	DEST
R2R0	0
R3R1	1

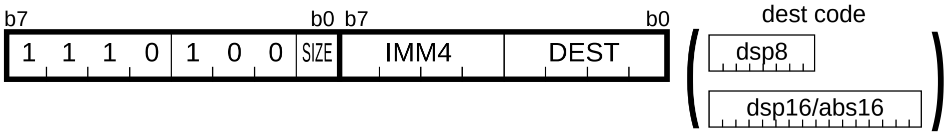
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+m

*1 m denotes the number of shifts performed.

SHL

(1) SHL.size #IMM, dest



.size	SIZE
.B	0
.W	1

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1+m	2/1+m	2/2+m	3/2+m	3/2+m	4/2+m	4/2+m	4/2+m

*1 m denotes the number of shifts performed.

SHL

(2) SHL.size R1H, dest

b7							b0	b7							b0
0	1	1	1	0	1	0	SIZE	1	1	1	0	DEST			

dest code

(dsp8 <br> dsp16/abs16)

.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/---	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	--- /R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2+m	2/2+m	2/3+m	3/3+m	3/3+m	4/3+m	4/3+m	4/3+m

*2 m denotes the number of shifts performed.

SHL

(3) SHL.L #IMM, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	1	0	0	DEST	IMM4			

#IMM	IMM4	#IMM	IMM4
+1	0 0 0 0	–1	1 0 0 0
+2	0 0 0 1	–2	1 0 0 1
+3	0 0 1 0	–3	1 0 1 0
+4	0 0 1 1	–4	1 0 1 1
+5	0 1 0 0	–5	1 1 0 0
+6	0 1 0 1	–6	1 1 0 1
+7	0 1 1 0	–7	1 1 1 0
+8	0 1 1 1	–8	1 1 1 1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+m

*2 m denotes the number of shifts performed.

SHL

(4) SHL.L R1H, dest

b7							b0	b7							b0
1	1	1	0	1	0	1	1	0	0	0	DEST	0	0	0	1

dest	DEST
R2R0	0
R3R1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4+m

*1 m denotes the number of shifts performed.

SMOVB

(1) SMOVB.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	0	1

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5+5×m

*2 m denotes the number of transfers performed.

SMOVF

(1) SMOVF.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	0	0

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/5+5×m

*1 m denotes the number of transfers performed.

SSTR

(1) SSTR.size

b7							b0	b7							b0
0	1	1	1	1	1	0	SIZE	1	1	1	0	1	0	1	0

.size	SIZE
.B	0
.W	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3+2×m

*1 m denotes the number of transfers performed.

STC

(1) STC src, dest

b7							b0	b7		b0
0	1	1	1	1	0	1	1	1	SRC	DEST

dest Code

(dsp8

dsp16/abs16)

src	SRC
---	0 0 0
INTBL	0 0 1
INTBH	0 1 0
FLG	0 1 1
ISP	1 0 0
SP	1 0 1
SB	1 1 0
FB	1 1 1

dest		DEST	dest		DEST
Rn	R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/1	2/1	2/2	3/2	3/2	4/2	4/2	4/2

STC

(2) STC PC, dest

b7							b0	b7				b0
0	1	1	1	1	1	0	0	1	1	0	0	DEST

dest Code

(dsp8

dsp16/abs16)

dest		DEST	dest		DEST
Rn	R2R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R3R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	---	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	---	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A1A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	---	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

*1 Marked by - - - cannot be selected.

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3

STCTX

(1) STCTX abs16, abs20

b7							b0	b7							b0		
0	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	abs16	abs20

[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/11+2×m

*1 m denotes the number of transfers performed.

STE

(1) STE.size src, abs20

b7							b0	b7				b0	src code	dest code
0	1	1	1	0	1	0	SIZE	0	0	0	0	SRC	(dsp8 dsp16/abs16)	abs20

.size	SIZE
.B	0
.W	1

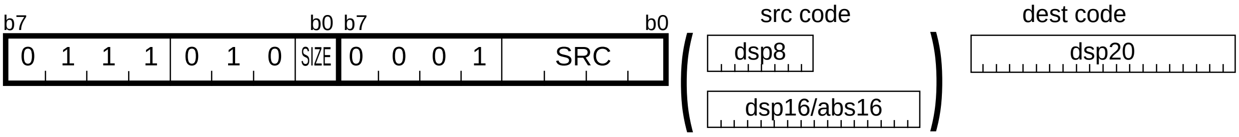
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	5/3	5/3	5/4	6/4	6/4	7/4	7/4	7/4

STE

(2) STE.size src, dsp:20[A0]



.size	SIZE
.B	0
.W	1

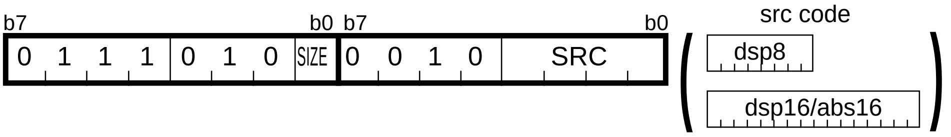
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	5/3	5/3	5/4	6/4	6/4	7/4	7/4	7/4

STE

(3) STE.size src, [A1A0]



.size	SIZE
.B	0
.W	1

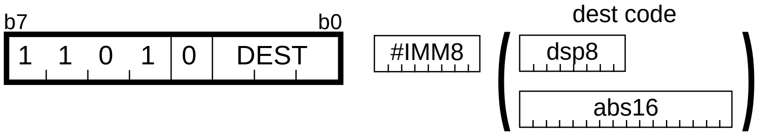
src		SRC	src		SRC
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4

STNZ

(1) STNZ #IMM8, dest



dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

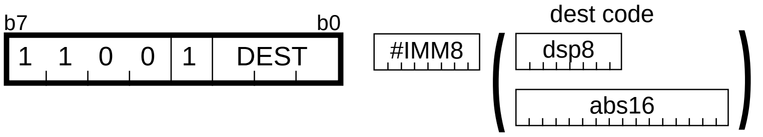
[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

*1 If the Z flag = 0, the number of cycles above is increased by 1.

STZ

(1) STZ #IMM8, dest



dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

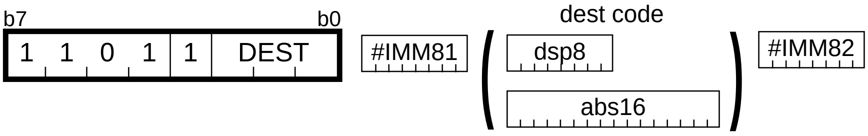
[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

*2 If the Z flag = 1, the number of cycles above is increased by 1.

STZX

(1) STZX #IMM8$_1$, #IMM8$_2$, dest



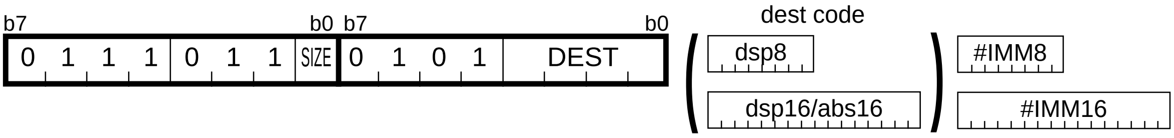
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	3/2	4/3	5/3

SUB

(1) SUB.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

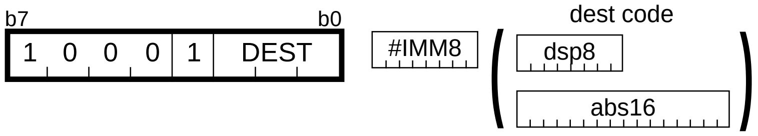
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

(2) SUB.B:S #IMM8, dest



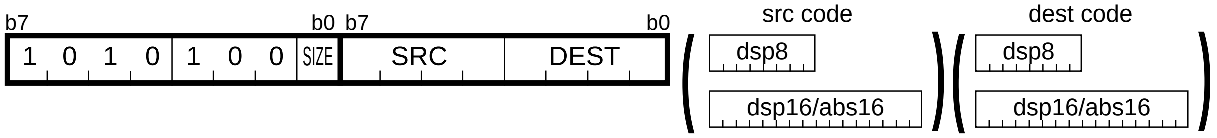
dest		DEST
Rn	R0H	0 1 1
	R0L	1 0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0 1
	dsp:8[FB]	1 1 0
abs16	abs16	1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

SUB

(3) SUB.size:G src, dest



.size	SIZE
.B	0
.W	1

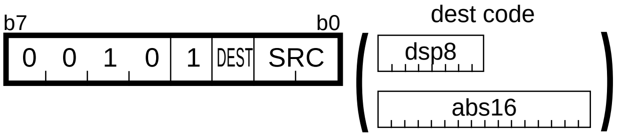
src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

SUB

(4) SUB.B:S src, R0L/R0H



src		SRC
Rn	R0L/R0H	0 0
dsp:8[SB/FB]	dsp:8[SB]	0 1
	dsp:8[FB]	1 0
abs16	abs16	1 1

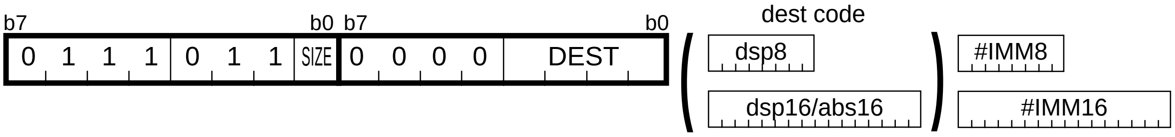
dest	DEST
R0L	0
R0H	1

[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/3	3/3

TST

(1) TST.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

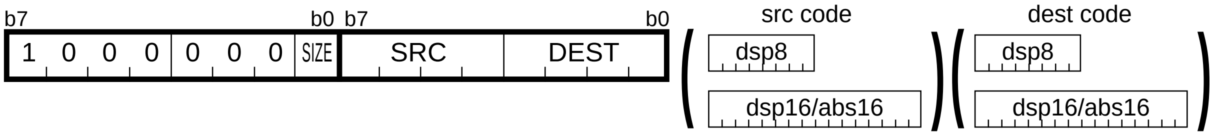
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

TST

(2) TST.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

UND

(1) UND

b7							b0
1	1	1	1	1	1	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/20

WAIT

(1) WAIT

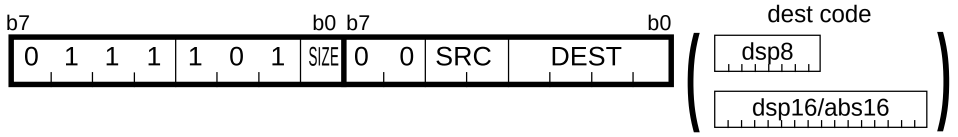
b7							b0	b7							b0
0	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

XCHG

(1) XCHG.size src, dest



.size	SIZE
.B	0
.W	1

src	SRC
R0L/R0	0 0
R0H/R1	0 1
R1L/R2	1 0
R1H/R3	1 1

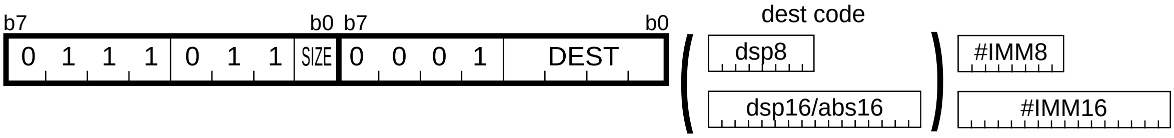
dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	2/4	2/4	2/5	3/5	3/5	4/5	4/5	4/5

XOR

(1) XOR.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		DEST	dest		DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

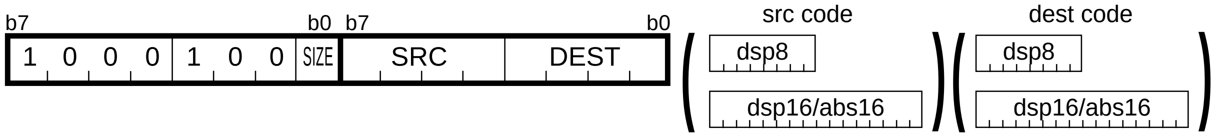
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	5/4

*1 If the size specifier (.size) is (.W), the number of bytes above is increased by 1.

XOR

(2) XOR.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		SRC/DEST	src/dest		SRC/DEST
Rn	R0L/R0	0 0 0 0	dsp:8[An]	dsp:8[A0]	1 0 0 0
	R0H/R1	0 0 0 1		dsp:8[A1]	1 0 0 1
	R1L/R2	0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	1 0 1 0
	R1H/R3	0 0 1 1		dsp:8[FB]	1 0 1 1
An	A0	0 1 0 0	dsp:16[An]	dsp:16[A0]	1 1 0 0
	A1	0 1 0 1		dsp:16[A1]	1 1 0 1
[An]	[A0]	0 1 1 0	dsp:16[SB]	dsp:16[SB]	1 1 1 0
	[A1]	0 1 1 1	abs16	abs16	1 1 1 1

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB]	abs16
Rn	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
An	2/2	2/2	2/3	3/3	3/3	4/3	4/3	4/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	4/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	5/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
dsp:16[SB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	6/4

Chapter 5

Interrupt

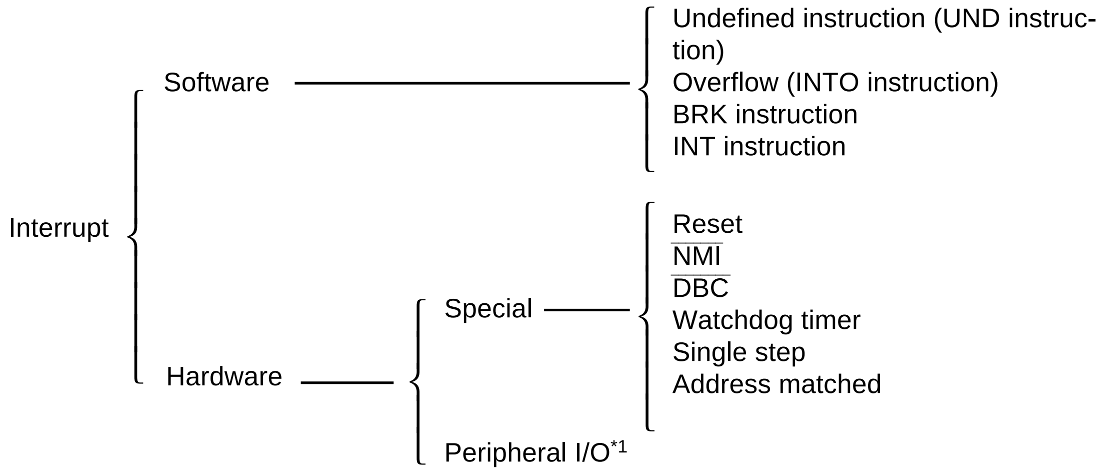
5.1 Outline of Interrupt
5.2 Interrupt Control
5.3 Interrupt Sequence
5.4 Return from Interrupt Routine
5.5 Interrupt Priority
5.6 Multiple Interrupts
5.7 Precautions for Interrupts

5.1 Outline of Interrupt

When an interrupt request is acknowledged, control branches to the interrupt routine that is set to an interrupt vector table. Each interrupt vector table must have had the start address of its corresponding interrupt routine set. For details about the interrupt vector table, refer to Section 1.10, “Vector Table.”

5.1.1 Types of Interrupts

Figure 5.1.1 lists the types of interrupts. Table 5.1.1 lists the source of interrupts (nonmaskable) and the fixed vector tables.



*1 Peripheral I/O interrupts are generated by the peripheral functions built into the microcomputer system.

Figure 5.1.1. Classification of interrupts

Table 5.1.1 Interrupt Source (Nonmaskable) and Fixed Vector Table

Interrupt source	Vector table addresses Address (L) to address (H)	Remarks
Undefined instruction	FFFDC$_{16}$ to FFFDF$_{16}$	Interrupt generated by the UND instruction.
Overflow	FFFE0$_{16}$ to FFFE3$_{16}$	Interrupt generated by the INTO instruction.
BRK instruction	FFFE4$_{16}$ to FFFE7$_{16}$	Executed beginning from address indicated by vector in variable vector table if all vector contents are FF$_{16}$
Address match	FFFE8$_{16}$ to FFFEB$_{16}$	Can be controlled by an interrupt enable bit.
Single step*1	FFFEC$_{16}$ to FFFEF$_{16}$	Normally do not use this interrupt.
Watchdog timer	FFFF0$_{16}$ to FFFF3$_{16}$	
$\overline{\text{DBC}}$ *1	FFFF4$_{16}$ to FFFF7$_{16}$	Normally do not use this interrupt.
$\overline{\text{NMI}}$	FFFF8$_{16}$ to FFFFB$_{16}$	External interrupt generated by driving $\overline{\text{NMI}}$ pin low.
Reset	FFFFC$_{16}$ to FFFFF$_{16}$	

*1 This interrupt is used exclusively for debugger purposes.

- ■ Maskable interrupt: This type of interrupt can be controlled by using the I flag to enable (or disable) an interrupt or by changing the interrupt priority level.
- ■ Nonmaskable interrupt: This type of interrupt cannot be controlled by using the I flag to enable (or disable) an interrupt or by changing the interrupt priority level.

5.1.2 Software Interrupts

Software interrupts are generated by some instruction that generates an interrupt request when executed. Software interrupts are nonmaskable interrupts.

(1) Undefined-instruction interrupt

This interrupt occurs when the UND instruction is executed.

(2) Overflow interrupt

This interrupt occurs if the INTO instruction is executed when the O flag is 1.
The following lists the instructions that cause the O flag to change:
ABS, ADC, ADCF, ADD, CMP, DIV, DIVU, DIVX, NEG, RMPA, SBB, SHA, SUB

(3) BRK interrupt

This interrupt occurs when the BRK instruction is executed.

(4) INT instruction interrupt

This interrupt occurs when the INT instruction is executed after specifying a software interrupt number from 0 to 63. Note that software interrupt numbers 0 to 31 are assigned to peripheral I/O interrupts. This means that by executing the INT instruction, you can execute the same interrupt routine as used in peripheral I/O interrupts.
The stack pointer used in INT instruction interrupt varies depending on the software interrupt number. For software interrupt numbers 0 to 31, the U flag is saved when an interrupt occurs and the U flag is cleared to 0 to choose the interrupt stack pointer (ISP) before executing the interrupt sequence. The previous U flag before the interrupt occurred is restored when control returns from the interrupt routine. For software interrupt numbers 32 to 63, such stack pointer switchover does not occur.

5.1.3 Hardware Interrupts

There are Two types in hardware Interrupts; special interrupts and Peripherai I/O interrupts.

(1) Special interrupts

Special interrupts are nonmaskable interrupts.

- **Reset**

  A reset occurs when the $\overline{\text{RESET}}$ pin is pulled low.

- **$\overline{\text{NMI}}$ interrupt**

  This interrupt occurs when the $\overline{\text{NMI}}$ pin is pulled low.

- **$\overline{\text{DBC}}$ interrupt**

  This interrupt is used exclusively for debugger purposes. You normally do not need to use this interrupt.

- **Watchdog timer interrupt**

  This interrupt is caused by the watchdog timer.

- **Single-step interrupt**

  This interrupt is used exclusively for debugger purposes. You normally do not need to use this interrupt. A single-step interrupt occurs when the D flag is set (= 1); in this case, an interrupt is generated each time an instruction is executed.

- **Address-match interrupt**

  This interrupt occurs when the program's execution address matches the content of the address match register while the address match interrupt enable bit is set (= 1).

  This interrupt does not occur if any address other than the start address of an instruction is set in the address match register.

(2) Peripheral I/O interrupts

These interrupts are generated by the peripheral functions built into the microcomputer system. The types of built-in peripheral functions vary with each M16C model, so do the types of interrupt causes. The interrupt vector table uses the same software interrupt numbers 0–31 that are used by the INT instruction. Peripheral I/O interrupts are maskable interrupts. For details about peripheral I/O interrupts, refer to the M16C User’s Manual.

5.2 Interrupt Control

The following explains how to enable/disable maskable interrupts and set acknowledge priority. The explanation here does not apply to non-maskable interrupts.

Maskable interrupts are enabled and disabled by using the interrupt enable flag (I flag), interrupt priority level select bit, and processor interrupt priority level (IPL). Whether there is any interrupt requested is indicated by the interrupt request bit. The interrupt request bit and interrupt priority level select bit are arranged in the interrupt control register provided for each specific interrupt. The interrupt enable flag (I flag) and processor interrupt priority level (IPL) are arranged in the flag register (FLG).

For details about the memory allocation and the configuration of interrupt control registers, refer to the M16C User's Manual.

5.2.1 Interrupt Enable Flag (I Flag)

The interrupt enable flag (I flag) is used to disable/enable maskable interrupts. When this flag is set (= 1), all maskable interrupts are enabled; when the flag is cleared to 0, they are disabled. This flag is automatically cleared to 0 after a reset is cleared.

When the I flag is changed, the altered flag status is reflected in determining whether or not to accept an interrupt request at the following timing:

- If the flag is changed by an REIT instruction, the changed status takes effect beginning with that REIT instruction.
- If the flag is changed by an FCLR, FSET, POPC, or LDC instruction, the changed status takes effect beginning with the next instruction.

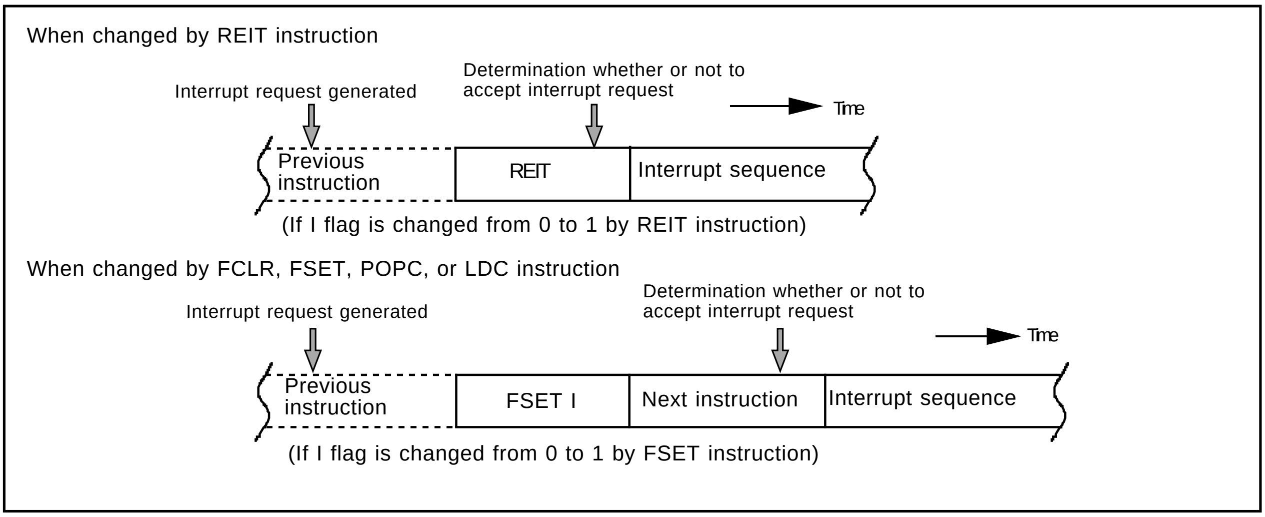

When changed by REIT instruction
Interrupt request generated
Determination whether or not to accept interrupt request
Time
Previous instruction
REIT
Interrupt sequence
(If I flag is changed from 0 to 1 by REIT instruction)
When changed by FCLR, FSET, POPC, or LDC instruction
Interrupt request generated
Determination whether or not to accept interrupt request
Time
Previous instruction
FSET I
Next instruction
Interrupt sequence
(If I flag is changed from 0 to 1 by FSET instruction)

Figure 5.2.1 Timing at which changes of I flag are reflected in interrupt handling

5.2.2 Interrupt Request Bit

This bit is set (= 1) when an interrupt request is generated. This bit remains set until the interrupt request is acknowledged. The bit is cleared to 0 when the interrupt request is acknowledged.

This bit can be cleared to 0 (but cannot be set to 1) in software.

5.2.3 Interrupt Priority Level Select Bit and Processor Interrupt Priority Level (IPL)

Interrupt priority levels are set by the interrupt priority select bit in an interrupt control register. When an interrupt request is generated, the interrupt priority level of this interrupt is compared with the processor interrupt priority level (IPL). This interrupt is enabled only when its interrupt priority level is greater than the processor interrupt priority level (IPL). This means that you can disable any particular interrupt by setting its interrupt priority level to 0.

Table 5.2.1 shows how interrupt priority levels are set. Table 5.2.2 shows interrupt enable levels in relation to the processor interrupt priority level (IPL).

The following lists the conditions under which an interrupt request is acknowledged:

- Interrupt enable flag (I flag) = 1
- Interrupt request bit = 1
- Interrupt priority level > Processor interrupt priority level (IPL)

The interrupt enable flag (I flag), interrupt request bit, interrupt priority level select bit, and the processor interrupt priority level (IPL) all are independent of each other, so they do not affect any other bit.

Table 5.2.1 Interrupt Priority Levels

Interrupt priority level select bit	Interrupt priority level	Priority order
b2 b1 b0 0 0 0	Level 0(interrupt disabled)	———
0 0 1	Level 1	Low
0 1 0	Level 2	
0 1 1	Level 3	
1 0 0	Level 4	
1 0 1	Level 5	
1 1 0	Level 6	
1 1 1	Level 7	High

Table 5.2.2 IPL and Interrupt Enable Levels

Processor interrupt priority level (IPL)	Enabled interrupt priority levels
IPL_2 IPL_1 IPL_0 0 0 0	Interrupt levels 1 and above are enabled.
0 0 1	Interrupt levels 2 and above are enabled.
0 1 0	Interrupt levels 3 and above are enabled.
0 1 1	Interrupt levels 4 and above are enabled.
1 0 0	Interrupt levels 5 and above are enabled.
1 0 1	Interrupt levels 6 and above are enabled.
1 1 0	Interrupt levels 7 and above are enabled.
1 1 1	All maskable interrupts are disabled.

When the processor interrupt priority level (IPL) or the interrupt priority level of some interrupt is changed, the altered level is reflected in interrupt handling at the following timing:

- If the processor interrupt priority level (IPL) is changed by an REIT instruction, the changed level takes effect beginning with the instruction that is executed two clock periods after the last clock of the REIT instruction.
- If the processor interrupt priority level (IPL) is changed by a POPC, LDC, or LDIPL instruction, the changed level takes effect beginning with the instruction that is executed three clock periods after the last clock of the instruction used.
- If the interrupt priority level of a particular interrupt is changed by an instruction such as MOV, the changed level takes effect beginning with the instruction that is executed two clock or three clock periods after the last clock of the instruction used.

  M16C/60, M16C/61 group, and M16C/20 series: two clock

  M16C/60 series after M16C/62 group (it has M16C/62 group), M16C/Tiny series : three clock

5.2.4 Rewrite the Interrupt Control Register

(1) The interrupt control register for any interrupt should be modified in places where no requests for that interrupt may occur. Otherwise, disable the interrupt before rewriting the interrupt control register.

(2) To rewrite the interrupt control register for any interrupt after disabling that interrupt, be careful with the instruction to be used.

Changing any bit other than the IR bit

If while executing an instruction, a request for an interrupt controlled by the register being modified occurs, the IR bit in the register may not be set to “1” (interrupt requested), with the result that the interrupt request is ignored. If such a situation presents a problem, use the instructions shown below to modify the register.

Usable instructions: AND, OR, BCLR, BSET

Changing the IR bit

Depending on the instruction used, the IR bit may not always be cleared to “0” (interrupt not requested). Therefore, be sure to use the MOV instruction to clear the IR bit.

(3) When using the I flag to disable an interrupt, refer to the sample program fragments shown below as you set the I flag. (Refer to (2) for details about rewrite the interrupt control registers in the sample program fragments.)

Examples 1 through 3 show how to prevent the I flag from being set to “1” (interrupts enabled) before the interrupt control register is rewrited, owing to the effects of the internal bus and the instruction queue buffer.

Example 1:Using the NOP instruction to keep the program waiting until the interrupt control register is modified

```
INT_SWITCH1:
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        NOP                        ; Four NOP instructions are required when using HOLD function.
        NOP                        ; Refer to hardware manual about the number of NOP
                                   ; instruction
        FSET      I                ; Enable interrupts.
```

Example 2:Using the dummy read to keep the FSET instruction waiting

```
INT_SWITCH2:
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        MOV.W     MEM, R0          ; Dummy read.
        FSET      I                ; Enable interrupts.
```

Example 3:Using the POPC instruction to changing the I flag

```
INT_SWITCH3:
        PUSHC     FLG
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        POPC      FLG              ; Enable interrupts.
```

5.3 Interrupt Sequence

An interrupt sequence — what are performed over a period from the instant an interrupt is accepted to the instant the interrupt routine is executed — is described here.

If an interrupt occurs during execution of an instruction, the processor determines its priority when the execution of the instruction is completed, and transfers control to the interrupt sequence from the next cycle. If an interrupt occurs during execution of either the SMOVB, SMOVF, SSTR or RMPA instruction, the processor temporarily suspends the instruction being executed, and transfers control to the interrupt sequence.

In the interrupt sequence, the processor carries out the following in sequence given:

(1) CPU gets the interrupt information (the interrupt number and interrupt request level) by reading address 00000_{16}.

(2) Saves the content of the flag register (FLG) as it was immediately before the start of interrupt sequence in the temporary register (Note) within the CPU.

(3) Sets the interrupt enable flag (I flag), the debug flag (D flag), and the stack pointer select flag (U flag) to “0” (the U flag, however does not change if the INT instruction, in software interrupt numbers 32 through 63, is executed)

(4) Saves the content of the temporary register (Note 1) within the CPU in the stack area.

(5) Saves the content of the program counter (PC) in the stack area.

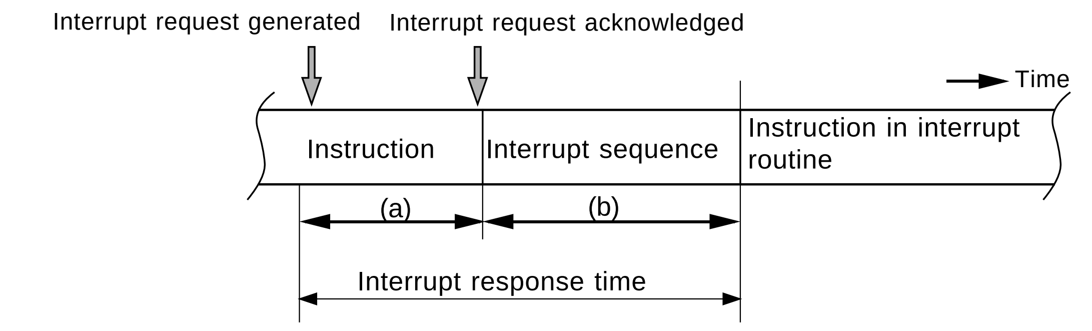
(6) Sets the interrupt priority level of the accepted instruction in the IPL.

After the interrupt sequence is completed, the processor resumes executing instructions from the first address of the interrupt routine.

Note: This register cannot be utilized by the user.

5.3.1 Interrupt Response Time

The interrupt response time means a period of time from when an interrupt request is generated till when the first instruction of the interrupt routine is executed. This period consists of time (a) from when an interrupt request is generated to when the instruction then under way is completed and time (b) in which an interrupt sequence is executed. Figure 5.3.1 shows the interrupt response time.


Interrupt request generated
Interrupt request acknowledged
Time
Instruction
Interrupt sequence
Instruction in interrupt routine
(a)
(b)
Interrupt response time

(a) Time from when interrupt request is generated to when the instruction then under execution is completed

(b) Time in which the interrupt sequence is executed

Figure 5.3.1. Interrupt response time

Time (a) varies with each instruction being executed. The DIVX instruction requires a maximum time that consists of 30 cycles (without wait state) .

Time (b) is shown below.

Table 5.3.1 Interrupt Sequence Execution Time

Interrupt vector address	Stack pointer (SP) value	16 bits data bus Without wait state	8 bits data bus Without wait state
Even address	Even address	18 cycle*1	20 cycle*1
Even address	Odd address	19 cycle*1	20 cycle*1
Odd address*2	Even address	19 cycle*1	20 cycle*1
Odd address*2	Odd address	20 cycle*1	20 cycle*1

*1 Add two cycles for the $\overline{\text{DBC}}$ interrupt. Add one cycle for the address match and single-step interrupts.

*2 Allocate interrupt vector addresses in even addresses as must as possible.

5.3.2 Changes of IPL When Interrupt Request Acknowledged

When an interrupt request is acknowledged, the interrupt priority level of the acknowledged interrupt is set to the processor interrupt priority level (IPL).

If an interrupt request is acknowledged that does not have an interrupt priority level, the value shown in Table 5.3.2 is set to the IPL.

Table 5.3.2 Relationship between Interrupts without Interrupt Priority Levels and IPL

Interrupt sources without interrupt priority levels	Value that is set to IPL
Watchdog timer, $\overline{\text{NMI}}$	7
Reset	0
Other	Not changed

5.3.3 Saving Registers

In an interrupt sequence, only the contents of the flag register (FLG) and program counter (PC) are saved to the stack area.

The order in which these contents are saved is as follows: First, the 4 high-order bits of the program counter and 4 high-order bits and 8 low-order bits of the FLG register for a total of 16 bits are saved to the stack area. Next, the 16 low-order bits of the program counter are saved. Figure 5.3.2 shows the stack status before an interrupt request is acknowledged and the stack status after an interrupt request is acknowledged.

If there are any other registers you want to be saved, save them in software at the beginning of the interrupt routine. The PUSHM instruction allows you to save all registers except the stack pointer (SP) by a single instruction.

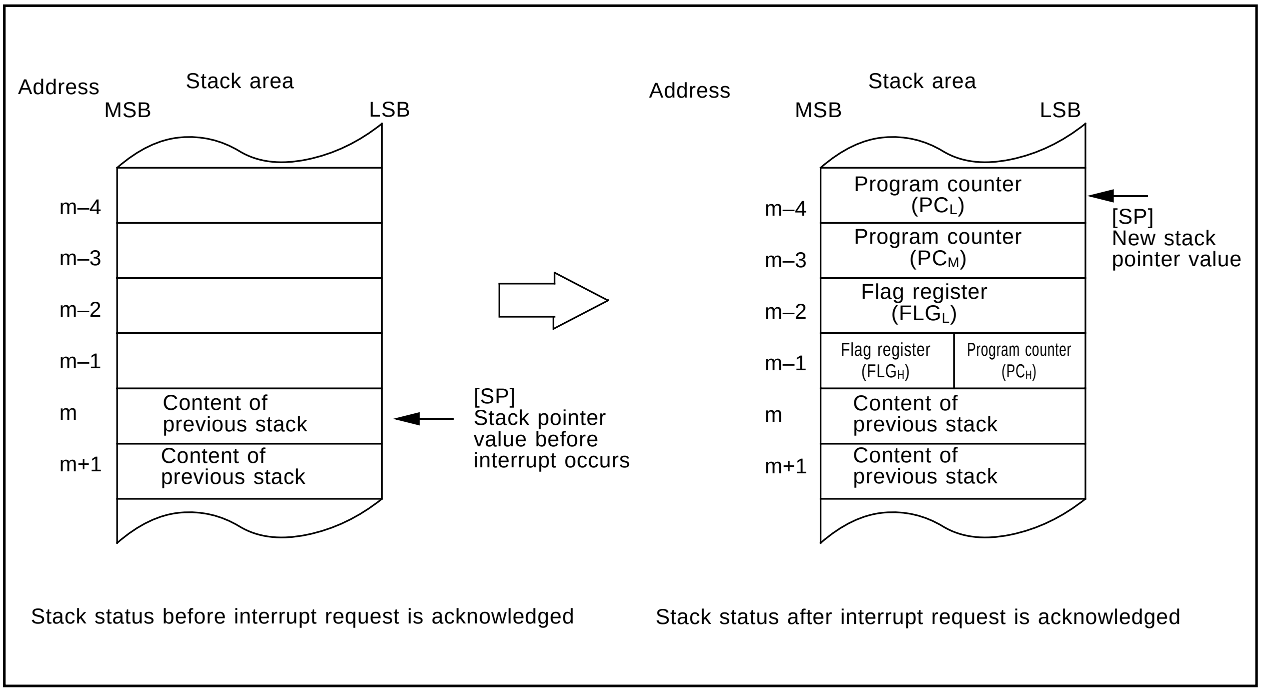
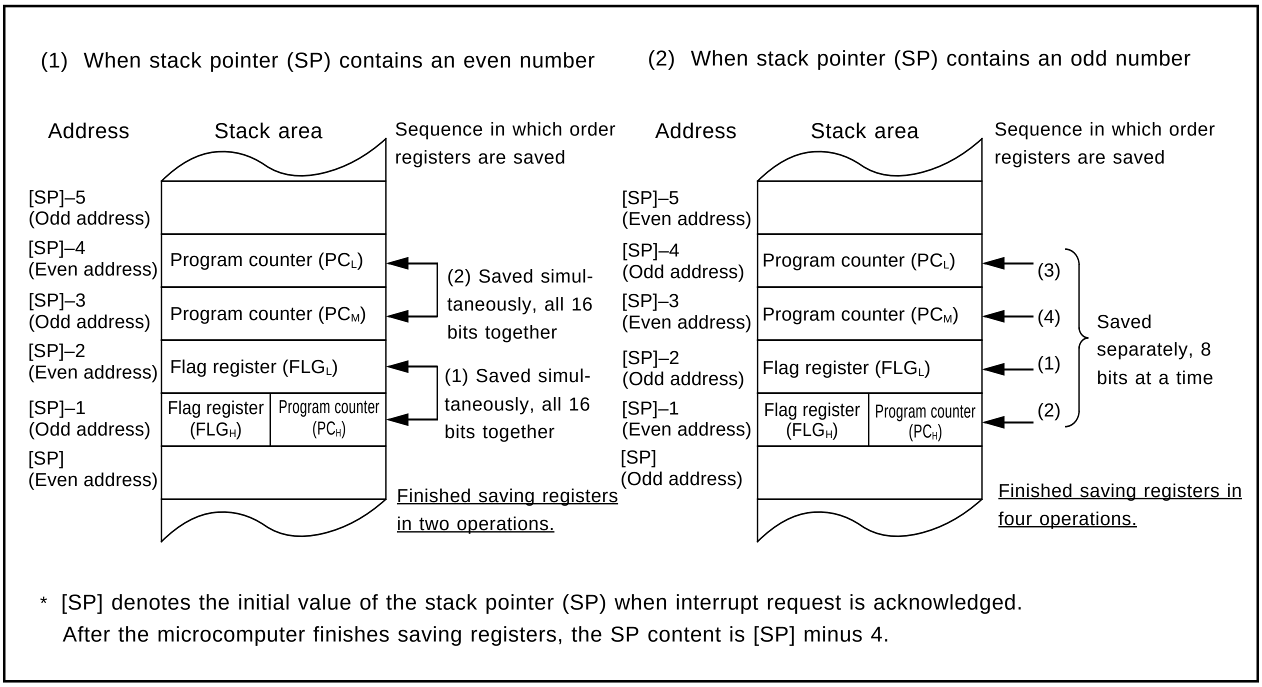

Address
Stack area
MSB
LSB
m–4
m–3
m–2
m–1
m
m+1
Content of previous stack
Content of previous stack
[SP] Stack pointer value before interrupt occurs
Stack status before interrupt request is acknowledged
Address
Stack area
MSB
LSB
m–4
m–3
m–2
m–1
m
m+1
Program counter (PCL)
Program counter (PCM)
Flag register (FLGL)
Flag register (FLGH)
Program counter (PCH)
Content of previous stack
Content of previous stack
[SP] New stack pointer value
Stack status after interrupt request is acknowledged

Figure 5.3.2 Stack status before and after an interrupt request is acknowledged

The register save operation performed in an interrupt sequence differs depending on whether the content of the stack pointer (SP)[*1] is an even or an odd number when an interrupt request is acknowledged. If the stack pointer (SP)[*1] indicates an even number, the contents of the flag register (FLG) and program counter (PC) each are saved simultaneously all 16 bits together. If the stack pointer indicates an odd number, the register contents each are saved in two operations 8 bits at a time. Figure 5.3.3 shows how registers are saved in each case.

*1 Stack pointer indicated by the U flag.



* [SP] denotes the initial value of the stack pointer (SP) when interrupt request is acknowledged.
After the microcomputer finishes saving registers, the SP content is [SP] minus 4.

Figure 5.3.3 Operations to save registers

5.4 Return from Interrupt Routine

As you execute the REIT instruction at the end of the interrupt routine, the contents of the flag register (FLG) and program counter (PC) that have been saved to the stack area immediately preceding the interrupt sequence are automatically restored. Then control returns to the routine that was under execution before the interrupt request was acknowledged, and processing is resumed from where control left off. If there are any registers you saved via software in the interrupt routine, be sure to restore them using an instruction (e.g., POPM instruction) before executing the REIT instruction.

5.5 Interrupt Priority

If two or more interrupt requests are sampled active at the same time, whichever interrupt request is acknowledged that has the highest priority.

Maskable interrupts (Peripheral I/O interrupts) can be assigned any desired priority by setting the interrupt priority level select bit accordingly. If some maskable interrupts are assigned the same priority level, the priority between these interrupts is resolved by the priority that is set in hardware[*1].

Certain nonmaskable interrupts such as a reset (reset is given the highest priority) and watchdog timer interrupt have their priority levels set in hardware. Figure 5.5.1 lists the hardware priority levels of these interrupts.

Software interrupts are not subjected to interrupt priority. They always cause control to branch to an interrupt routine whenever the relevant instruction is executed.

*1 Hardware priority varies with each M16C model. Please refer to your M16C User’s Manual.

Reset > $\overline{\text{NMI}}$ > $\overline{\text{DBC}}$ > Watchdog timer > Peripheral I/O > Single step > Address match

Figure 5.5.1. Interrupt priority that is set in hardware

5.6 Multiple Interrupts

The following shows the internal bit states when control has branched to an interrupt routine:

- The interrupt enable flag (I flag) is cleared to 0 (interrupts disabled).
- The interrupt request bit for the acknowledged interrupt is cleared to 0.
- The processor interrupt priority level (IPL) equals the interrupt priority level of the acknowledged interrupt.

By setting the interrupt enable flag (I flag) (= 1) in the interrupt routine, you can reenable interrupts so that an interrupt request can be acknowledged that has higher priority than the processor interrupt priority level (IPL). Figure 5.6.1 shows how multiple interrupts are handled.
The interrupt requests that have not been acknowledged for their low interrupt priority level are kept pending. When the IPL is restored by an REIT instruction and interrupt priority is resolved against it, the pending interrupt request is acknowledged if the following condition is met:

Interrupt priority level of pending interrupt request > Restored processor interrupt priority level (IPL)

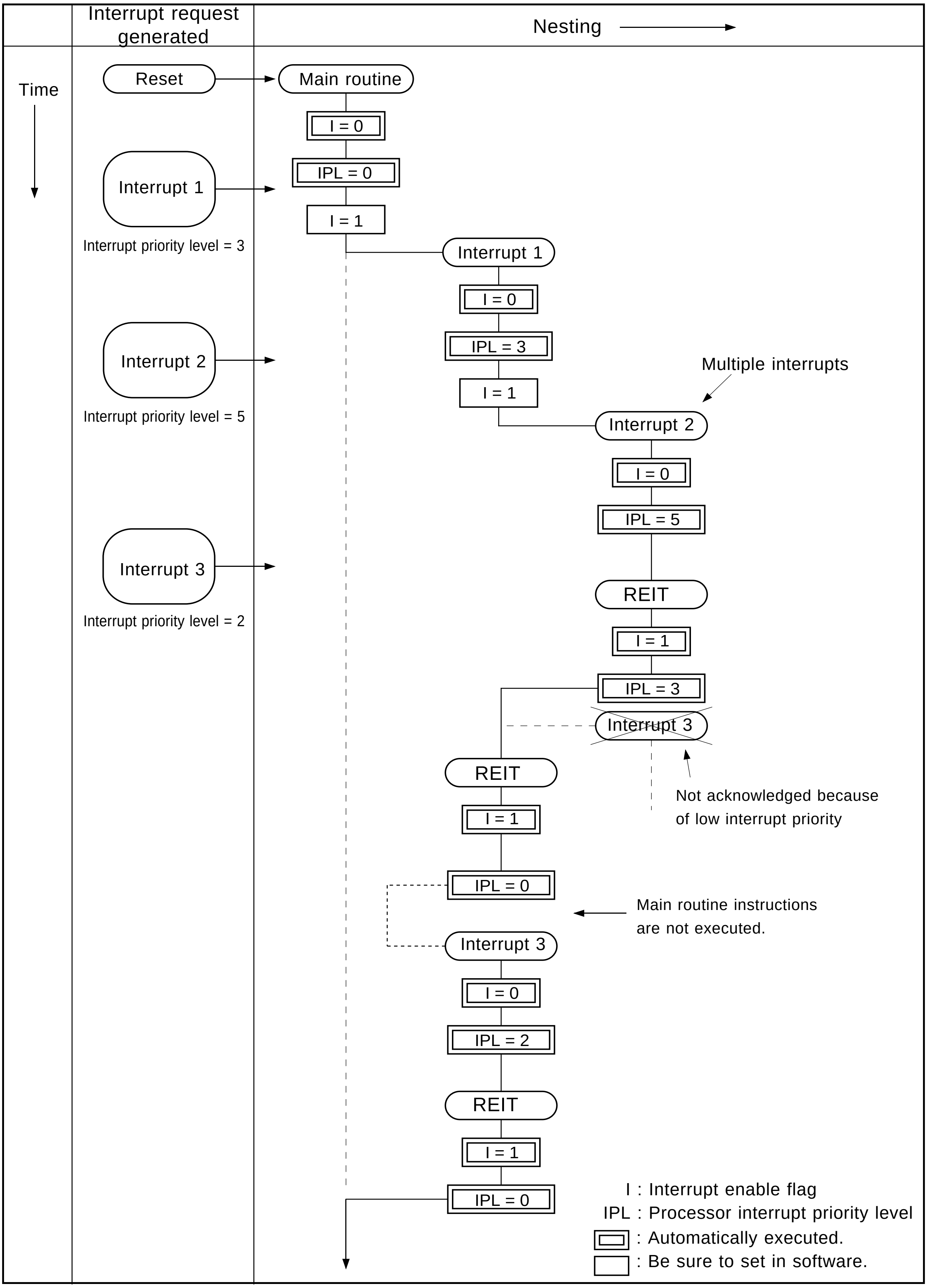

Interrupt request generated
Nesting
Time
Reset
Main routine
I = 0
IPL = 0
I = 1
Interrupt 1
Interrupt priority level = 3
Interrupt 1
I = 0
IPL = 3
I = 1
Interrupt 2
Interrupt priority level = 5
Multiple interrupts
Interrupt 2
I = 0
IPL = 5
Interrupt 3
Interrupt priority level = 2
REIT
I = 1
IPL = 3
Interrupt 3
Not acknowledged because of low interrupt priority
REIT
I = 1
IPL = 0
Main routine instructions are not executed.
Interrupt 3
I = 0
IPL = 2
REIT
I = 1
IPL = 0
I : Interrupt enable flag
IPL : Processor interrupt priority level
▭ : Automatically executed.
□ : Be sure to set in software.

Figure 5.6.1. Multiple interrupts

5.7 Precautions for Interrupts

5.7.1 Reading address 00000_{16}

Do not read the address 00000_{16} in a program. When a maskable interrupt request is accepted, the CPU reads interrupt information (interrupt number and interrupt request priority level) from the address 00000_{16} during the interrupt sequence. At this time, the IR bit for the accepted interrupt is cleared to “0”. If the address 00000_{16} is read in a program, the IR bit for the interrupt which has the highest priority among the enabled interrupts is cleared to “0”. This causes a problem that the interrupt is canceled, or an unexpected interrupt request is generated.

5.7.2 Setting the SP

Set any value in the SP(USP, ISP) before accepting an interrupt. The SP(USP, ISP) is cleared to ‘0000_{16}’ after reset. Therefore, if an interrupt is accepted before setting any value in the SP(USP, ISP), the program may go out of control.

Especially when using $\overline{\text{NMI}}$ interrupt, set a value in the ISP at the beginning of the program. For the first and only the first instruction after reset, all interrupts including $\overline{\text{NMI}}$ interrupt are disabled.

5.7.3 Rewrite the Interrupt Control Register

(1) The interrupt control register for any interrupt should be modified in places where no requests for that interrupt may occur. Otherwise, disable the interrupt before rewriting the interrupt control register.

(2) To rewrite the interrupt control register for any interrupt after disabling that interrupt, be careful with the instruction to be used.

Changing any bit other than the IR bit

If while executing an instruction, a request for an interrupt controlled by the register being modified occurs, the IR bit in the register may not be set to “1” (interrupt requested), with the result that the interrupt request is ignored. If such a situation presents a problem, use the instructions shown below to modify the register.

Usable instructions: AND, OR, BCLR, BSET

Changing the IR bit

Depending on the instruction used, the IR bit may not always be cleared to “0” (interrupt not requested). Therefore, be sure to use the MOV instruction to clear the IR bit.

(3) When using the I flag to disable an interrupt, refer to the sample program fragments shown below as you set the I flag. (Refer to (2) for details about rewrite the interrupt control registers in the sample program fragments.)

Examples 1 through 3 show how to prevent the I flag from being set to “1” (interrupts enabled) before the interrupt control register is rewrited, owing to the effects of the internal bus and the instruction queue buffer.

Example 1:Using the NOP instruction to keep the program waiting until the interrupt control register is modified

```
INT_SWITCH1:
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        NOP                        ; Four NOP instructions are required when using HOLD function.
        NOP                        ; Refer to hardware manual about the number of NOP
                                   ; instruction
        FSET      I                ; Enable interrupts.
```

Example 2:Using the dummy read to keep the FSET instruction waiting

```
INT_SWITCH2:
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        MOV.W     MEM, R0          ; Dummy read.
        FSET      I                ; Enable interrupts.
```

Example 3:Using the POPC instruction to changing the I flag

```
INT_SWITCH3:
        PUSHC     FLG
        FCLR      I                ; Disable interrupts.
        AND.B     #00h, 0055h      ; Set the TA0IC register to “0016”.
        POPC      FLG              ; Enable interrupts.
```

Chapter 6

Calculation Number of Cycles

6.1 Instruction queue buffer

6.1 Instruction queue buffer

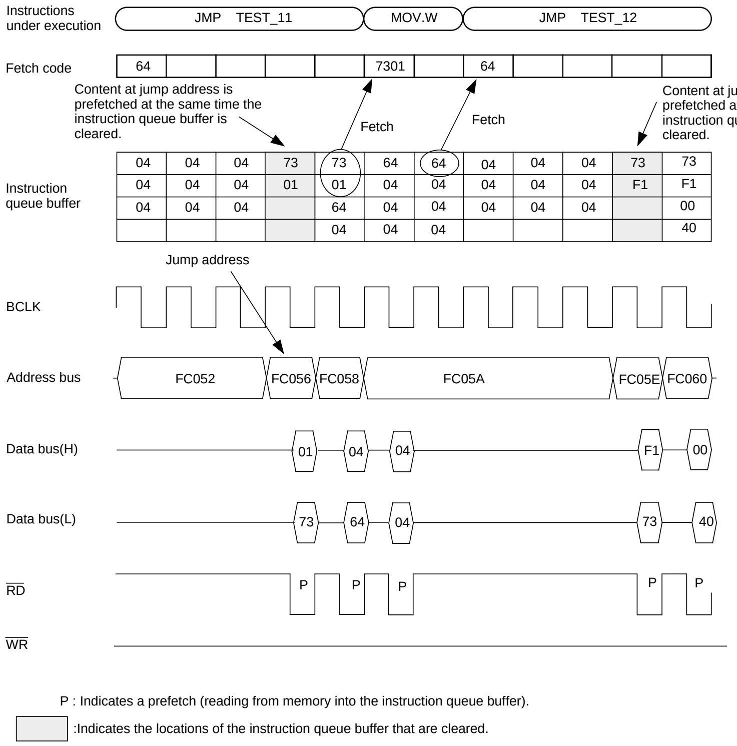
The M16C/60, M16C/20, M16C/Tiny series have 4-stage (4-byte) instruction queue buffers. If the instruction queue buffer has a free space when the CPU can use the bus, instruction codes are taken into the instruction queue buffer. This is referred to as “prefetch”. The CPU reads (fetches) these instruction codes from the instruction queue buffer as it executes a program.

Explanation about the number of cycles in Chapter 4 assumes that all the necessary instruction codes are placed in the instruction queue buffer, and that data is read or written to the memory connected via a 16-bit bus (including the internal memory) beginning with even addresses without software wait or $\overline{\text{RDY}}$ or other wait states. In the following cases, more cycles may be needed than the number of cycles shown in this manual:

- When not all of the instruction codes needed by the CPU are placed in the instruction queue buffer...
  Instruction codes are read in until all of the instruction codes required for program execution are available. Furthermore, the number of read cycles increases in the following cases:
  (1) The number of read cycles increases as many as the number of wait cycles incurred when reading instruction codes from an area in which software wait or $\overline{\text{RDY}}$ or other wait states exist.
  (2) When reading instruction codes from memory chips connected to an 8-bit bus, more read cycles are required than for 16-bit bus.
- When reading or writing data to an area in which software wait or $\overline{\text{RDY}}$ or other wait states exist...
  The number of read or write cycles increases as many as the number of wait cycles incurred.
- When reading or writing 16-bit data to memory chips connected to an 8-bit bus...
  The memory is accessed twice to read or write one 16-bit data. Therefore, the number of read or write cycles increases by one for each 16-bit data read or written.
- When reading or writing 16-bit data to memory chips connected to a 16-bit bus beginning with an odd address...
  The memory is accessed twice to read or write one 16-bit data. Therefore, the number of read or write cycles increases by one for each 16-bit data read or written.

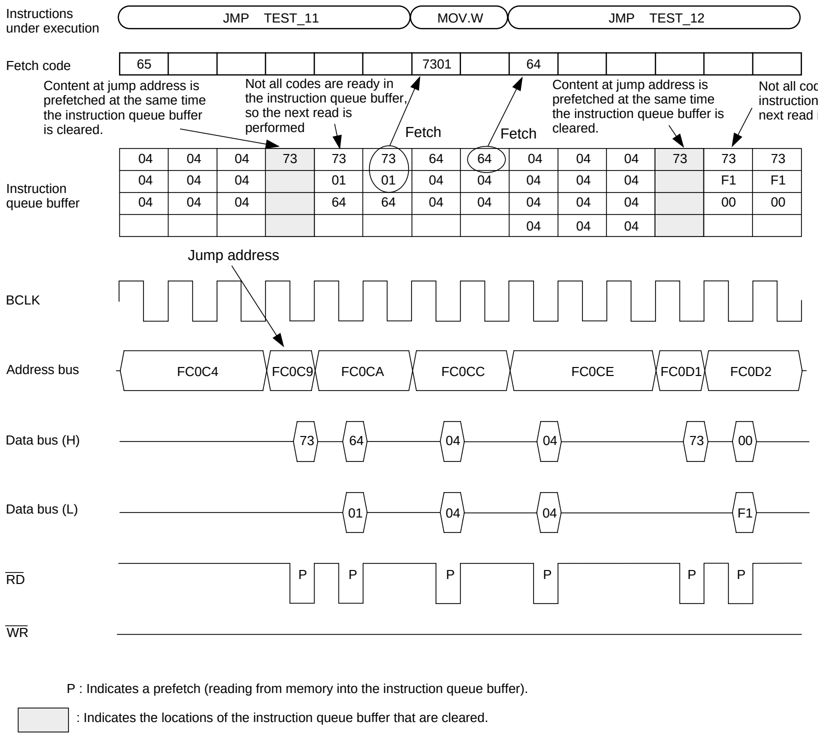
Note that if prefetch and data access occur in the same timing, data access has priority. Also, if more than three bytes of instruction codes exist in the instruction queue buffer, the CPU assumes there is no free space in the instruction queue buffer and, therefore, does not prefetch instruction code.

Figures 6.1.1 to 6.1.8 show examples of instruction queue buffer operation and CPU execution cycles.


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
64
7301
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
04 04 04 73 73 64 64 04 04 04 73 73
04 04 04 01 01 04 04 04 04 04 F1 F1
04 04 04 64 04 04 04 04 04 00
04 04 04 40
Jump address
BCLK
Address bus
FC052
FC056
FC058
FC05A
FC05E
FC060
Data bus(H)
01
04
04
F1
00
Data bus(L)
73
64
04
73
40
RD
P P P P P
WR
P : Indicates a prefetch (reading from memory into the instruction queue buffer).
:Indicates the locations of the instruction queue buffer that are cleared.

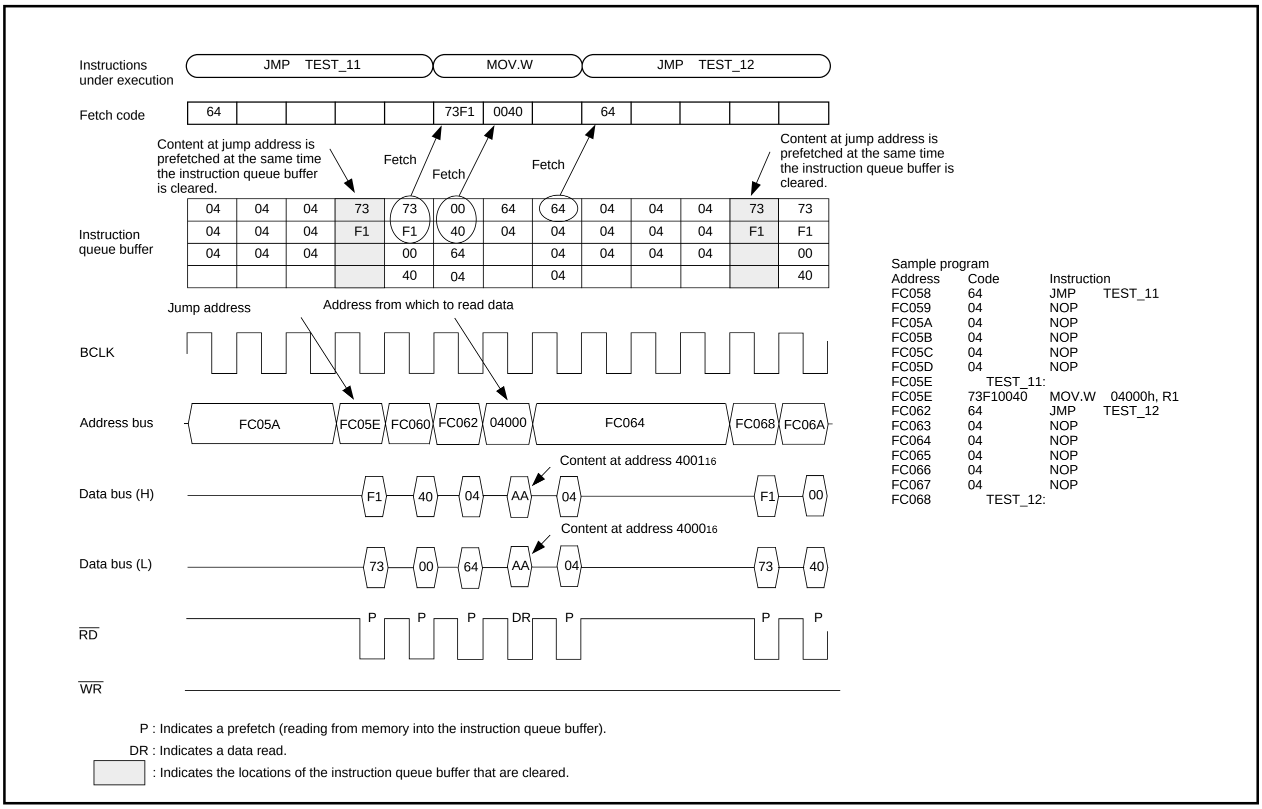
```
Sample program
Address  Code        Instruction
FC050    64          JMP      TEST_11
FC051    04          NOP
FC052    04          NOP
FC053    04          NOP
FC054    04          NOP
FC055    04          NOP
FC056       TEST_11:
FC056    7301        MOV.W    R0,R1
FC058    64          JMP      TEST_12
FC059    04          NOP
FC05A    04          NOP
FC05B    04          NOP
FC05C    04          NOP
FC05D    04          NOP
FC05E       TEST_12:
```

Figure 6.1.1. When executing a register transfer instruction starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
65
7301
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Not all codes are ready in the instruction queue buffer, so the next read is performed
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Not all codes are ready in the instruction queue buffer, so the next read is performed
Instruction queue buffer
04 04 04 73 73 73 64 64 04 04 04 73 73 73
04 04 04 01 01 04 04 04 04 04 F1 F1
04 04 04 64 64 04 04 04 04 04 00 00
04 04 04
Jump address
BCLK
Address bus
FC0C4
FC0C9
FC0CA
FC0CC
FC0CE
FC0D1
FC0D2
Data bus (H)
73
64
04
04
73
00
Data bus (L)
01
04
04
F1
RD
P P P P P P
WR
P : Indicates a prefetch (reading from memory into the instruction queue buffer).
: Indicates the locations of the instruction queue buffer that are cleared.

```
Sample program
Address  Code        Instruction
FC0C2    65          JMP      TEST_11
FC0C3    04          NOP
FC0C4    04          NOP
FC0C5    04          NOP
FC0C6    04          NOP
FC0C7    04          NOP
FC0C8    04          NOP
FC0C9       TEST_11:
FC0C9    7301        MOV.W       R0,R1
FC0CB    64          JMP      TEST_12
FC0CC    04          NOP
FC0CD    04          NOP
FC0CE    04          NOP
FC0CF    04          NOP
FC0D0    04          NOP
FC0D1       TEST_12:
```

Figure 6.1.2. When executing a register transfer instruction starting from an odd address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
Jump address
Address from which to read data
BCLK
Address bus
FC05A
FC05E
FC060
FC062
04000
FC064
FC068
FC06A
Content at address 4001₁₆
Data bus (H)
F1
40
04
AA
04
F1
00
Content at address 4000₁₆
Data bus (L)
73
00
64
AA
04
73
40
RD
P
P
P
DR
P
P
P
WR

Fetch code	64					73F1	0040		64				

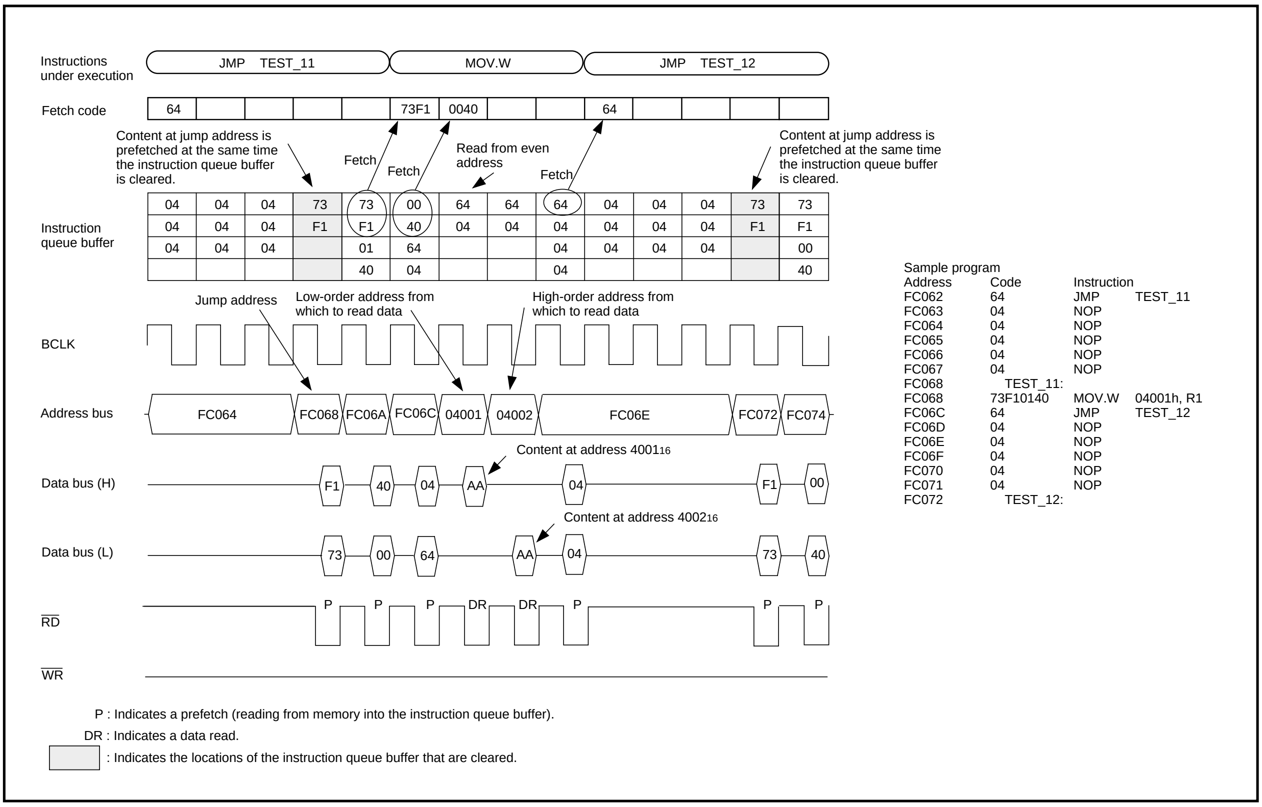
Instruction queue buffer													
04	04	04	73	73	00	64	64	04	04	04	73	73
04	04	04	F1	F1	40	04	04	04	04	04	F1	F1
04	04	04	▨	00	64		04	04	04	04	▨	00
			▨	40	04		04				▨	40

Sample program

Address	Code	Instruction	
FC058	64	JMP	TEST_11
FC059	04	NOP	
FC05A	04	NOP	
FC05B	04	NOP	
FC05C	04	NOP	
FC05D	04	NOP	
FC05E	TEST_11:		
FC05E	73F10040	MOV.W	04000h, R1
FC062	64	JMP	TEST_12
FC063	04	NOP	
FC064	04	NOP	
FC065	04	NOP	
FC066	04	NOP	
FC067	04	NOP	
FC068	TEST_12:		

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

Figure 6.1.3. When executing an instruction to read from even addresses starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Read from even address
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
Jump address
Low-order address from which to read data
High-order address from which to read data
BCLK
Address bus
FC064
FC068
FC06A
FC06C
04001
04002
FC06E
FC072
FC074
Content at address 4001₁₆
Data bus (H)
F1
40
04
AA
04
F1
00
Content at address 4002₁₆
Data bus (L)
73
00
64
AA
04
73
40
RD
P
P
P
DR
DR
P
P
P
WR

Fetch code	64					73F1	0040			64				

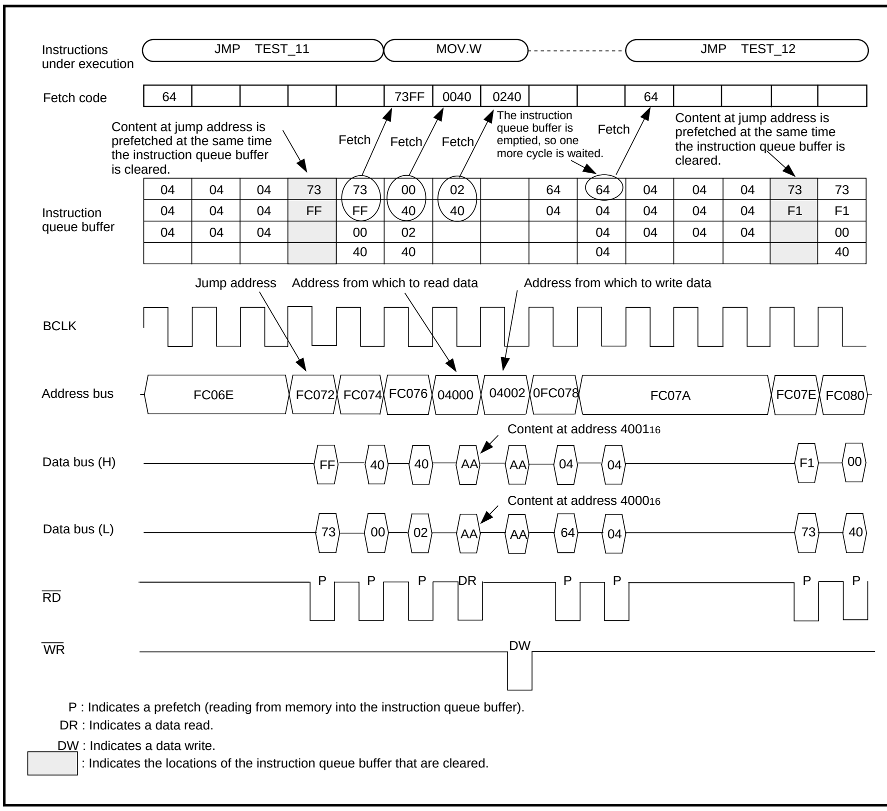
Instruction queue buffer														
04	04	04	73	73	00	64	64	64	04	04	04	73	73
04	04	04	F1	F1	40	04	04	04	04	04	04	F1	F1
04	04	04	▨	01	64			04	04	04	04	▨	00
			▨	40	04			04				▨	40

Sample program

Address	Code	Instruction	
FC062	64	JMP	TEST_11
FC063	04	NOP	
FC064	04	NOP	
FC065	04	NOP	
FC066	04	NOP	
FC067	04	NOP	
FC068	TEST_11:		
FC068	73F10140	MOV.W	04001h, R1
FC06C	64	JMP	TEST_12
FC06D	04	NOP	
FC06E	04	NOP	
FC06F	04	NOP	
FC070	04	NOP	
FC071	04	NOP	
FC072	TEST_12:		

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

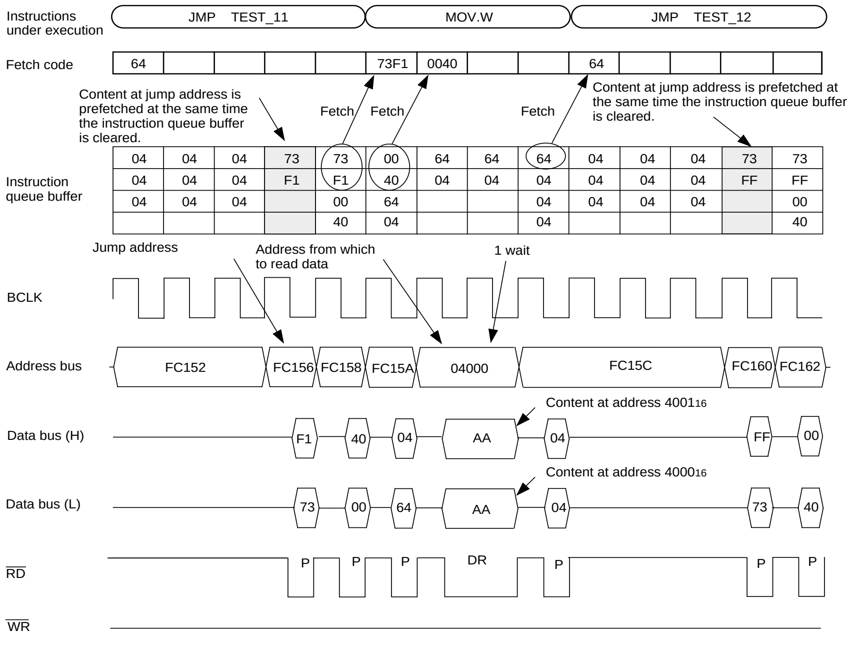
Figure 6.1.4. When executing an instruction to read from odd addresses starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
64
73FF
0040
0240
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Fetch
The instruction queue buffer is emptied, so one more cycle is waited.
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
04 04 04 73 73 00 02 64 64 04 04 04 73 73
04 04 04 FF FF 40 40 04 04 04 04 04 F1 F1
04 04 04 00 02 04 04 04 04 00
40 40 04 40
Jump address
Address from which to read data
Address from which to write data
BCLK
Address bus
FC06E
FC072
FC074
FC076
04000
04002
0FC078
FC07A
FC07E
FC080
Content at address 4001₁₆
Data bus (H)
FF 40 40 AA AA 04 04 F1 00
Content at address 4000₁₆
Data bus (L)
73 00 02 AA AA 64 04 73 40
RD
P P P DR P P P P
WR
DW

```
Sample program
Address   Code              Instruction
FC06C     64                JMP      TEST_11
FC06D     04                NOP
FC06E     04                NOP
FC06F     04                NOP
FC070     04                NOP
FC071     04                NOP
FC072        TEST_11:
FC072     73FF00400240      MOV.W    04000h, 04002h
FC078     64                JMP      TEST_12
FC079     04                NOP
FC07A     04                NOP
FC07B     04                NOP
FC07C     04                NOP
FC07D     04                NOP
FC07E        TEST_12:
```

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
DW : Indicates a data write.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

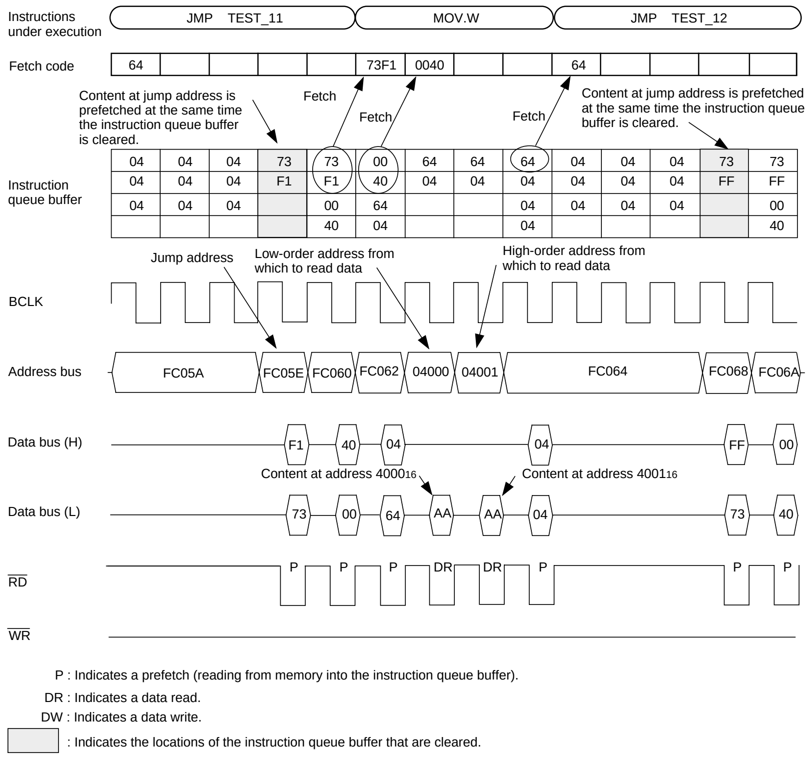
Figure 6.1.5. When executing an instruction to transfer data between even addresses starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
64
73F1
0040
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
04 04 04 73 73 00 64 64 64 04 04 04 73 73
04 04 04 F1 F1 40 04 04 04 04 04 04 FF FF
04 04 04 00 64 04 04 04 04 00
40 04 04 40
Jump address
Address from which to read data
1 wait
BCLK
Address bus
FC152
FC156
FC158
FC15A
04000
FC15C
FC160
FC162
Content at address 4001₁₆
Data bus (H)
F1 40 04 AA 04 FF 00
Content at address 4000₁₆
Data bus (L)
73 00 64 AA 04 73 40
RD
P P P DR P P P
WR

```
Sample program
Address   Code        Instruction
FC150     64          JMP      TEST_11
FC151     04          NOP
FC152     04          NOP
FC153     04          NOP
FC154     04          NOP
FC155     04          NOP
FC156        TEST_11:
FC156     73F10040    MOV.W    04000h, R1
FC15A     64          JMP      TEST_12
FC15B     04          NOP
FC15C     04          NOP
FC15D     04          NOP
FC15E     04          NOP
FC15F     04          NOP
FC160        TEST_12:
```

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
DW : Indicates a data write.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

Figure 6.1.6. When executing an instruction to read from even addresses starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus with wait state)

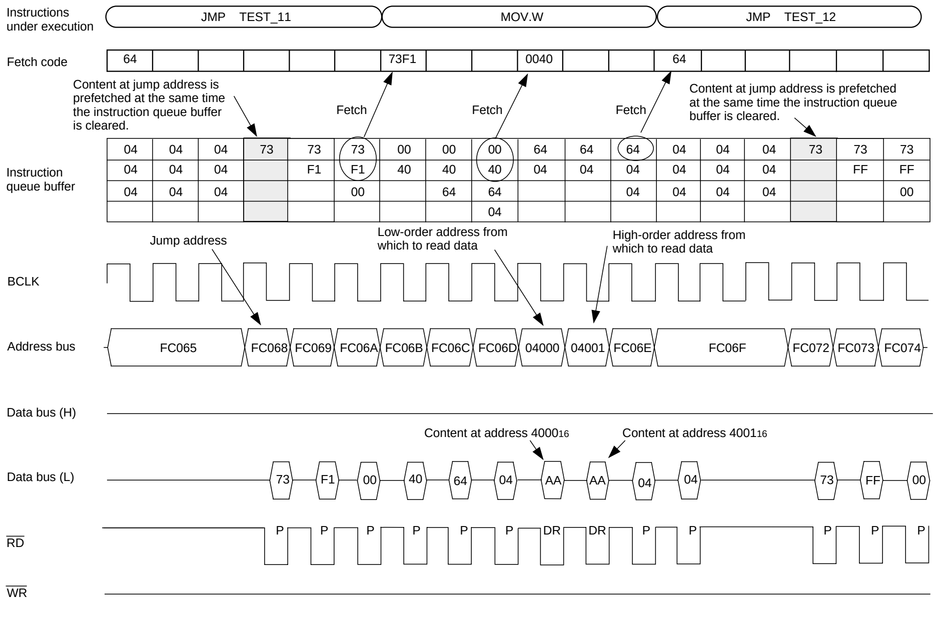

Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
64
73F1
0040
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
Jump address
Low-order address from which to read data
High-order address from which to read data
BCLK
Address bus
FC05A
FC05E
FC060
FC062
04000
04001
FC064
FC068
FC06A
Data bus (H)
F1
40
04
04
FF
00
Content at address 4000₁₆
Content at address 4001₁₆
Data bus (L)
73
00
64
AA
AA
04
73
40
RD
P
P
P
DR
DR
P
P
P
WR

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
DW : Indicates a data write.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

Sample program

Address	Code	Instruction
FC058	64	JMP TEST_11
FC059	04	NOP
FC05A	04	NOP
FC05B	04	NOP
FC05C	04	NOP
FC05D	04	NOP
FC05E	TEST_11:	
FC05E	73F10040	MOV.W 04000h, R1
FC062	64	JMP TEST_12
FC063	04	NOP
FC064	04	NOP
FC065	04	NOP
FC066	04	NOP
FC067	04	NOP
FC068	TEST_12:	

Figure 6.1.7. When executing a read instruction for memory connected to 8-bit bus (Program area: 16-bit bus without wait state; Data area: 8-bit bus without wait state)


Instructions under execution
JMP TEST_11
MOV.W
JMP TEST_12
Fetch code
64
73F1
0040
64
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Fetch
Fetch
Fetch
Content at jump address is prefetched at the same time the instruction queue buffer is cleared.
Instruction queue buffer
Jump address
Low-order address from which to read data
High-order address from which to read data
BCLK
Address bus
FC065
FC068
FC069
FC06A
FC06B
FC06C
FC06D
04000
04001
FC06E
FC06F
FC072
FC073
FC074
Data bus (H)
Content at address 4000₁₆
Content at address 4001₁₆
Data bus (L)
73
F1
00
40
64
04
AA
AA
04
04
73
FF
00
RD
P
P
P
P
P
P
DR
DR
P
P
P
P
P
WR

P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
DW : Indicates a data write.
▨ : Indicates the locations of the instruction queue buffer that are cleared.

Sample programs

Address	Code	Instruction
FC062	64	JMP TEST_11
FC063	04	NOP
FC064	04	NOP
FC065	04	NOP
FC066	04	NOP
FC067	04	NOP
FC068	TEST_11:	
FC068	73F10040	MOV.W 04000h, R1
FC06C	64	JMP TEST_12
FC06D	04	NOP
FC06E	04	NOP
FC06F	04	NOP
FC070	04	NOP
FC071	04	NOP
FC072	TEST_12:	

Figure 6.1.8. When executing a read instruction for memory connected to 8-bit bus (Program area: 8-bit bus without wait state; Data area: 8-bit bus without wait state)

Q & A

Information in a Q&A form to be used to make the most of the M16C family is given below.

Usually, one question and the answer to it are given on one page; the upper section is for the question, and the lower section is for the answer (if a pair of question and answer extends over two or more pages, a page number is given at the lower-right corner).

Functions closely connected with the contents of a page are shown at its upper-right corner.

Q

How do I distinguish between the static base register (SB) and the frame base register (FB)?

A

SB and FB function in the same manner, so you can use them as intended in programming in the assembly language. If you write a program in C, use FB as a stack frame base register.

Q

Is it possible to change the value of the interrupt table register (INTB) while a program is being executed?

A

Yes. But there can be a chance that the microcomputer runs away out of control if an interrupt request occurs in changing the value of INTB. So it is not recommended to frequently change the value of INTB while a program is being executed.

Q

What is the difference between the user stack pointer (USP) and the interrupt stack pointer (ISP)?, What are their roles?

A

You use USP when using the OS. When several tasks run, the OS secures stack areas to save registers of individual tasks. Also, stack areas have to be secured, task by task, to be used for handling interrupts that occur while tasks are being executed. If you use USP and ISP in such an instance, the stack for interrupts can be shared by these tasks; this allows you to efficiently use stack areas.

Q

How does the instruction code become if I use a bit instruction in absolute addressing ?

A

An explanation is given here by taking BSET bit,base:16 as an example.
This instruction is a 4-byte instruction. The 2 higher-order bytes of the instruction code indicate operation code, and the 2 lower-order bytes make up addressing mode to expresse bit,base:16.
The relation between the 2 lower-order bytes and bit,base:16 is as follows.
2 lower-order bytes = base:16 × 8 + bit

For example, in the case of BSET 2,0AH (setting bit 2 of address $000A_{16}$ to 1), the 2 lower-order bytes turn to A × 8 + 2 = 52H.
In the case of BSET 18,8H (setting the 18th bit from bit 0 of address 0008_{16} to 1), the 2 lower-order bytes turn to 8 × 8 + 18 = 52H, which is equivalent to BSET 2,AH.

The maximum value of base:16 × 8 + bit, FFFFH, indicates bit 7 of address $1FFF_{16}$. This is the maximum bit you can specify when using the bit instruction in absolute addressing.

Q

What is the difference between the DIV instruction and the DIVX instruction?

A

Either of the DIV instruction and the DIVX instruction is an instruction for signed division, the sign of the remainder is different.

The sign of the remainder left after the DIV instruction is the same as that of the dividend, on the contrary, the sign of the remainder of the DIVX instruction is the same as that of the divisor.

In general, the following relation among quotient, divisor, dividend, and remainder holds.

dividend = divisor × quotient + remainder

Since the sign of the remainder is different between these instructions, the quotient obtained either by dividing a positive integer by a negative integer or by dividing a negative integer by a positive integer using the DIV instruction is different from that obtained using the DIVX instruction.

For example, dividing 10 by –3 using the DIV instruction yields –3 and leaves +1, while doing the same using the DIVX instruction yields –4 and leaves –2.

Dividing –10 by +3 using the DIV instruction yields –3 and leaves –1, while doing the same using the DIVX instruction yields –4 and leaves +2.

Glossary

Technical terms used in this software manual are explained below. They are good in this manual only.

Term	Meaning	Related word
borrow	Tomove a digit to the next lower position.	carry
carry	Tomove a digit to the next higher position.	borrow
context	Registers that a program uses.	
decimal addition	An addition in terms of decimal system.	
displacement	The difference between the initial position and later position.	
effective address	An after-modification address to be actually used.	
extention area	For the M16C/60, M16C/20, M16C/Tiny series, the area from 10000_{16} through $FFFFF_{16}$.	
LSB	Abbreviation for Least Significant Biit The bit occupying the lowest-order position of a data item.	MSB

Term	Meaning	Related word
macro instruction	An instruction, written in a source language, to be expressed in a number of machine instructions when compiled into a machine code program.	
MSB	Abbreviation for Most Significant Bit The bit occupying the highest-order position of a data item.	LSB
operand	A part of instruction code that indicates the object on which an operation is performed.	operation code
operation	A generic term for move, comparison, bit processing, shift, rotation, arithmetic, logic, and branch.	
operation code	A part of instruction code that indicates what sort of operation the instruction performs.	operand
overflow	To exceed the maximum expressible value as a result of an operation.	
pack	To join data items. Used to mean to form two 4-bit data items into one 8-bit data item, to form two 8-bit data items into one 16-bit data item, etc.	unpack
SFR area	Abbreviation for Special Function Area. An area in which control bits of peripheral circuits embodied in a microcomputer and control registers are located.	

Term	Meaning	Related word
shift out	To move the content of a register either to the right or left until fully overflowed.	
sign bit	A bit that indicates either a positive or a negative (the highest-order bit).	
sign extension	To extend a data length in which the higher-order to be extended are made to have the same sign of the sign bit. For example, sign-extending FF_{16} results in $FFFF_{16}$, and sign-extending $0F_{16}$ results in $000F_{16}$.	
stack frame	An area for automatic variables the functions of the C language use.	
string	A sequence of characters.	
unpack	To restore combined items or packed information to the original form. Used to mean to separate 8-bit information into two parts — 4 lower-order bits and four higher-order bits, to separate 16-bit information into two parts — 8 lower-order bits and 8 higher-order bits, or the like.	pack
zero extension	To extend a data length by turning higher-order bits to 0's. For example, zero-extending FF_{16} to 16 bits results in $00FF_{16}$.	

Table of symbols

Symbols used in this software manual are explained below. They are good in this manual only.

Symbol	Meaning
←	Transposition from the right side to the left side
←→	Interchange between the right side and the left side
+	Addition
−	Subtraction
×	Multiplication
÷	Division
∧	Logical conjunction
∨	Logical disjunction
∀	Exclusive disjunction
‾	Logical negation
dsp16	16-bit displacement
dsp20	20-bit displacement
dsp8	8-bit displacement
EVA(　)	An effective address indicated by what is enclosed in (Å@)
EXT(　)	Sign extension
(H)	Higher-order byte of a register or memory
H4:	Four higher-order bits of an 8-bit register or 8-bit memory
| |	Absolute value
(L)	Lower-order byte of a register or memory
L4:	Four lower-order bits of an 8-bit register or 8-bit memory
LSB	Least Significant Bit
M(　)	Content of memory indicated by what is enclosed in (Å@)
(M)	Middle-order byte of a register or memory
MSB	Most Significant Bit
PCH	Higher-order byte of the program counter
PCML	Middle-order byte and lower-order byte of the program counter
FLGH	Four higher-order bits of the flag register
FLGL	Eight lower-order bits of the flag register

Index

A

A0 and A1 ••• **5**

A1A0 ••• **5**

Address register ••• **5**

Address space ••• **3**

Addressing mode ••• 22

B

B flag ••• **6**

Byte (8-bit) data ••• **16**

C

C flag ••• **6**

Carry flag ••• **6**

Cycles ••• 139

D

D flag ••• **6**

Data arrangement in memory ••• **17**

Data arrangement in Register ••• **16**

Data register ••• **4**

Data type ••• **10**

Debug flag ••• **6**

Description example ••• 37

dest ••• **18**

F

FB ••• **5**

Fixed vector table ••• **19**

Flag change ••• 37

Flag register ••• **5**

FLG ••• **5**

Frame base register ••• **5**

Function ••• 37

I

Interrupt table register ••• **5**

I flag ••• **6**

Instruction code ••• 139

Instruction Format ••• **18**

Instruction format specifier ••• 35

INTB ••• **5**

Integer ••• **10**

Interrupt enable flag ••• **6**

Interrupt stack pointer ••• **5**

Interrupt vector table ••• **19**

IPL ••• **7**

ISP ••• **5**

L

Long word (32-bit) data ••• **16**

M

Maskable interrupt ••• **248**

Memory bit ••• **12**

Mnemonic ••• 35, 38

N

Nibble (4-bit) data ••• **16**

Nonmaskable interrupt ••• **248**

O

O flag ••• **6**

Operand ••• 35, 38

Operation ••• 37

Overflow flag ••• **6**

P

PC ••• **5**

Processor interrupt priority level ••• **7**

Program counter ••• **5**

R

R0, R1, R2, and R3 ••• **4**

R0H, R1H ••• **4**

R0L, R1L ••• **4**

R2R0 ••• **4**

R3R1 ••• **4**

Register bank ••• **8**

Register bank select flag ••• **6**

Register bit ••• **12**

Related instruction ••• 37

Reset ••• **9**

S

S flag ••• **6**

SB ••• **5**

Selectable src / dest (label) ••• 37

Sign flag ••• **6**

Size specifier ••• 35

Software interrupt number ••• **20**

Special page number ••• **19**

Special page vector table ••• **19**

src ••• **18**

Stack pointer ••• **5**

Stack pointer select flag ••• **6**

Static base register ••• **5**

String ••• **15**

Syntax ••• 35, 38

U

U flag ••• **6**

User stack pointer ••• **5**

USP ••• **5**

V

Variable vector table ••• **20**

W

Word (16-bit) data ••• **16**

Z

Z flag ••• **6**

Zero flag ••• **6**

REVISION HISTORY	M16C/60, M16C/20, M16C/Tiny Series Software Manual

Rev.	Date	Description	
		Page	Summary
B	Sep 09, 1999	–	Page 104 [Operation] Line 3 Add to “ *1 When *dest* is SP or when the U flag = “0” and *dest* is ISP, the value 2 is not added to SP.” Page 108 [Operation] Line 3 Add to “ *1 When *src* is SP or when the U flag = “0” and *src* is ISP, the SP before being subtracted by 2 is saved. ” *Page 111 [Function] Line 5 Add to “ A0, A1 and R3 are indeterminate. ” *Page 125 [Function] Line 3 Add to “ However, the flag changes depending on the A0 or A1 status (16 bits) before the operation is performed. ” *Page 265 to 270 Add to “Chapter 6”
B1	Sep 21, 2000	–	Page 194 (1) LDINTB #IMM *1 #IMM1 indicates the 4 high-order bits of #IMM. #IMM2 indicates the 4 low-order bits of #IMM. ---> *1 #IMM1 indicates the 4 high-order bits of #IMM. #IMM2 indicates the 16 low-order bits of #IMM.
B2	Mar 07, 2001	–	Page 255 The DIVX instruction requires a maximum time that consists of 30 cycles (without wait state) ~~or 31 cycles (with one wait cycle)~~.
B3	Jul 09, 2002	–	Page 74 • If you selected (.B) for the size specifier (.size), R0 is sign extended to 32 bits. In this case, R2 is used for the upper bytes. ---> • If you selected (.W) for the size specifier (.size), R0 is sign extended to 32 bits. In this case, R2 is used for the upper bytes. Page 126 **[Function]** ---> **[Function]** • This instruction transfers *src* to *dest* when the Z flag is 0.

Rev.	Date	Description	
		Page	Summary
B3	Jul 09, 2002	–	Page 127 [Function] • This instruction transfers *src* to *dest* when the Z flag is 0. ---> • This instruction transfers *src* to *dest* when the Z flag is 1. Page 128 [Function] • This instruction transfers *src* to *dest* when the Z flag is 1. ---> • This instruction transfers *src1* to *dest* when the Z flag is 1. When the Z flag is 0, it transfers *src2* to *dest*. Page 129 [Function] • This instruction transfers *src1* to *dest* when the Z flag is 1. When the Z flag is 0, it transfers *src2* to *dest*. • This instruction subtracts *src* from *dest* and stores the result in *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to ---> • This instruction subtracts *src* from *dest* and stores the result in *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1. Page 131 [Function] perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1. • Each flag in the flag register changes state depending on the result of logical AND of *src* and *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to ---> • Each flag in the flag register changes state depending on the result of logical AND of *src* and *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), *src* is zero-expanded to perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1.

Rev.	Date	Description	
		Page	Summary
B3	Jul 09, 2002	–	Page 132 [Function] perform operation in 16 bits. If *src* is an A0 or A1, operation is performed on the 8 low-order bits of A0 or A1. ---> • This instruction generates an undefined instruction interrupt. • The undefined instruction interrupt is a nonmaskable interrupt. Page 133 [Function] • This instruction generates an undefined instruction interrupt. ---> • This instruction halts program execution. Program execution is restarted when an interrupt of a higher priority level than IPL is acknowledged or a reset is generated. Page 134 [Function] • The undefined instruction interrupt is a nonmaskable interrupt. • This instruction halts program execution. Program execution is restarted when an interrupt of a higher priority level than IPL is acknowledged or a reset is generated. ---> • This instruction exchanges contents between *src* and *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), 16 bits of zero- expanded *src* data are placed in the A0 or A1 and the 8 low-order bits of the A0 or A1 are placed in *src*. Page 135 [Function] • This instruction exchanges contents between *src* and *dest*. • If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), 16 bits of zero- expanded *src* data are placed in the A0 or A1 and the 8 low-order bits of the A0 or A1 are placed in *src*. ---> • ~~This instruction exchanges contents between *src* and *dest*.~~ • ~~If *dest* is an A0 or A1 when the size specifier (.size) you selected is (.B), 16 bits of zero- expanded *src* data are placed in the A0 or A1 and the 8 low-order bits of the A0 or A1 are placed in *src*.~~

REVISION HISTORY	M16C/60, M16C/20, M16C/Tiny Series Software Manual

Rev.	Date	Description	
		Page	Summary
4.00	Jan 21, 2004	253, 262	Add TECHNICAL NEWS NO M16C-85-0204 “M16C Family Usage Precautions when Clearing Interrupt Request Bit “

RENESAS 16-BIT SINGLE-CHIP MICROCOMPUTER
SOFTWARE MANUAL
M16C/60, M16C/20, M16C/Tiny Series

Publication Data : Rev.B3 Jul 15, 2002
Rev.4.00 Jan 21, 2004
Published by : Sales Strategic Planning Div.
Renesas Technology Corp.

© 2003, 2004. Renesas Technology Corp., All rights reserved. Printed in Japan.

M16C/60, M16C/20, M16C/Tiny Series
Software Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan