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/
32

M32C/80 Series

Software Manual

RENESAS 16/32-BIT SINGLE-CHIP MICROCOMPUTER
M16C FAMILY / M32C/80 SERIES

Renesas Electronics
www.renesas.com

Rev.1.00 2006.03

Keep safety first in your circuit designs!

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

Notes regarding these materials

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

Using This Manual

This manual is written for the M32C/80 Series software. This manual can be used for all types of MCUs having the M32C/80 Series CPU core.
The reader of this manual is expected to have the basic knowledge of electric and logic circuits and MCUs.
This manual consists of seven 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 M32C/80 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”**
- To understand interrupt ..... **Chapter 5, "Interrupt"**
- To understand calculation number of cycles **Chapter 6, "Caluculation Number of Cycles"**
- To refer to precautions ..... **Chapter 7, "Precautions"**

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
Shortsheet	Hardware overview
Datasheet	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 MCU 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.

Note:

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

Table of Contents

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	43	174	DADD	74	208
ADC	44	174	DEC	75	210
ADCF	45	176	DIV	76	210
ADD	46	176	DIVU	77	212
ADDX	48	183	DIVX	78	213
ADJNZ	49	185	DSBB	79	215
AND	50	186	DSUB	80	217
BAND	52	188	ENTER	81	219
BCLR	53	188	EXITD	82	219
BITINDEX	54	189	EXTS	83	220
BM*Cnd*	55	190	EXTZ	84	222
BMEQ/Z	55	190	FCLR	85	223
BMGE	55	190	FREIT	86	223
BMGEU/C	55	190	FSET	87	224
BMGT	55	190	INC	88	225
BMGTU	55	190	INDEXB	89	225
BMLE	55	190	INDEXBD	89	226
BMLEU	55	190	INDEXBS	89	226
BMLT	55	190	INDEXL	89	227
BMLTU/NC	55	190	INDEXLD	89	227
BMN	55	190	INDEXLS	89	228
BMNE/NZ	55	190	INDEXW	89	228
BMNO	55	190	INDEXWD	89	229
BMO	55	190	INDEXWS	89	229
BMPZ	55	190	INT	90	230
BNAND	56	192	INTO	91	230
BNOR	57	192	J*Cnd*	92	231
BNOT	58	193	JEQ/Z	92	231
BNTST	59	193	JGE	92	231
BNXOR	60	194	JGEU/C	92	231
BOR	61	194	JGT	92	231
BRK	62	195	JGTU	92	231
BRK2	63	195	JLE	92	231
BSET	64	196	JLEU	92	231
BTST	65	196	JLT	92	231
BTSTC	66	197	JLTU/NC	92	231
BTSTS	67	198	JN	92	231
BXOR	68	198	JNE/NZ	92	231
CLIP	69	199	JNO	92	231
CMP	70	200	JO	92	231
CMPX	72	206	JPZ	92	231
DADC	73	206			

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
JMP	93	231	SBB	130	277
JMPI	94	233	SBJNZ	131	279
JMPS	95	234	SC*cnd*	132	280
JSR	96	235	SCEQ/Z	132	280
JSRI	97	236	SCGE	132	280
JSRS	98	237	SCGEU/C	132	280
LDC	99	237	SCGT	132	280
LDCTX	100	240	SCGTU	132	280
LDIPL	101	241	SCLE	132	280
MAX	102	241	SCLEU	132	280
MIN	103	243	SCLT	132	280
MOV	104	245	SCLTU/NC	132	280
MOVA	106	254	SCN	132	280
MOV*Dir*	107	255	SCNE/NZ	132	280
MOVHH	107	255	SCNO	132	280
MOVHL	107	255	SCPZ	132	280
MOVLH	107	255	SCMPU	133	281
MOVLL	107	255	SHA	134	282
MOVX	108	257	SHANC	135	284
MUL	109	257	SHL	136	285
MULEX	110	260	SHLNC	137	288
MULU	111	260	SIN	138	288
NEG	112	263	SMOVB	139	289
NOP	113	263	SMOVF	140	289
NOT	114	264	SMOVU	141	290
OR	115	264	SOUT	142	290
POP	117	267	SSTR	143	291
POPC	118	267	STC	144	291
POPM	119	268	STCTX	145	293
PUSH	120	269	STNZ	146	293
PUSHA	121	271	STZ	147	294
PUSHC	122	271	STZX	148	294
PUSHM	123	272	SUB	149	295
REIT	124	273	SUBX	151	299
RMPA	125	273	TST	152	301
ROLC	126	274	UND	154	303
RORC	127	274	WAIT	155	303
ROT	128	275	XCHG	156	304
RTS	129	276	XOR	157	304

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code/ number of cycles
Transfer	MOV	Transfer	104	245
	MOVA	Transfer effective address	106	254
	MOVDir	Transfer 4-bit data	107	255
	MOVX	Transfer extend sign	108	257
	POP	Restore register/memory	117	267
	POPC	Restore control register	118	267
	POPM	Restore multiple registers	119	268
	PUSH	Save register/memory/immediate data	120	269
	PUSHA	Save effective address	121	271
	PUSHM	Save multiple registers	123	272
	STNZ	Conditional transfer	146	293
	STZ	Conditional transfer	147	294
	STZX	Conditional transfer	148	294
	XCHG	Exchange	156	304
Bit manupulation	BAND	Logically AND bits	52	188
	BCLR	Clear bit	53	188
	BITINDEX	Bit index	54	189
	BM*Cnd*	Conditional bit transfer	55	190
	BNAND	Logically AND inverted bits	56	192
	BNOR	Logically OR inverted bits	57	192
	BNOT	Invert bit	58	193
	BNTST	Test inverted bit	59	193
	BNXOR	Exclusive OR inverted bits	60	194
	BOR	Logically OR bits	61	194
	BSET	Set bit	64	196
	BTST	Test bit	65	196
	BTSTC	Test bit & clear	66	197
	BTSTS	Test bit & set	67	198
	BXOR	Exclusive OR bits	68	198
Shift	ROLC	Rotate left with carry	126	274
	RORC	Rotate right with carry	127	274
	ROT	Rotate	128	275
	SHA	Shift arithmetic	134	282
	SHANC	Shift arithmetic	135	284
	SHL	Shift logical	136	285
	SHLNC	Shift logical	137	288
Arithmetic	ABS	Absolute value	43	174
	ADC	Add with carry	44	174
	ADCF	Add carry flag	45	176
	ADD	Add without carry	46	176
	ADDX	Add extend sigh without carry	48	183

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code/ number of cycles
Arithmetic	CLIP	Clip	69	199
	CMP	Compare	70	200
	CPMX	Compare extended sigh	72	206
	DADC	Decimal add with carry	73	206
	DADD	Decimal add without carry	74	208
	DEC	Decrement	75	210
	DIV	Signed divide	76	210
	DIVU	Unsigned divide	77	212
	DIVX	Singed divide	78	213
	DSBB	Decimal subtract with borrow	79	215
	DSUB	Decimal subtract without borrow	80	217
	EXTS	Extend sign	83	220
	EXTZ	Extend zero	84	222
	INC	Increment	88	225
	MAX	Select maximum value	102	241
	MIN	Select minimum value	103	243
	MUL	Signed multiply	109	257
	MULEX	Multiple extend sign	110	260
	MULU	Unsigned multiply	111	260
	NEG	Two’s complement	112	263
	RMPA	Calculate sum-of-products	125	273
	SBB	Subtract with borrow	130	277
	SUB	Subtract without borrow	149	295
	SUBX	Subtract extend without borrow	151	299
Logical	AND	Logical AND	50	186
	NOT	Invert all bits	114	264
	OR	Logical OR	115	264
	TST	Test	152	301
	XOR	Exclusive OR	157	304
Jump	ADJNZ	Add & conditional jump	49	185
	SBJNZ	Subtract & conditional jump	131	279
	JCnd	Jump on condition	92	231
	JMP	Unconditional jump	93	231
	JMPI	Jump indirect	94	233
	JMPS	Jump to special page	95	234
	JSR	Subroutine call	96	235
	JSRI	Indirect subroutine call	97	236
	JSRS	Special page subroutine call	98	237
	RTS	Return from subroutine	129	276
String	SCMPU	String compare unequal	133	281
	SIN	String input	138	288

Quick Reference by Function

Function	Mnemonic	Content	See page for function	See page for instruction code/ number of cycles
String	SMOVB	Transfer string backward	139	289
	SMOVF	Transfer string forward	140	289
	SMOVU	Transfer string	141	290
	SOUT	String output	142	290
	SSTR	Store string	143	291
Other	BRK	Debug interrupt	62	195
	BRK2	Debug interrupt 2	63	195
	ENTER	Build stack frame	81	219
	EXITD	Deallocate stack frame	82	219
	FCLR	Clear flag register bit	85	223
	FREIT	Fast return from interrupt	86	223
	FSET	Set flag register bit	87	224
	INDEX Type	Index	89	225
	INT	Interrupt by INT instruction	90	230
	INTO	Interrupt on overflow	91	230
	LDC	Transfer to control register	99	237
	LDCTX	Restore context	100	240
	LDIPL	Set interrupt enable level	101	241
	NOP	No operation	113	263
	POPC	Restore control register	118	267
	PUSHC	Save control register	122	271
	REIT	Return from interrupt	124	273
	STC	Transfer from control register	144	291
	STCTX	Save context	145	293
	SC*cnd*	Store on condition	132	280
	UND	Interrupt for undefined instruction	154	303
	WAIT	Wait	155	303

Quick Reference by Addressing (general instruction addressing)

Mnemonic	Addressing																									See page for function	See page for instruction code /number of cycles
	R0L/R0/R2R0	R0H/R2/-	R1L/R1/R3R1	R1H/R3/-	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24	#IMM8	#IMM16	#IMM24	#IMM32	#IMM	[[An]]	[dsp:8[An]]	[dsp:8[SB/FB]]	[dsp:16[An]]	[dsp:16[SB/FB]]	[dsp:24[An]]	[abs16]	[abs24]		
ABS	√*2	√	√*3	√	√	√	√	√	√	√	√								√	√	√	√	√	√	√	√	43	174
ADC	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√												44	174
ADCF	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	45	176
ADD*1	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√		√	√	√	√	√	√	√	√	√	√	46	176
ADDX	√*2	√*4	√*3	√*5	√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	48	183
ADJNZ*1	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√					√									49	185
AND	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	50	186
BITINDEX	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√														54	189
CLIP	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√												69	199
CMP	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√		√	√	√	√	√	√	√	√	√	√	70	200
CMPX	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	72	206
DADC	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√												73	206
DADD	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√												74	208
DEC	√	√	√	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	75	210
DIV	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	76	210
DIVU	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	77	212
DIVX	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	78	213
DSBB	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	79	215
DSUB	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	80	217
ENTER																√											81	219
EXTS	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√														83	220
EXTZ	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√														84	222
INC	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	88	225
INDEXType	√*2	√	√*3	√	√	√	√	√	√	√	√	√	√														89	225

*1 Has special instruction addressing.

*2 Only R0L/R0 can be selected.

*3 Only R1L/R1 can be selected.

*4 Only R0L can be selected.

*5 Only R0H can be selected.

*6 Only R1L can be selected.

*7 Only R1H 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/R2R0	R0H/R2/-	R1L/R1/R3R1	R1H/R3/-	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24	#IMM8	#IMM16	#IMM24	#IMM32	#IMM	[[An]]	[dsp:8[An]]	[dsp:8[SB/FB]]	[dsp:16[An]]	[dsp:16[SB/FB]]	[dsp:24[An]]	[abs16]	[abs24]		
INT																		√									90	230
JMP*1													√														93	231
JMPI*1	√*2	√*3	√*4	√*5	√	√	√	√	√	√	√	√	√														94	233
JMPS																√											95	234
JSRI	√*2	√*3	√*4	√*5	√	√	√	√	√	√	√	√	√														97	236
JSRS														√													98	237
LDC*1	√*2	√*3	√*4	√*5	√	√	√	√	√	√	√	√	√		√	√											99	237
LDIPL																		√									101	241
MAX	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√	√	√												102	241
MIN	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√	√	√												103	243
MOV*1	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	104	245
MOVA	√*8		√*9		√		√	√	√	√	√	√	√														106	254
MOVDir	√*10	√*11	√*12	√*13		√	√	√	√	√	√	√	√														107	255
MOVX	√*8		√*9		√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	108	257
MUL	√	√	√		√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	109	257
MULEX				√*5	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	110	260
MULU	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	111	260
NEG	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	112	263
NOT	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	114	264
OR	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	115	264
POP	√*6	√	√*7	√	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	117	267
POPM*1	√	√	√	√																							119	268
PUSH	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√		√		√	√	√	√	√	√	√	√	120	269
PUSHA							√	√	√	√	√	√	√														121	271

*1 Has special instruction addressing.
*2 Only R0/R2R0 can be selected.
*3 Only R2 can be selected.
*4 Only R1/R3R1 can be selected.
*5 Only R3 can be selected.
*6 Only R0L/R0 can be selected.
*7 Only R1L/R1 can be selected.
*8 Only R2R0 can be selected.
*9 Only R3R1 can be selected.
*10 Only R0L can be selected.
*11 Only R0H can be selected.
*12 Only R1L can be selected.
*13 Only R1H 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/R2R0	R0H/R2/-	R1L/R1/R3R1	R1H/R3/-	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24	#IMM8	#IMM16	#IMM24	#IMM32	#IMM	[[An]]	[dsp:8[An]]	[dsp:8[SB/FB]]	[dsp:16[An]]	[dsp:16[SB/FB]]	[dsp:24[An]]	[abs16]	[abs24]		
PUSHM*1	√	√	√	√																							123	272
ROLC	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	126	274
RORC	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	127	274
ROT	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	√	128	275
SBB	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√												130	277
SBJNZ*1	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√					√									131	279
SCCnd	√*4	√*5	√*6	√*7	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	132	280
SHA	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	√	134	282
SHANC	√*12		√*13		√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	135	284
SHL	√	√	√	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	√	136	285
SHLNC	√*12		√*13		√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	137	288
STC*1	√*4	√*5	√*6	√*7	√	√	√	√	√	√	√	√	√														144	291
STCTX*1	√	√	√	√																							145	293
STNZ	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	146	293
STZ	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	147	294
STZX	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	148	294
SUB	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√		√		√	√	√	√	√	√	√	√	149	295
SUBX	√*8	√*9	√*10	√*11	√	√	√	√	√	√	√	√	√	√					√	√	√	√	√	√	√	√	151	299
TST	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√												152	301
XCHG	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√						√	√	√	√	√	√	√	√	156	304
XOR	√*2	√	√	√*3	√	√	√	√	√	√	√	√	√	√	√				√	√	√	√	√	√	√	√	157	304

*1 Has special instruction addressing.
*2 Only R0L/R0 can be selected.
*3 Only R1L/R1 can be selected.
*4 Only R0 can be selected.
*5 Only R2 can be selected.
*6 Only R1 can be selected.
*7 Only R3 can be selected.
*8 Only R0L/R2R0 can be selected.
*9 Only R0H can be selected.
*10 Only R1L/R3R1 can be selected.
*11 Only R1H can be selected.
*12 Only R2R0 can be selected.
*13 Only R3R1 can be selected.

Quick Reference by Addressing (special instruction addressing)

Mnemonic	Addressing													See page for function	See page for instruction code /number of cycles
	label	SB/FB	ISP/USP	FLG	INTB	SVP/VCT	SVF	DMD0/DMD1	DCT0/DCT1	DRC0/DRC1	DMA0/DMA1	DRA0/DRA1	DSA0/DSA1		
ADD*1			√											46	176
ADJNZ*1	√													49	185
J*Cnd*	√													92	231
JMP*1	√													93	231
JSR*1	√													96	235
LDC*1		√	√	√	√	√	√	√	√	√	√	√	√	99	237
POPC		√	√	√	√	√	√	√	√					118	267
POPM*1		√												119	268
PUSHC		√	√	√	√	√	√	√	√					122	271
PUSHM*1		√												123	272
SBJNZ*1	√													131	279
STC*1		√	√	√	√	√	√	√	√	√	√	√	√	144	291

*1 Has general instruction addressing.

Quick Reference by Addressing (bit instruction addressing)

Mnemonic	Addressing												See page for function	See page for instruction code /number of cycles
	bit,R0L/R0H	bit,R1L/R1H	bit,An	bit,[An]	bit,base:11[An]	bit,base:11[SB/FB]	bit,base:19[An]	bit,base:19[SB/FB]	bit,base:27[An]	bit,base:27	bit,base:19	U/I/O/B/S/Z/D/C		
BAND	√	√	√	√	√	√	√	√	√	√	√		52	188
BCLR	√	√	√	√	√	√	√	√	√	√	√		53	188
BM*Cnd*	√	√	√	√	√	√	√	√	√	√	√	√	55	190
BNAND	√	√	√	√	√	√	√	√	√	√	√		56	192
BNOR	√	√	√	√	√	√	√	√	√	√	√		57	192
BNOT	√	√	√	√	√	√	√	√	√	√	√		58	193
BNTST	√	√	√	√	√	√	√	√	√	√	√		59	193
BNXOR	√	√	√	√	√	√	√	√	√	√	√		60	194
BOR	√	√	√	√	√	√	√	√	√	√	√		61	194
BSET	√	√	√	√	√	√	√	√	√	√	√		64	196
BTST	√	√	√	√	√	√	√	√	√	√	√		65	196
BTSTC	√	√	√	√	√	√	√	√	√	√	√		66	197
BTSTS	√	√	√	√	√	√	√	√	√	√	√		67	198
BXOR	√	√	√	√	√	√	√	√	√	√	√		68	198
FCLR												√	85	223
FSET												√	87	224

Chapter 1

Overview

1.1 Features of M32C/80 Series

The M32C/80 Series is a single-chip MCU developed for built-in applications where the MCU is built into applications equipment.
The M32C/80 Series supports 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 one clock cycle, making fast arithmetic processing possible.
Its instruction set consists of 108 discrete instructions matched to the M32C'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.
M32C/80 Series models incorporate a multiplier, allowing for high-speed computation.

■ Features of M32C/80 Series

• Register configuration

Data registers : Four 16-bit registers (of which two registers can be used as 8-bit registers, or two registers are combined and can be used as 32-bit registers)
Address registers : Two 24-bit registers
Base registers : Two 24-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.

• 16M-byte linear address area

Relative jump instructions matched to distance of jump

• Fast instruction execution time

Shortest 1-cycle instructions : 108 instructions include 39 1-cycle instructions.

■ Speed performance (types incorporating a multiplier, operating at 32 MHz)

	Cycle	Execution Time
Register-register transfer	1	31.3 ns
Register-memory transfer	1	31.3 ns
Register-register addition/subtraction	1	31.3 ns
8 bits x 8 bits register-register operation	3	93.8 ns
16 bits x 16 bits register-register operation	3	93.8 ns
16 bits / 8 bits register-register operation	18	562.5 ns
32 bits / 16 bits register-register operation	18	562.5 ns

1.2 Address Space

Fig. 1.2.1 shows an address space.

Addresses 000000$_{16}$ through 0003FF$_{16}$ make up an SFR (special function register) area. In individual models of the M32C/80 Series, the SFR area extends from 0003FF$_{16}$ toward lower addresses.

Addresses from 000400$_{16}$ on make up a memory area. In individual models of the M32C/80 Series, a RAM area extends from address 000400$_{16}$ toward higher addresses, and a ROM area extends from FFFFFF$_{16}$ toward lower addresses. Addresses FFFE00$_{16}$ through FFFFFF$_{16}$ make up a fixed vector area.

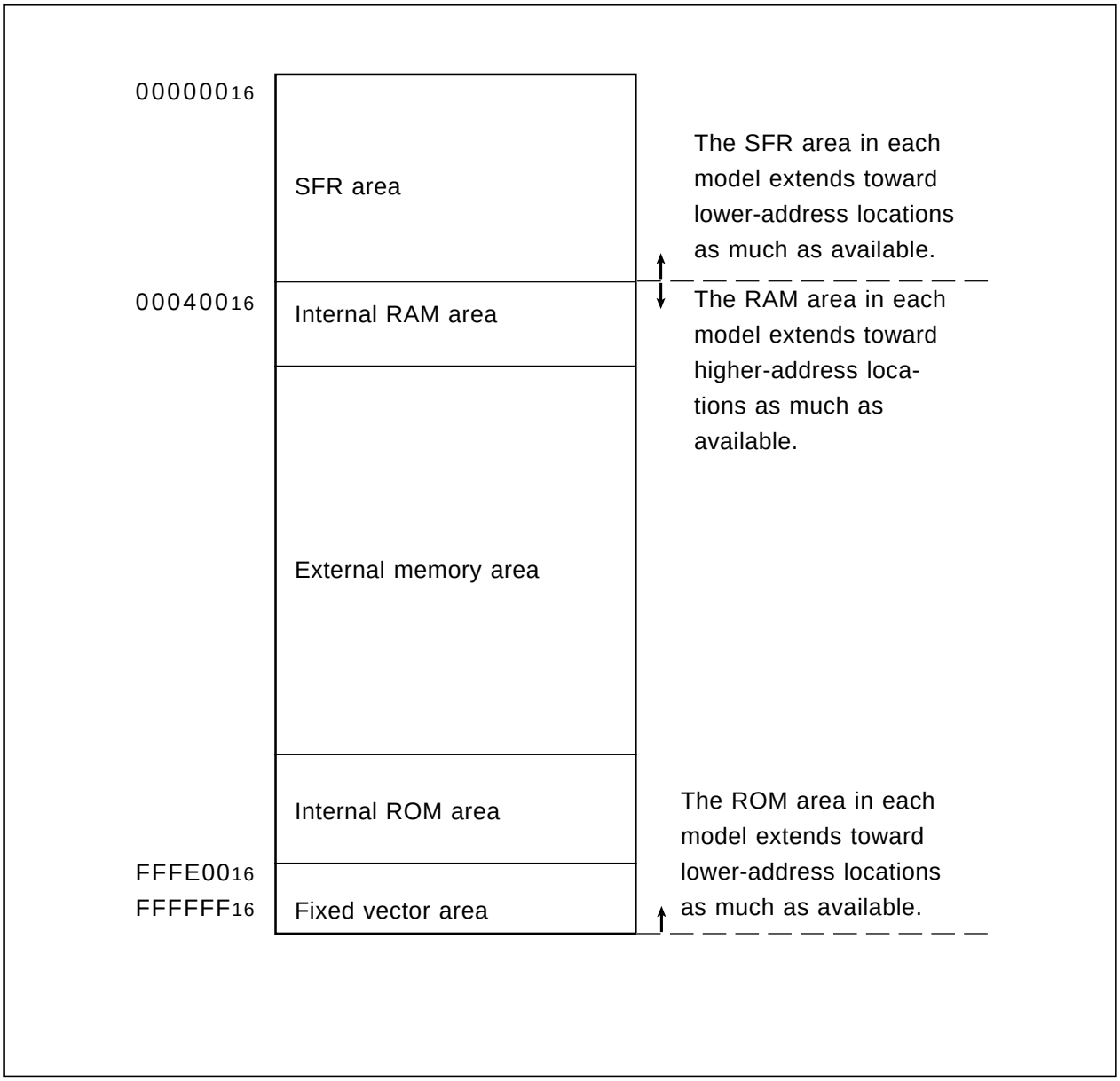


Figure 1.2.1 Address area

1.3 Register Configuration

Figure 1.3.1 shows the CPU registers. The register bank is comprised of eight registers (R0, R1, R2, R3, A0, A1, FB, and SB) out of 28 CPU registers. There are two sets of register banks.

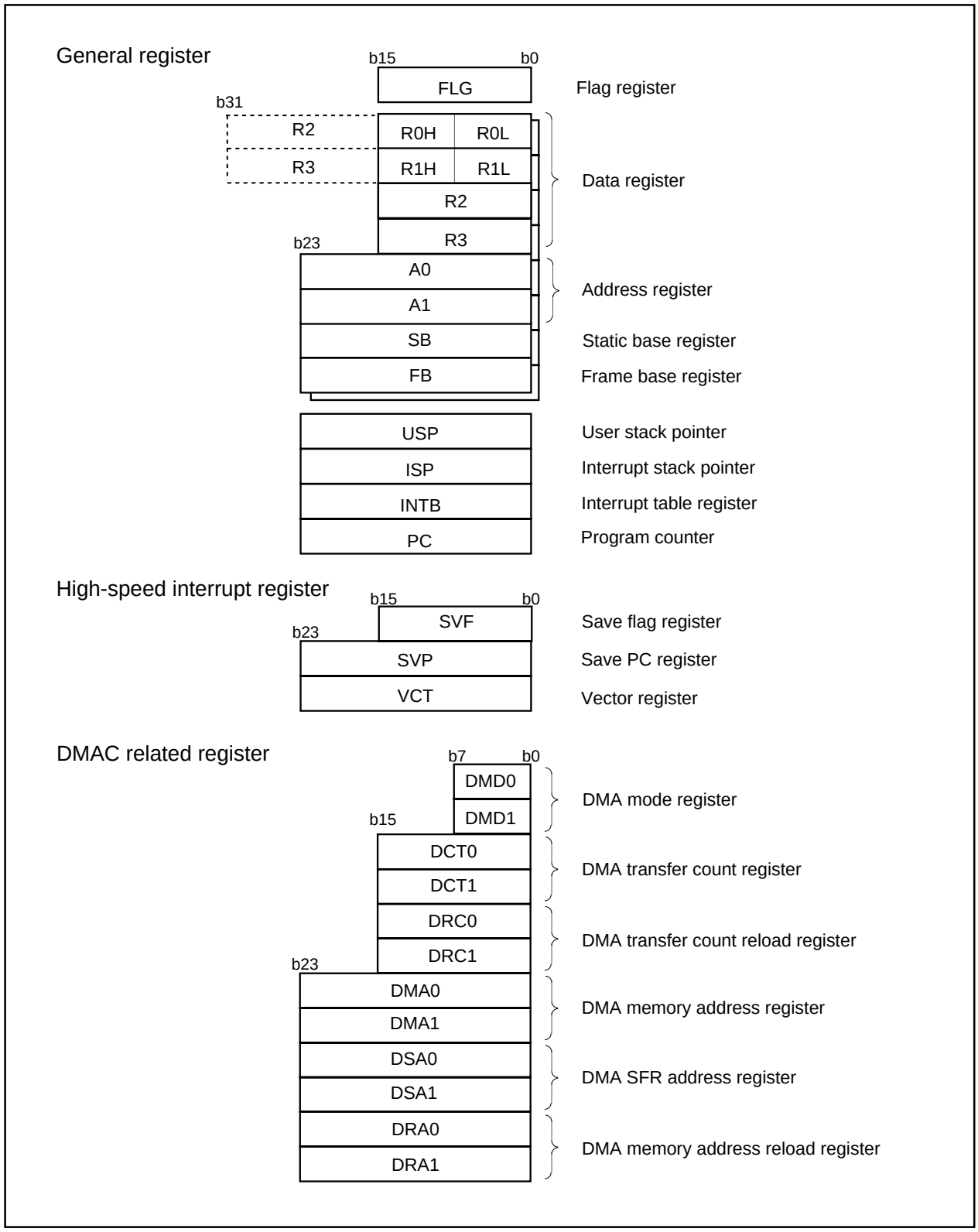


Figure 1.3.1 CPU register configuration

(1) Data Registers (R0, R0H, R0L, R1, R1H, R1L, R2, R3, R2R0, and R3R1)

These registers 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. Moreover, you can combine R2 and R0 or R3 and R1 to configure a 32-bit data register (R2R0 or R3R1).

(2) Address Registers (A0 and A1)

These registers consist of 24 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.

(3) Static Base Register (SB)

This register consists of 24 bits, and is used for SB-based relative addressing.

(4) Frame Base Register (FB)

This register consists of 24 bits, and is used for FB-based relative addressing.

(5) Program Counter (PC)

This counter consists of 24 bits, indicating the address of an instruction to be executed next.

(6) Interrupt Table Register (INTB)

This register consists of 24 bits, indicating the initial address of an interrupt vector table.

(7) 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 24 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 in the flag register (FLG).

Set an even number to USP and ISP. When an even number is set, execution becomes efficient.

(8) Flag Register (FLG)

This register 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)**."

(9) Save Flag Register (SVF)

This register consists of 16 bits and is used to save the flag register when a high-speed interrupt is generated.

(10) Save PC Register (SVP)

This register consists of 16 bits and is used to save the program counter when a high-speed interrupt is generated.

(11) Vector Register (VCT)

This register consists of 24 bits and is used to indicate the jump address when a high-speed interrupt is generated.

(12) DMA Mode Registers (DMD0/DMD1)

These registers consist of 8 bits and are used to set transfer mode, etc. for DMA.

(13) DMA Transfer Count Registers (DCT0/DCT1)

These registers consist of 16 bits and are used to set the number of DMA transfers to be performed.

(14) DMA Transfer Count Reload Registers (DRC0/DRC1)

These registers consist of 16 bits and are used to reload the DMA transfer count registers.

(15) DMA Memory Address Registers (DMA0/DMA1)

These registers consist of 24 bits and are used to set a memory address at the source or destination of DMA transfer.

(16) DMA SFR Address Registers (DSA0/DSA1)

These registers consist of 24 bits and are used to set a fixed address at the source or destination of DMA transfer.

(17) DMA Memory Address Reload Registers (DRA0/DRA1)

These registers consist of 24 bits and are used to reload the DMA memory address registers.

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) Bit 0: Carry Flag (C flag)

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

(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.

(3) Bit 2: Zero Flag (Z flag)

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

(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.

(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; when the flag is 1, register bank 1 is selected.

(6) Bit 5: Overflow Flag (O flag)

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

(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.

(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 the INT instruction of software interrupt numbers 0 to 31 is executed.

(9) Bits 8-11: Reserved Area

(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.

(11) Bit 15: Reserved Area

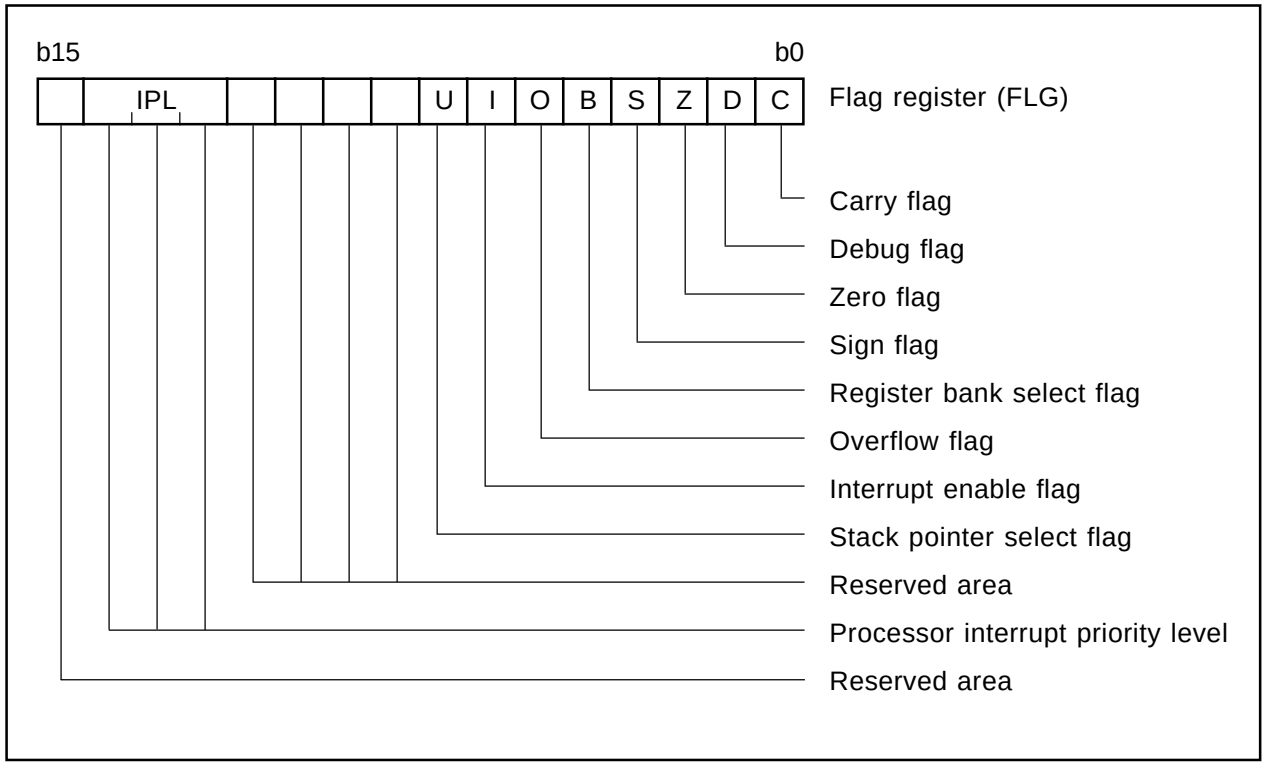
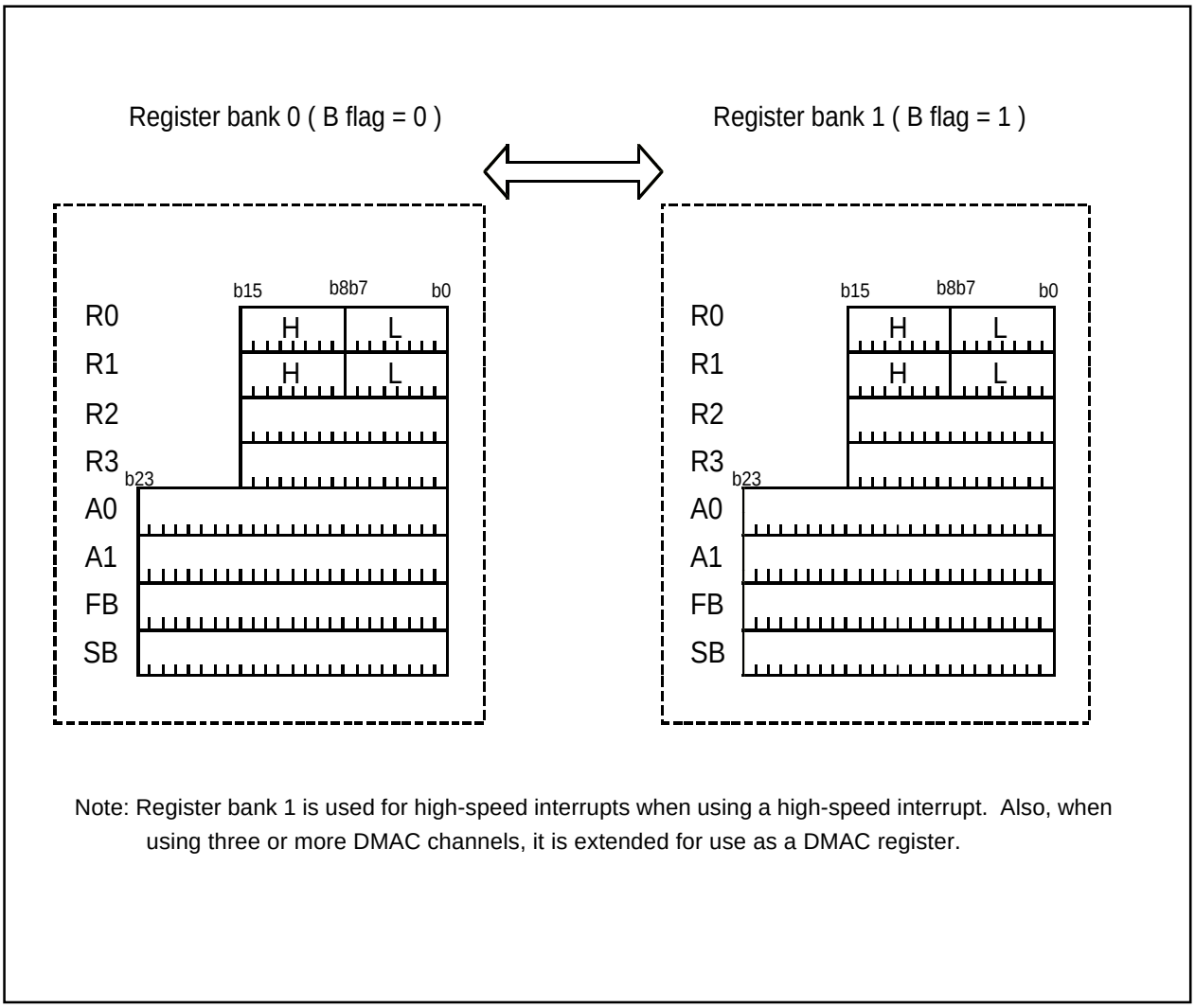


Figure 1.4.1 Configuration of flag register (FLG)

1.5 Register Bank

The M32C/80 has two register banks, each configured with data registers (R0, R1, R2, and R3), address registers (A0 and A1), frame base register (FB), and static base register (SB). These two register banks are switched over by the register bank select flag (B flag) in the flag register (FLG).
Figure 1.5.1 shows a configuration of register banks.



Note: Register bank 1 is used for high-speed interrupts when using a high-speed interrupt. Also, when using three or more DMAC channels, it is extended for use as a DMAC register.

Figure 1.5.1 Configuration of register banks

1.6 Internal State after Reset

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

- Data registers (R0, R1, R2, and R3) : 0000_{16}
- Address registers (A0 and A1) : 00000000_{16}
- Static base register (SB) : 00000000_{16}
- Frame base register (FB) : 00000000_{16}
- Interrupt table register (INTB) : 00000000_{16}
- User stack pointer (USP) : 00000000_{16}
- Interrupt stack pointer (ISP) : 00000000_{16}
- Flag register (FLG) : 0000_{16}
- DMA mode register (DMD0/DMD1) : 00_{16}
- DMA transfer count register (DCT0/DCT1) : undefined
- DMA transfer count reload register (DRC0/DRC1) : undefined
- DMA memory address register (DMA0/DMA1) : undefined
- DMA SFR address register (DSA0/DSA1) : undefined
- DMA memory address reload register (DRA0/DRA1) : undefined
- Save flag register (SVF) : undefined
- Save PC register (SVP) : undefined
- Vector register (VCT) : undefined

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.

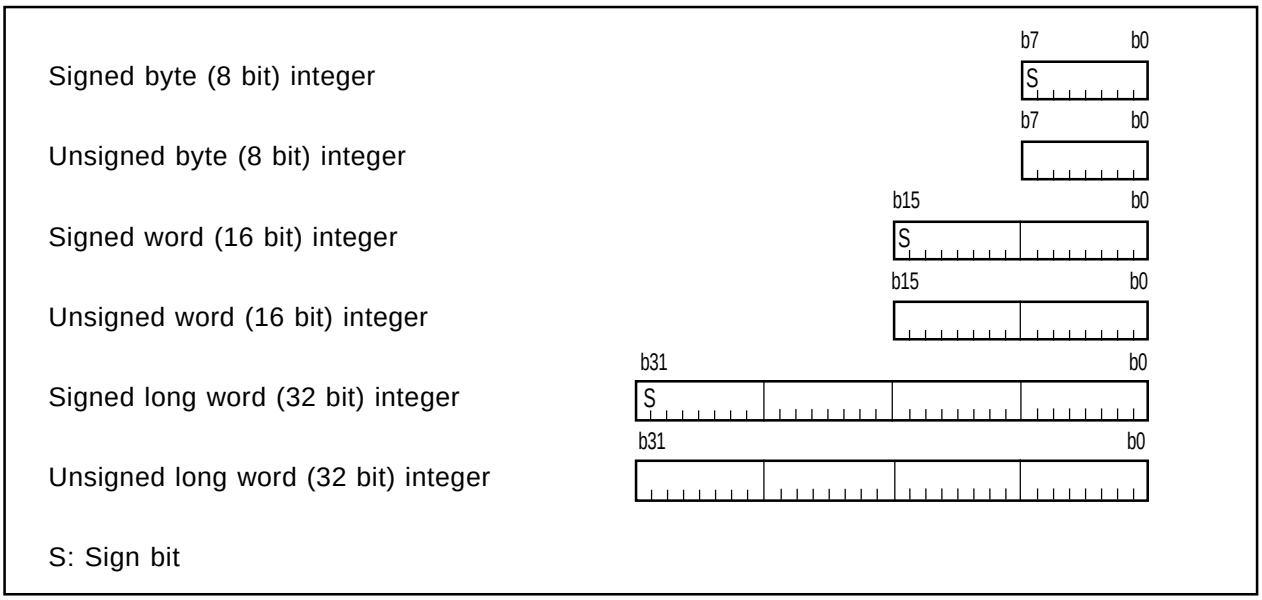


Figure 1.7.1 Integer data

1.7.2 Decimal

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

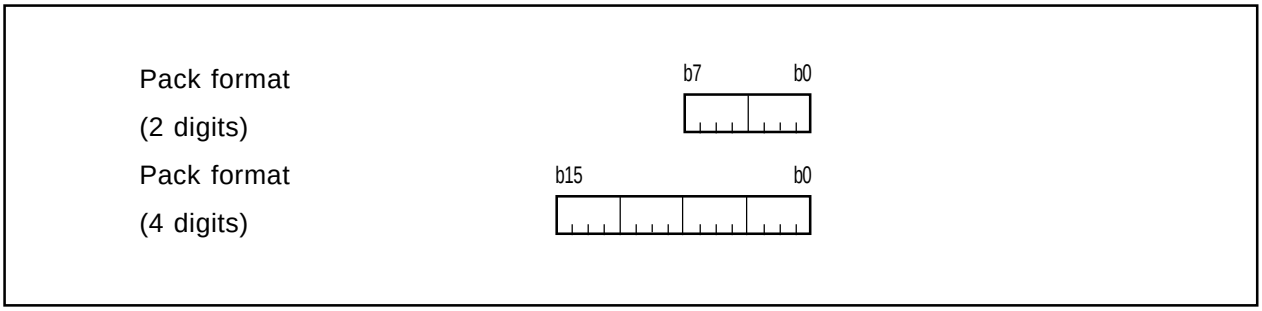


Figure 1.7.2 Decimal data

1.7.3 Bits

(1) Register bits

Figure 1.7.3 shows register bit specification.

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

For bit in **bit,RnH/RnL** and **bit,An**, you can specify a bit number in the range of 0 to 7.

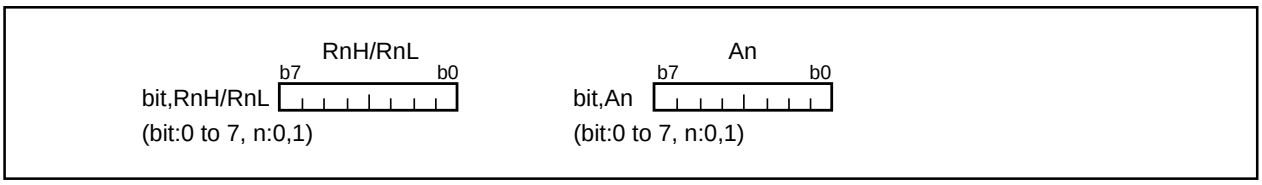


Figure 1.7.3 Register bit specification

(2) 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.

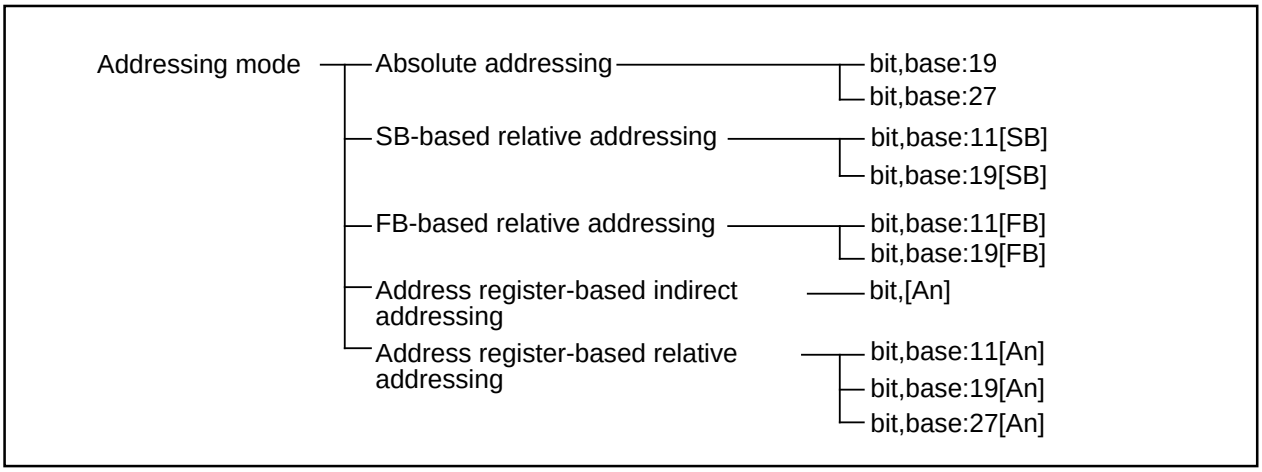


Figure 1.7.4 Addressing modes used for memory bit specification

Table 1.7.1 Bit-Specifying Address Range

Addressing	Specification range		The access range
	Lower limit (address)	Upper limit (address)	
bit,base:19	000000$_{16}$	00FFFF$_{16}$	
bit,base:27	000000$_{16}$	FFFFFF$_{16}$	
bit,base:11[SB]	[SB]	[SB]+000FF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,base:19[SB]	[SB]	[SB]+0FFFF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,base:11[FB]	[FB]-000080$_{16}$	[FB]+00007F$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,base:19[FB]	[FB]-008000$_{16}$	[FB]+007FFF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,[An]	000000$_{16}$	FFFFFF$_{16}$	
bit,base:11[An]	[An]	[An]+0000FF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,base:19[An]	[An]	[An]+00FFFF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.
bit,base:27[An]	[An]	[An]+FFFFFF$_{16}$	000000$_{16}$ to FFFFFF$_{16}$.

(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.

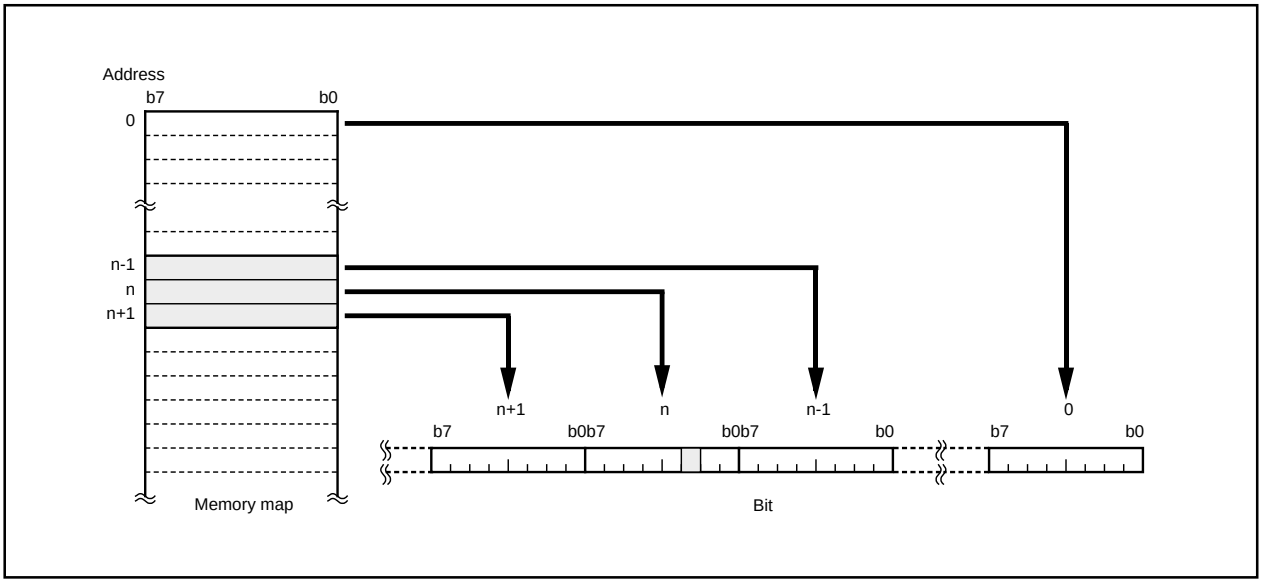


Figure 1.7.5 Relationship between memory map and bit map

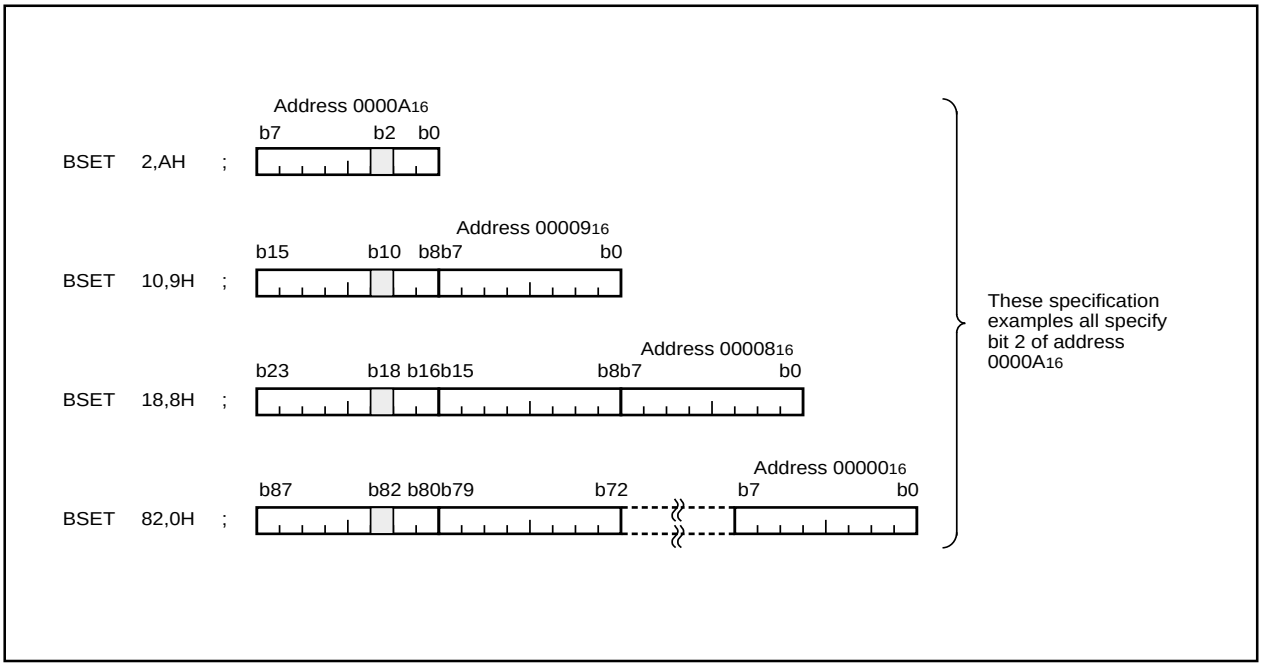


Figure 1.7.6 Examples of how to specify bit 2 of address 0000A16

(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 the desired bit position to **bit**.

(3) Address register indirect/relative bit specification

For address register indirect addressing, use bit 0 of the address that is set to address register(**An**) as the reference (= 0), and set the desired bit position to **bit**.

For address register indirect addressing, specified bit range is 0 to 7.

For address register relative addressing, use bit 0 of the address that is the sum of the address set to address register (**An**) plus the address set to **base** as the reference (= 0), and set the desired bit position to **bit**.

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 seven types of string instructions: character string backward transfer (SMOVB instruction), character string forward transfer (SMOVF instruction), specified area initialize (SSTR instruction), character string transfer compare (SCMPU instruction), character string transfer (SMOVU instruction), character string input (SIN instruction) and character string output (SOUT instruction).

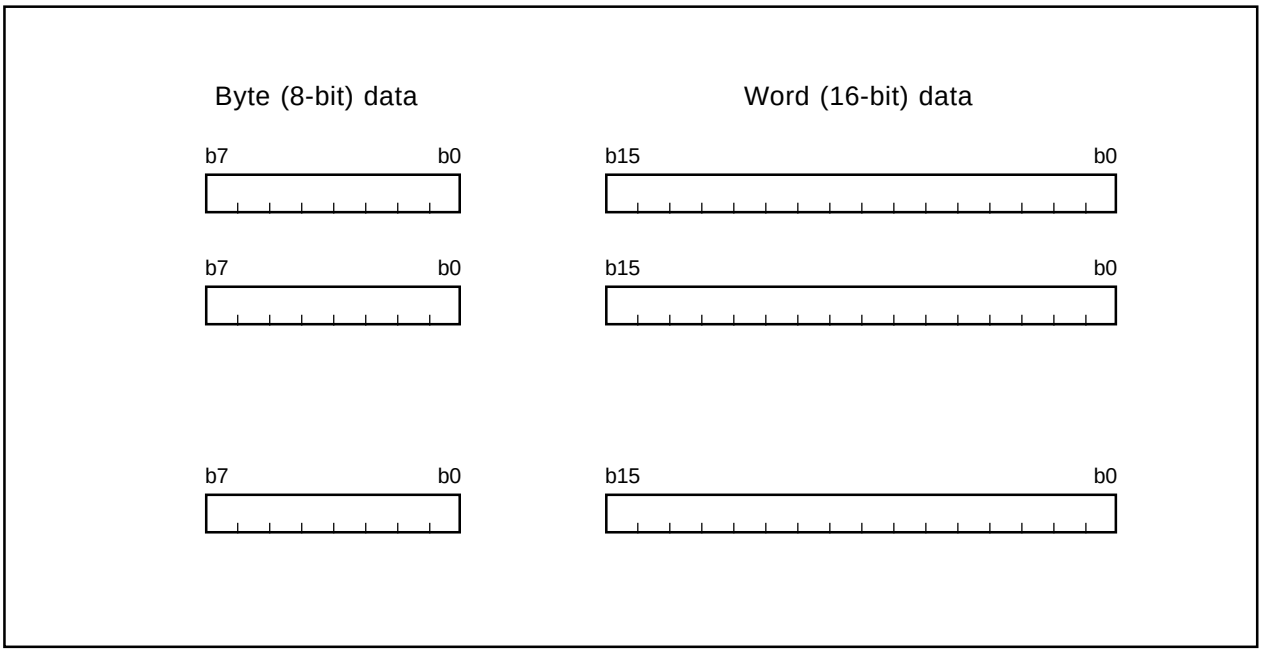


Figure 1.7.7 String data

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.

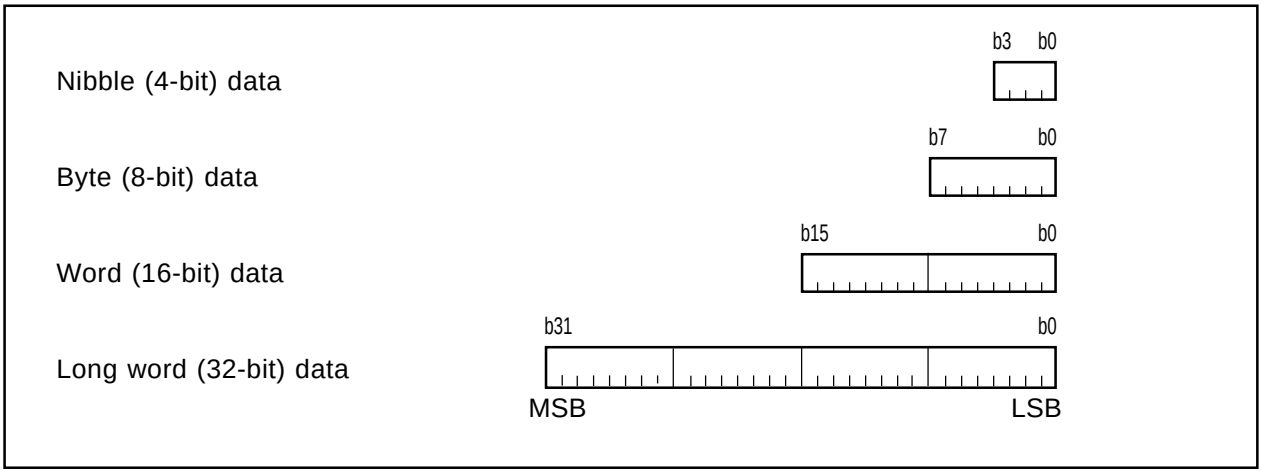


Figure 1.8.1 Data arrangement in register

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.

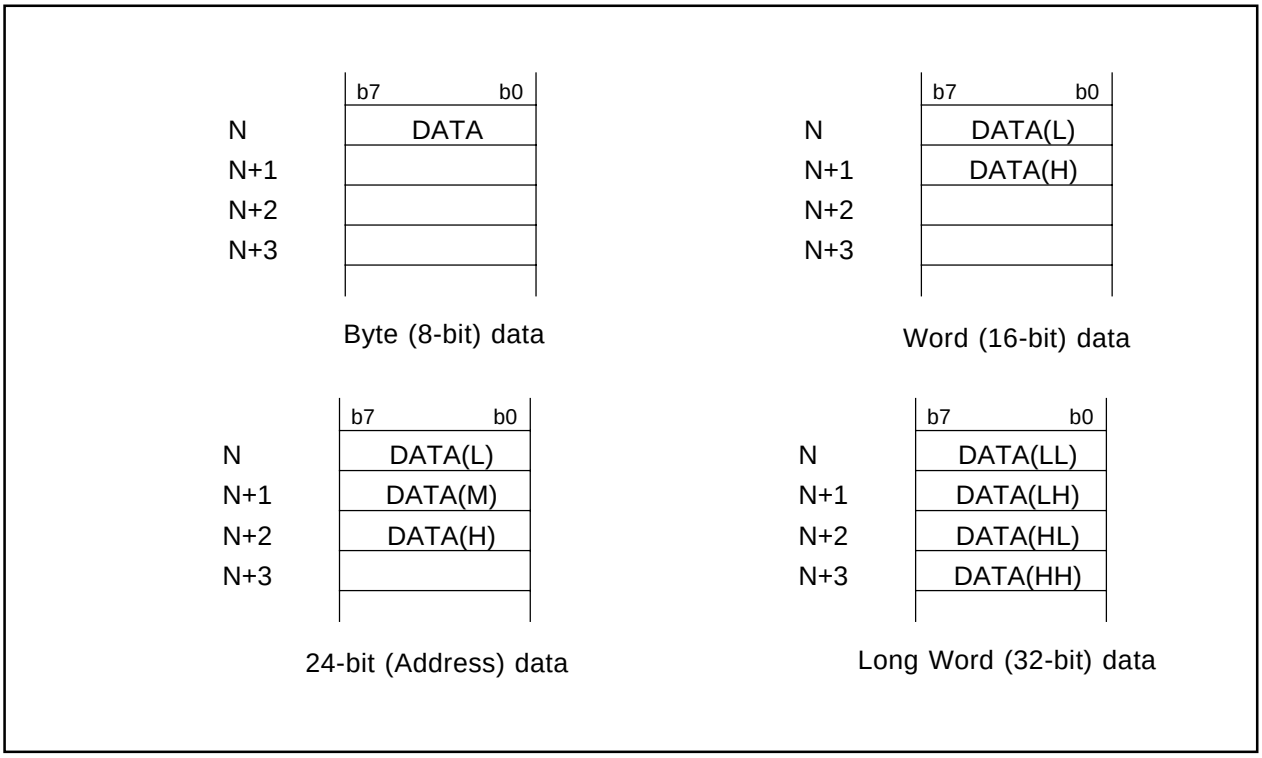


Figure 1.8.2 Data arrangement in memory

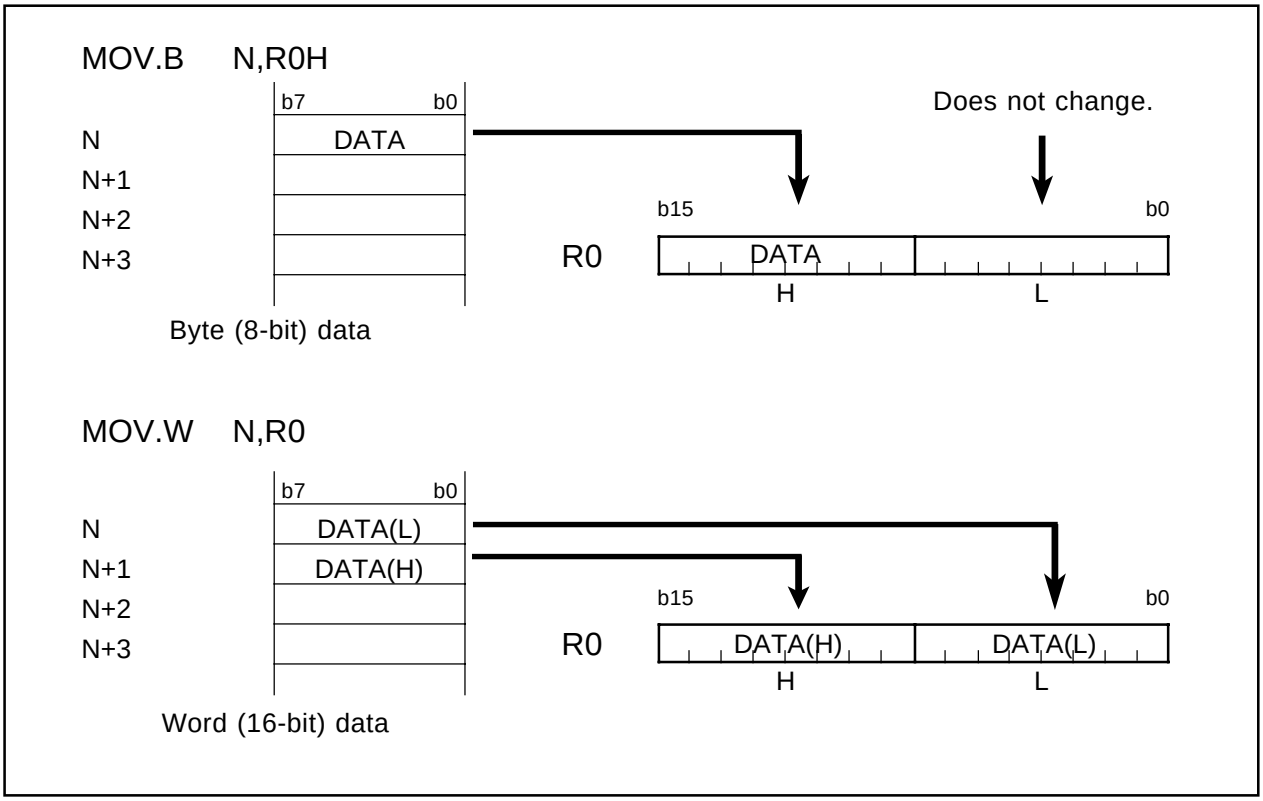


Figure 1.8.3 Examples of operation

1.9 Instruction Format

The instruction format can be classified into four types: generic, quick, short, and zero. The number of bytes in the instruction 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) Generic format (:G)

Op-code in this format consists of 2-3 bytes. This op-code contains information on operation and src*1 and dest*2 addressing modes.
Instruction code here is comprised of op-code (2-3 bytes), src code (0-4 bytes), and dest code (0-3 bytes).

(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 (0-3 bytes).

(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).

(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 FFFE00$_{16}$ through FFFFDB$_{16}$, and part of the interrupt vector table is allocated to addresses FFFFDC$_{16}$ through FFFFFF$_{16}$. 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.

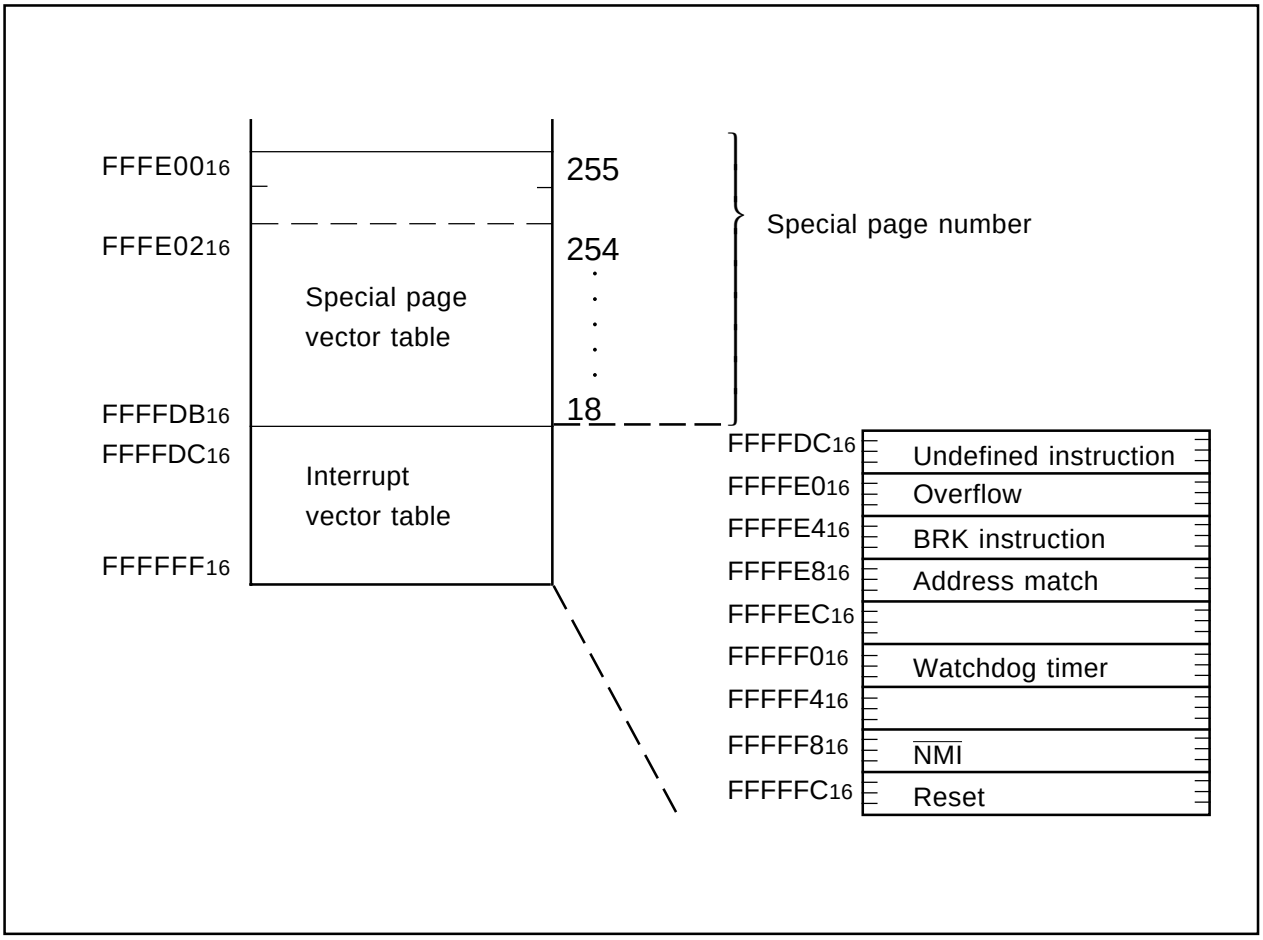


Figure 1.10.1 Fixed vector table

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.

The built-in peripheral I/O interrupts are assigned to variable vector table by MCU type expansion. Interrupts from the internal peripheral functions are assigned from software interrupt numbers 0. The number of interrupts is different depending on MCU type.

The stack pointer (SP) used for INT instruction interrupts varies with each software interrupt number. For software interrupt numbers 0 through 31, the stack pointer specifying flag (U flag) is saved when an interrupt request is accepted and the interrupt sequence is executed after clearing the U flag to 0 and selecting the interrupt stack pointer (ISP). The U flag that was saved before accepting the interrupt request is restored upon returning from the interrupt handler routine.

For software interrupt numbers 32 through 63, the stack pointer is not switched over.

For peripheral I/O interrupts, the interrupt stack pointer (ISP) is selected irrespective of software interrupt numbers when accepting an interrupt request as for software interrupt numbers 0 through 31.

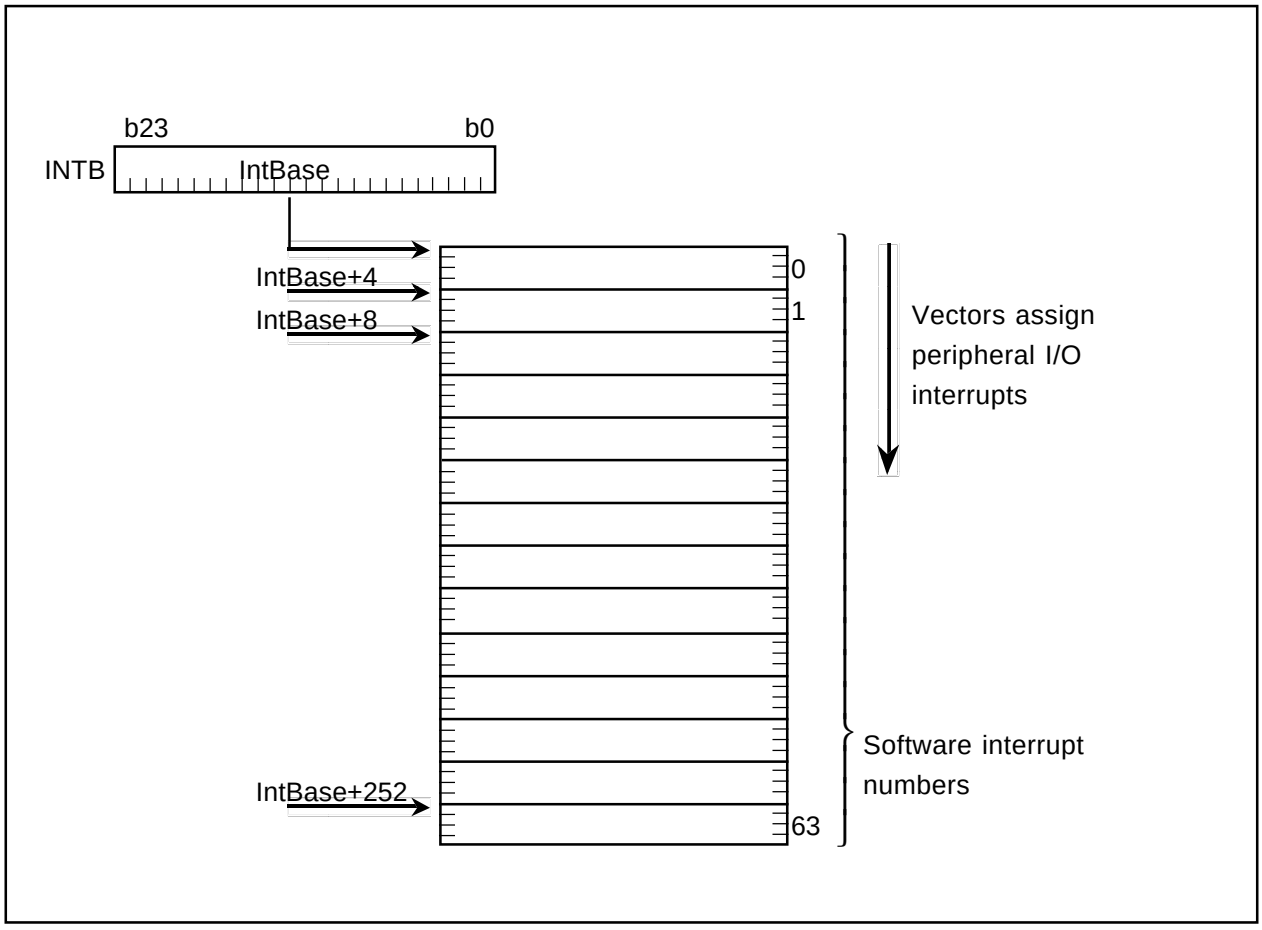


Figure 1.10.2 Variable vector table

Chapter 2

Addressing Modes

2.1 Addressing Modes

This section describes addressing mode-representing symbols and operations for each addressing mode. The M32C/80 has four addressing modes outlined below.

(1) General instruction addressing

This addressing accesses an area from address 000000_{16} through address $FFFFFF_{16}$.

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) Indirect instruction addressing

This addressing accesses an area from address 000000_{16} through address $FFFFFF_{16}$.

The following lists the name of each indirect instruction addressing:

- Absolute indirect
- Two-stage address register indirect
- Address register relative indirect
- SB relative indirect
- FB relative indirect

(3) Special instruction addressing

This addressing accesses an area from address 000000_{16} through address $FFFFFF_{16}$ and control registers.

The following lists the name of each specific instruction addressing:

- Control register direct
- Program counter relative

(4) Bit instruction addressing

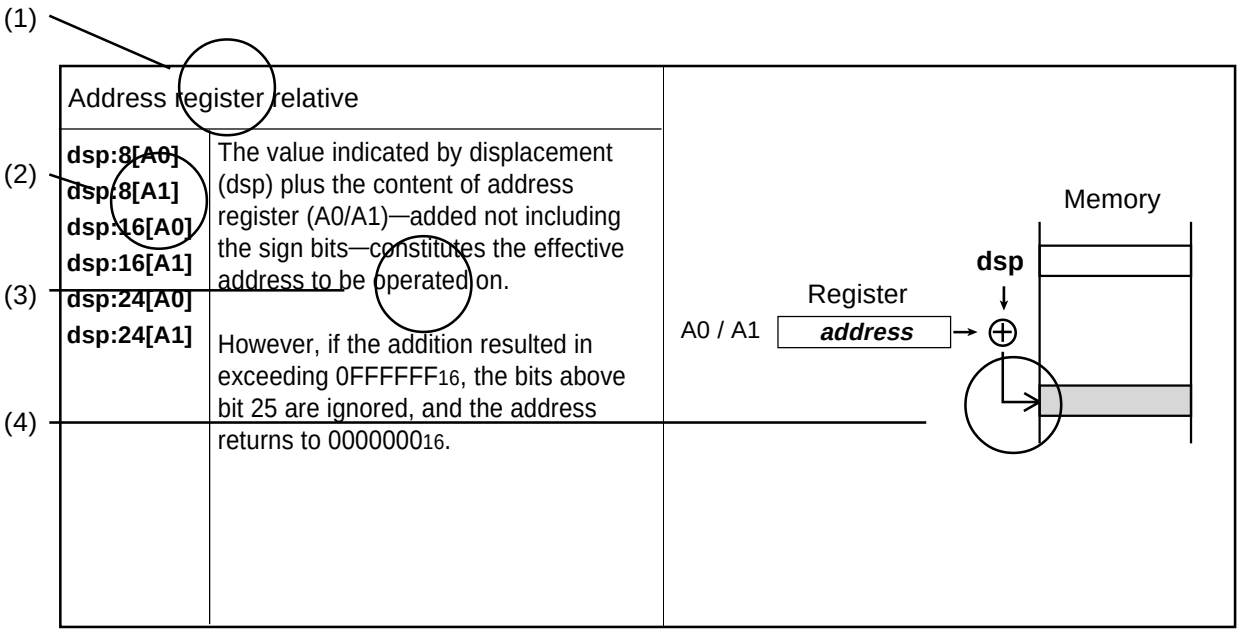
This addressing accesses an area from address 000000_{16} through address $FFFFFF_{16}$.

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.



(1) Name

Indicates the name of addressing.

(2) Symbol

Represents addressing mode.

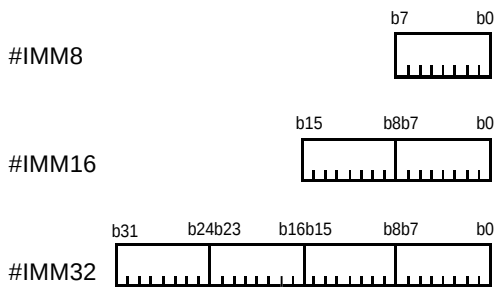
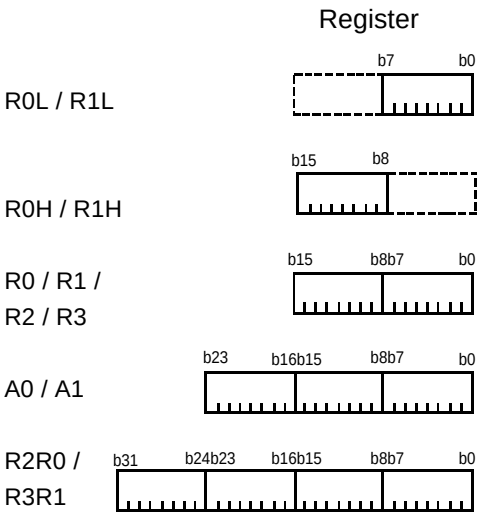
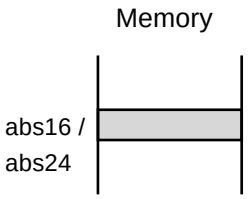
(3) Explanation

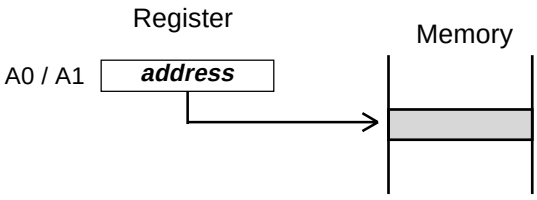
Describes the addressing operation and the effective address range.

(4) Operation diagram

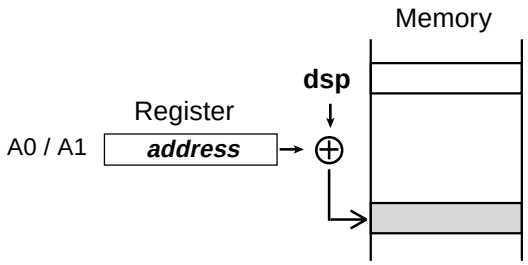
Diagrammatically explains the addressing operation.

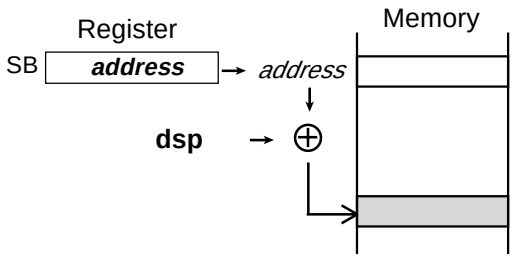
2.3 General Instruction Addressing

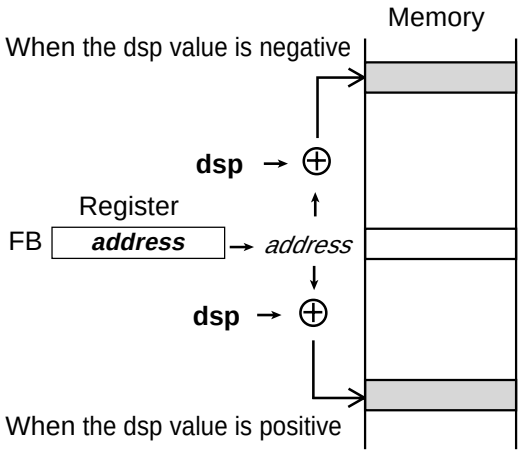
<table>
<tr><td colspan="2">Immediate</td><td rowspan="2"> 
</td></tr>
<tr><td>#IMM
#IMM8
#IMM16
#IMM32</td><td>The immediate data indicated by #IMM is the object to be operated on.</td></tr>
<tr><td colspan="2">Register direct</td><td rowspan="2"> 
</td></tr>
<tr><td>R0L
R0H
R1L
R1H
R0
R1
R2
R3
A0
A1
R2R0
R3R1</td><td>The specified register is the object to be operated on.</td></tr>
<tr><td colspan="2">Absolute</td><td rowspan="2"> 
</td></tr>
<tr><td>abs16
abs24</td><td>The value indicated by abs constitutes the effective address to be operated on.

The effective address range is 000000016 to 000FFFF16 at abs16, and 000000016 to 0FFFFFF16 at abs24.</td></tr>
<tr><td colspan="2">Address register indirect</td><td rowspan="2"> 
</td></tr>
<tr><td>[A0]
[A1]</td><td>The value indicated by the content of address register (A0/A1) constitutes the effective address to be operated on.

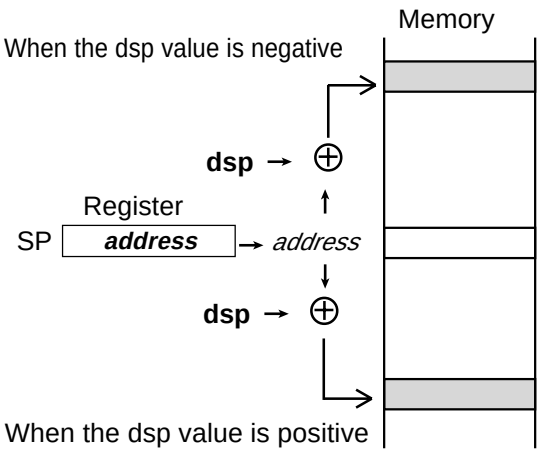
The effective address range is 000000016 to 0FFFFFF16.</td></tr>
</table>

<table>
<tr><td colspan="2">Address register relative</td><td rowspan="2"> 
</td></tr>
<tr><td>dsp:8[A0]
dsp:8[A1]
dsp:16[A0]
dsp:16[A1]
dsp:24[A0]
dsp:24[A1]</td><td>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.

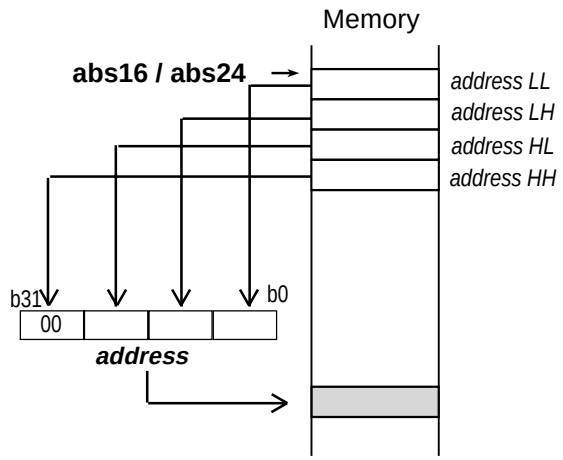
However, if the addition resulted in exceeding 0FFFFFF16, the bits above bit 25 are ignored, and the address returns to 0000000₁₆.</td></tr>
<tr><td colspan="2">SB relative</td><td rowspan="2"> 
</td></tr>
<tr><td>dsp:8[SB]
dsp:16[SB]</td><td>The address indicated by the content of static base register (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 0FFFFFF16, the bits above bit 25 are ignored, and the address returns to 0000000₁₆.</td></tr>
<tr><td colspan="2">FB relative</td><td rowspan="2"> 
</td></tr>
<tr><td>dsp:8[FB]
dsp:16[FB]</td><td>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.

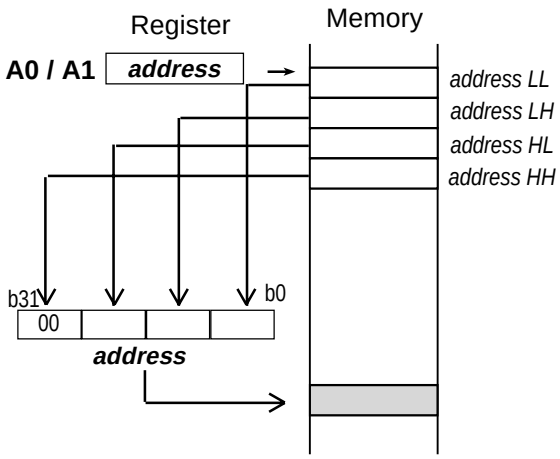
However, if the addition resulted in exceeding 0000000₁₆- 0FFFFFF16, the bits above bit 25 are ignored, and the address returns to 0000000₁₆ or 0FFFFFF16.</td></tr>
</table>

Stack pointer relative		
dsp:8[SP]	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 0000000$_{16}$- 0FFFFFF$_{16}$, the bits above bit 25 are ignored, and the address returns to 0000000$_{16}$ or 0FFFFFF$_{16}$. This addressing can be used in MOV instruction.	

2.4 Indirect Instruction Addressing

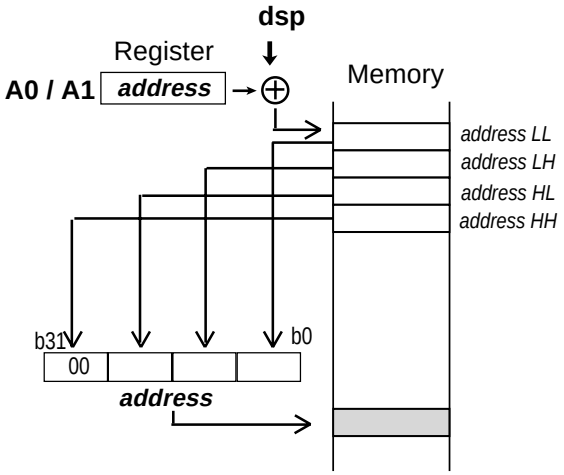
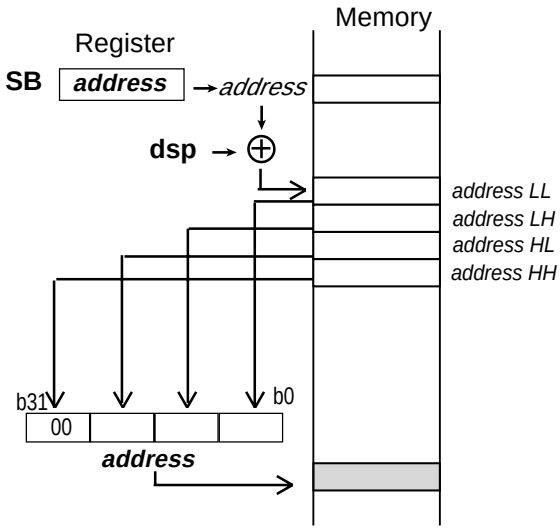
<table>
<tr><td colspan="2">Absolute indirect</td><td rowspan="2"> 

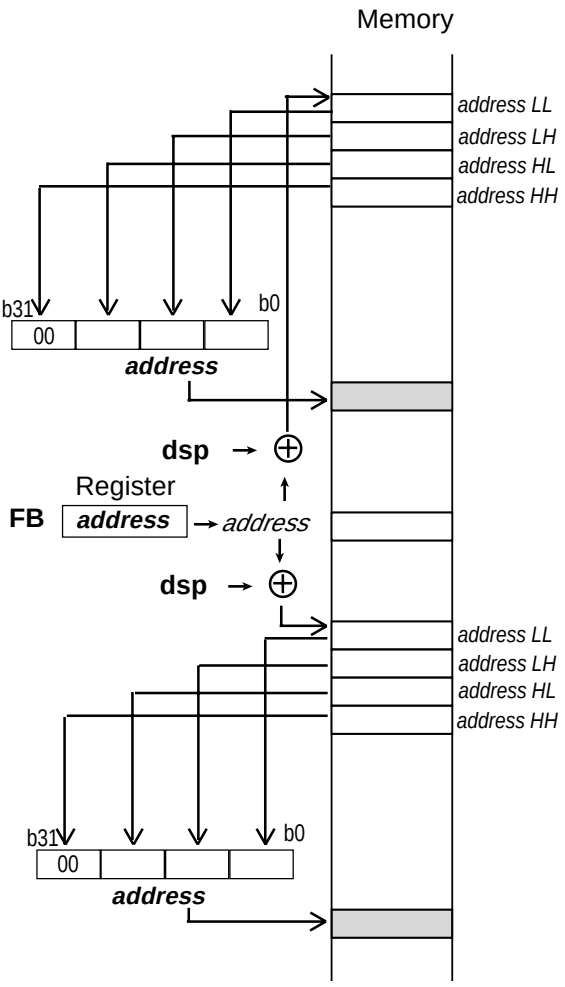
(The upper 8-bit is ignored.)</td></tr>
<tr><td>[abs16]
[abs24]</td><td>The 4-byte value indicated by absolute addressing constitutes the effective address to be operated on.

The effective address range is 0000000₁₆ to 0FFFFFF₁₆.</td></tr>
<tr><td colspan="2">Two-stage address register indirect</td><td rowspan="2"> 

(The upper 8-bit is ignored.)</td></tr>
<tr><td>[[A0]]
[[A1]]</td><td>The 4-byte value indicated by address register (A0/A1) indirect constitutes the effective address to be operated on.

The effective address range is 0000000₁₆ to 0FFFFFF₁₆.</td></tr>
</table>

Address register relative indirect		
[dsp:8[A0]] **[dsp:8[A1]]** **[dsp:16[A0]]** **[dsp:16[A1]]** **[dsp:24[A0]]** **[dsp:24[A1]]**	The 4-byte value indicated by address register relative constitutes the effective address to be operated on. The effective address range is 000000$_{16}$ to 0FFFFFF$_{16}$.	 (The upper 8-bit is ignored.)
SB relative indirect		
[dsp:8[SB]] **[dsp:16[SB]]**	The 4-byte value indicated by SB relative constitutes the effective address to be operated on. The effective address range is 000000$_{16}$ to 0FFFFFF$_{16}$.	 (The upper 8-bit is ignored.)

FB relative indirect		
[dsp:8[FB]] **[dsp:16[FB]]**	The 4-byte value indicated by FB relative constitutes the effective address to be operated on. The effective address range is 00000000$_{16}$ to 0FFFFFF$_{16}$.	 (The upper 8-bit is ignored.)

Indirecting addressing mode cannot be used since op-code of the following instructions is 3 bytes.

div.L

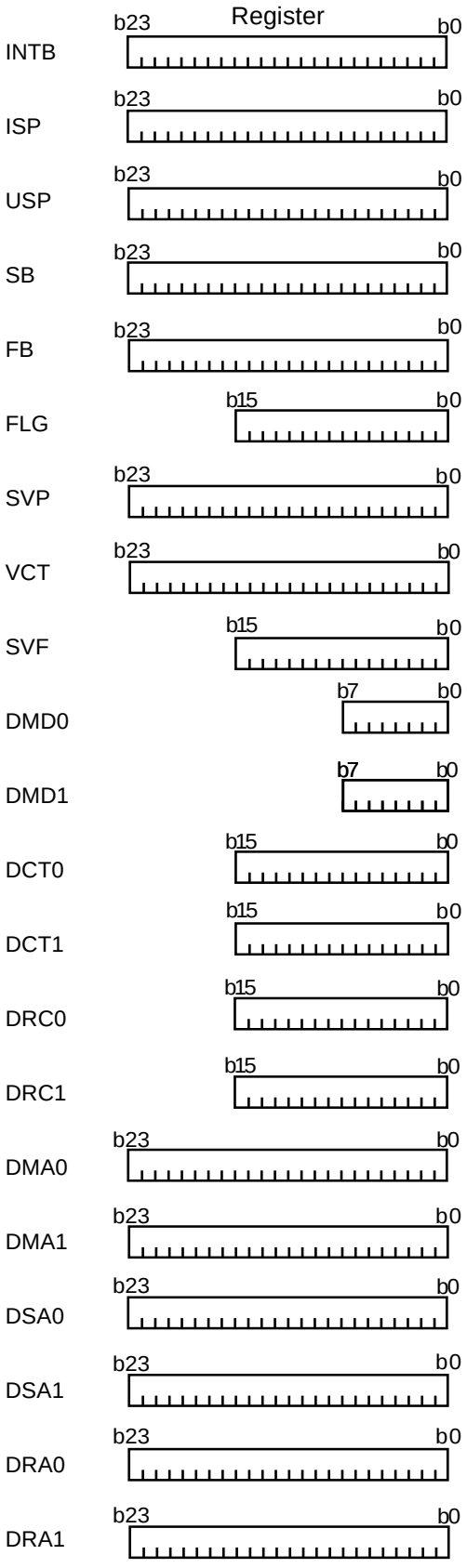
divu.L

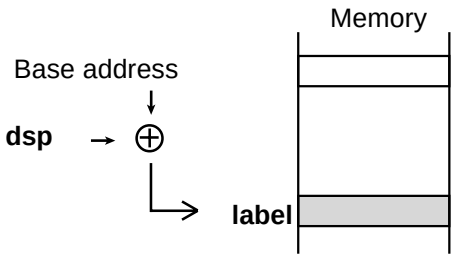
divx.L

mul.L

mulu.L

2.5 Special Instruction Addressing

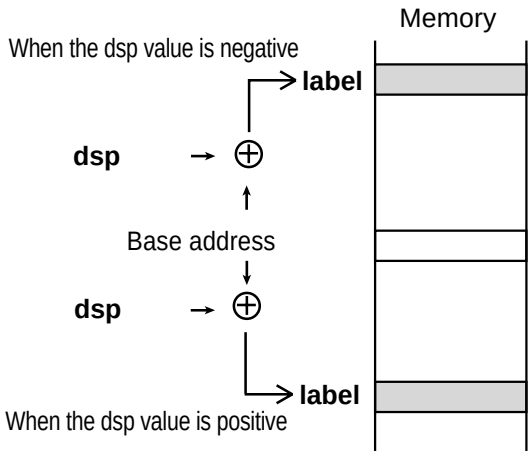
Control register direct		
INTB **ISP** **SP** **SB** **FB** **FLG** **SVP** **VCT** **SVF** **DMD0** **DMD1** **DCT0** **DCT1** **DRC0** **DRC1** **DMA0** **DMA1** **DSA0** **DSA1** **DRA0** **DRA1**	The specified control register is the object to be operated on. This addressing can be used in LDC and STC instructions. If you specify SP, the stack pointer indicated by the U flag is the object to be operated on.	

<table>
<tr><td colspan="2">Program counter relative</td><td rowspan="2"> 

+0 ≦ dsp ≦ +7
*1 The base address is the (start address of instruction + 2).</td></tr>
<tr><td>label</td><td>• When 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.</td></tr>
<tr><td></td><td>• When 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 00000000₁₆- 0FFFFFF₁₆, the bits above bit 25 are ignored, and the address returns to 00000000₁₆ or 0FFFFFF₁₆.

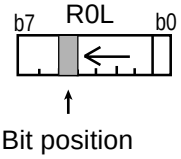
This addressing can be used in JMP and JSR instructions.</td><td> 

When the specifier is (.B), -128 ≤ dsp ≤ +127
When the specifier is (.W), -32768 ≤ dsp ≤ +32767

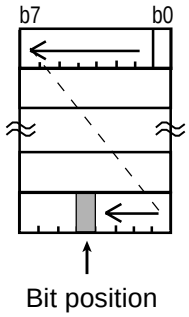
*2 The base address varies with each instruction.</td></tr>
</table>

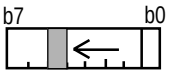
2.6 Bit Instruction Addressing

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

<table>
<tr><td colspan="2">Register direct</td><td rowspan="2">bit , R0L
 
</td></tr>
<tr><td>bit,R0L
bit,R0H
bit,R1L
bit,R1H
bit,A0
bit,A1</td><td>The specified register bit is the object to be operated on.

For the bit position (bit) you can specify 0 to 7.

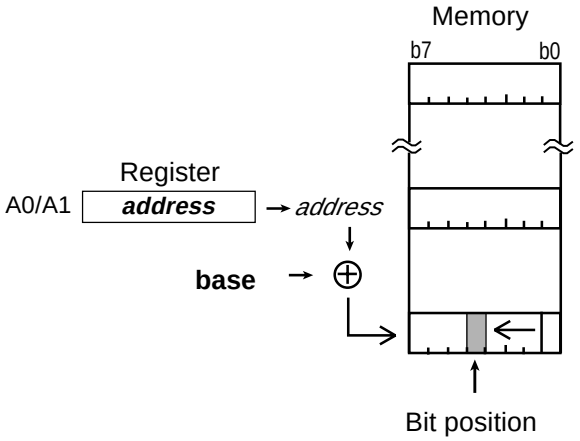
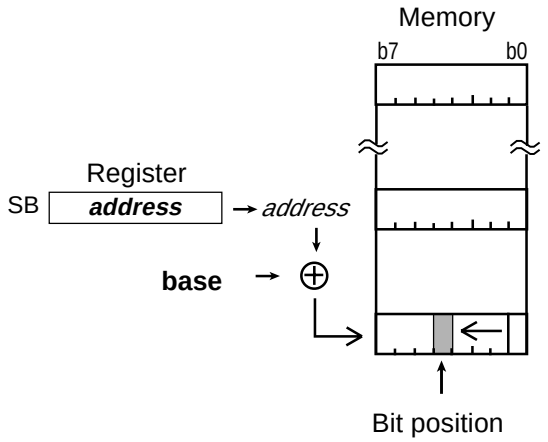
For the address register (A0,A1), you can specify 8 low-order bits.</td></tr>
<tr><td colspan="2">Absolute</td><td rowspan="2">base
 
</td></tr>
<tr><td>bit,base:19
bit,base:27</td><td>The bit that is as much away from bit 0 at the address indicated by base as the number of bits indicated by bit is the object to be operated on.

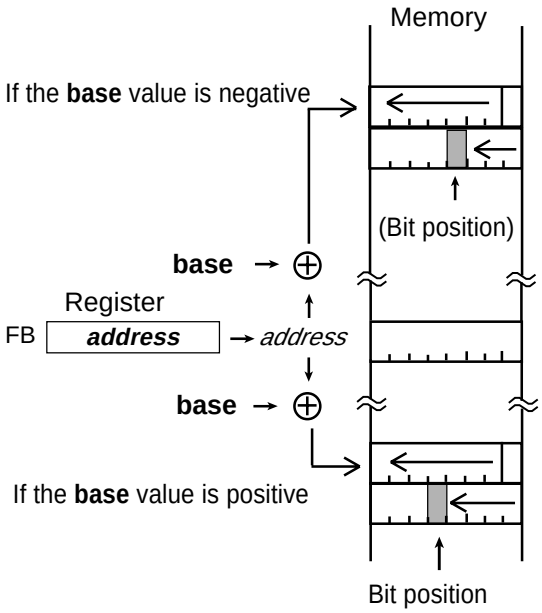
The address range that can be specified by bit,base:19 and bit,base:27 respectively are 00000016 through 000FFFF16 and 00000016 through 0FFFFFF16.</td></tr>
<tr><td colspan="2">Address register indirect</td><td rowspan="2">A0/A1 Register address
 

Bit position</td></tr>
<tr><td>bit,[A0]
bit,[A1]</td><td>The bit that is as much away from bit 0 at address indicated by address register (A0/A1) as the number of bits is the object to be operated on.

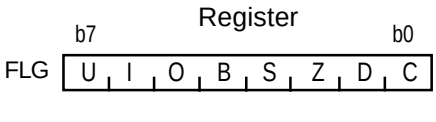
Bits at addresses 00000016 through 0FFFFFF16 can be the object to be operated on.

For the bit position (bit) you can specify 0 to 7.</td></tr>
</table>

Address register relative		
bit,base:11[A0] **bit,base:11[A1]** **bit,base:19[A0]** **bit,base:19[A1]** **bit,base:27[A0]** **bit,base:27[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 0FFFFFF16, the bits above bit 25 are ignored and the address returns to 0000000016. The address range that can be specified by bit,base:11, bit,base:19 and bit,base:27 respectively are 256 bytes, 65,536 bytes and 16,777,216 bytes from address register (A0/A1) value.	
SB relative		
bit,base:11[SB] **bit,base:19[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 0FFFFFF16, the bits above bit 25 are ignored and the address returns to 0000000016. The address ranges that can be specified by bit,base: 11, and bit,base:19 respectively are 256 bytes, and 65,536 bytes from the static base register (SB) value.	

<table>
<tr><td colspan="2">FB relative</td><td rowspan="2"> 
</td></tr>
<tr><td>bit,base:11[FB]
bit,base:19[FB]</td><td>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 0000000₁₆-0FFFFFF₁₆, the bits above bit 25 are ignored and the address returns to 0000000₁₆ or 0FFFFFF₁₆.

The address range that can be specified by bit,base:11 and bit,base:19 are 128 bytes toward lower addresses or 127 bytes toward higher addresses from the frame base register (FB) value, and 32,768 bytes toward lower addresses or 32,767 bytes toward higher addresses, respectively.</td></tr>
<tr><td colspan="2">FLG direct</td><td rowspan="2"> 
</td></tr>
<tr><td>U
I
O
B
S
Z
D
C</td><td>The specified flag is the object to be operated on.

This addressing can be used in FCLR and FSET instructions.</td></tr>
</table>

2.7 Read and write operations with 24-bit registers

This section describes operation when 24 bits register (A0, A1) is src or dest for each size specifier (.size/ .B .W .L).

When (.B) is specified for the size specifier (.size)

• Read

The 8 low-order bits are read. The flags change states depending on the result of 8-bit operation.

• Write

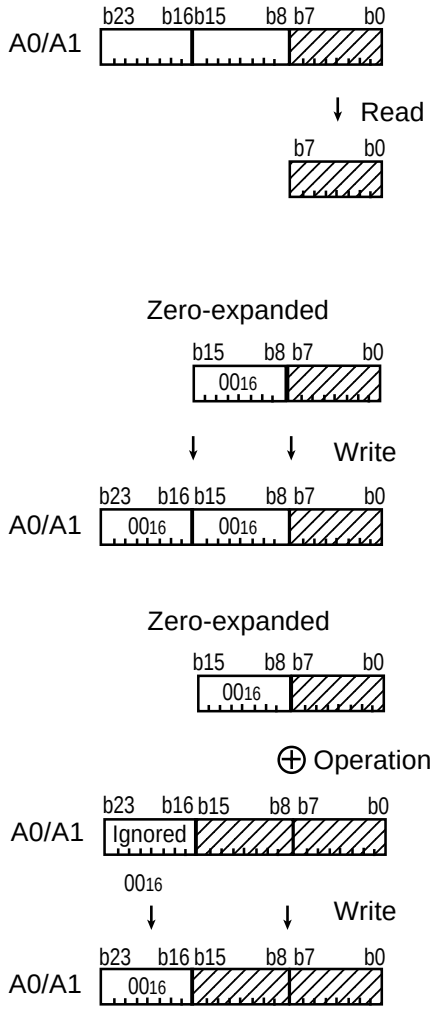
[Transfer instruction]

src is zero-expanded to 16 bits and saved to the low-order 16-bit. In this case, the 8 high-order bits become 0. The flags change states depending on the result of 16-bit transfer data.

[Operating instructions]

src is zero-expanded to perform operation in 16-bit.

In this case, the 8 high-order bits become 0. The flags change states depending on the result of 16-bit operation.



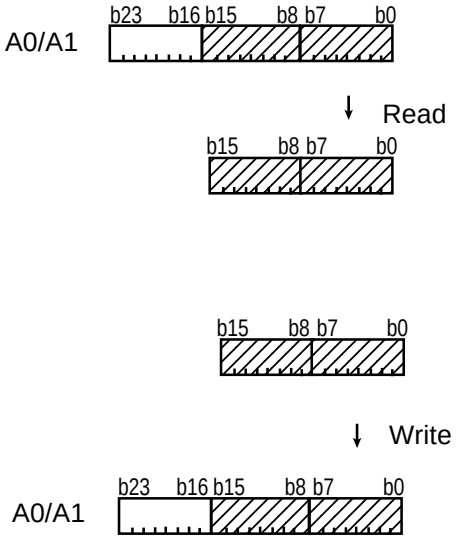
When (.W) is specified for the size specifier (.size)

• Read

The low order 16-bit are read. The flags change states depending on the result of 16-bit operation.

• Write

Write to the low order 16-bit. In this case, the 8 high-order bits become 0. The flags change states depending on the result of 16-bit transfer data.



When (.L) is specified for the size specifier (.size)

• Read

32 bits are read out after being zero-extended. The flag varies depending on the result of a 32-bit operation.

• Write

The low-order 24-bit is written, with the 8 high-order bit ignored. The flag varies depending on the result of a 32-bit operation (not the value of the 24-bit register).

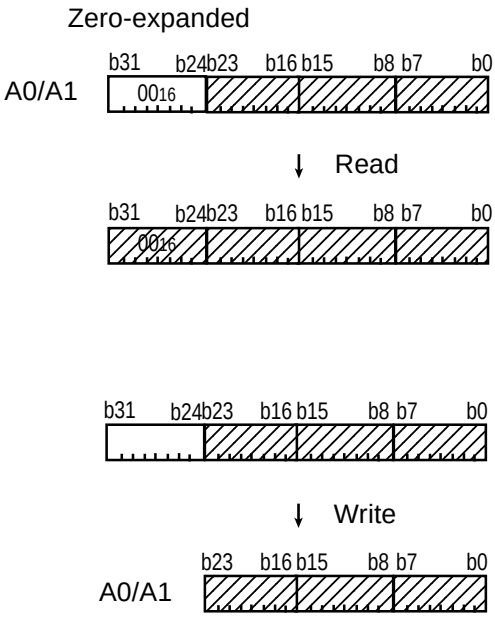
Example: MOV.L#80000000h,A0

Flag status after execution

S flag = 1 (The MSB is bit 31.)

Z flag = 0 (Set to 1 when all of 32 bits are 0s.)

The value of A0 after executing the above instruction becomes 000000_{16}. However, since operation is performed on 32-bit data, the S flag is set to 1 and the Z flag is cleared to 0.

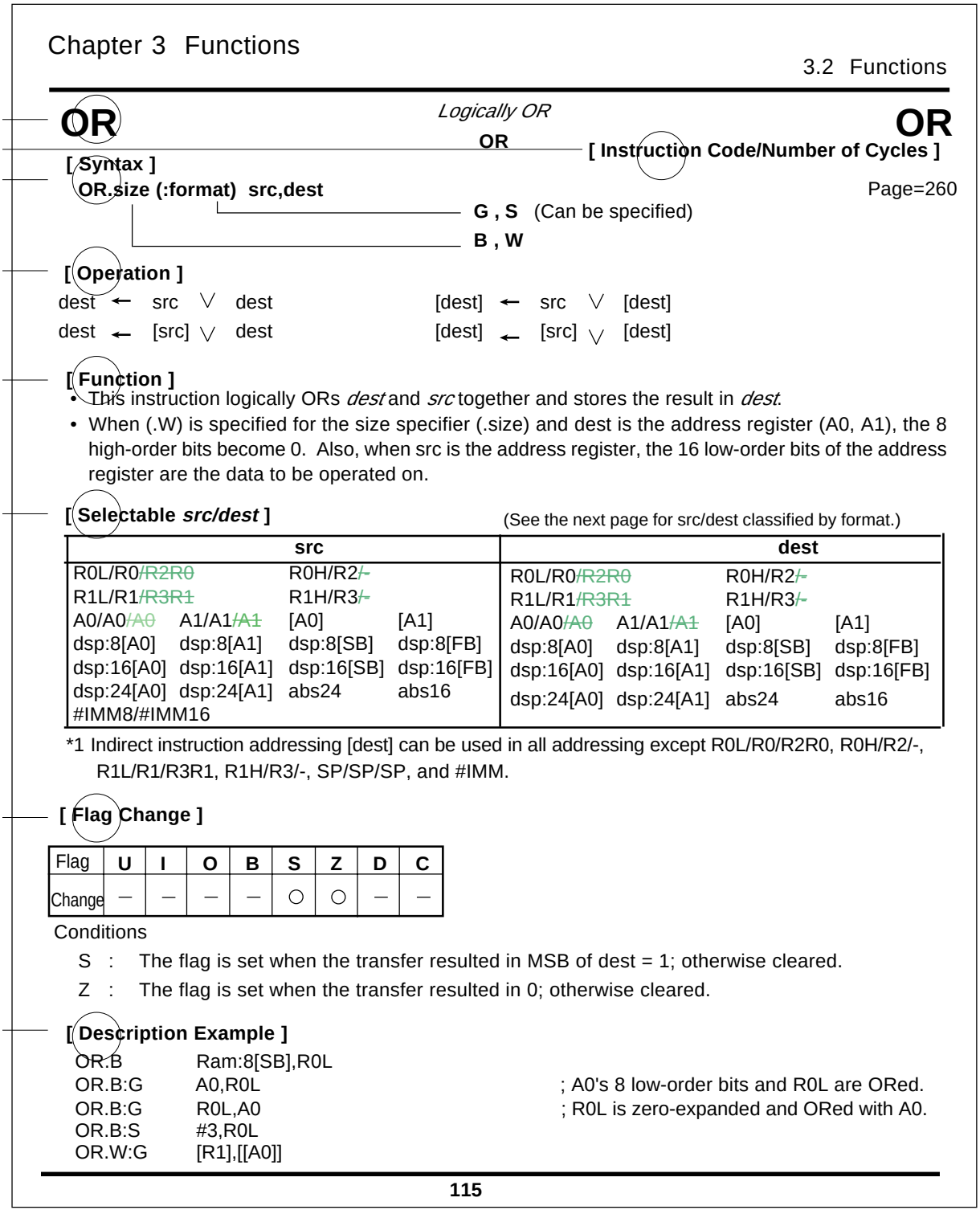


Chapter 3

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, and description examples.
The following shows how to read this chapter by using an actual page as an example.



Chapter 3 Functions

3.2 Functions

(1) **OR** *Logically OR* **OR**

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

(3) **[Syntax]**

OR.size (:format) src,dest

Page=260

G , S (Can be specified)

B , W

(4) **[Operation]**

dest ← src ∨ dest [dest] ← src ∨ [dest]

dest ← [src] ∨ dest [dest] ← [src] ∨ [dest]

(5) **[Function]**

- This instruction logically ORs *dest* and *src* together and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and dest is the address register (A0, A1), the 8 high-order bits become 0. Also, when src is the address register, the 16 low-order bits of the address register are the data to be operated on.

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

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
A0/A0/~~A0~~	A1/A1/~~A1~~	[A0]	[A1]	A0/A0/~~A0~~	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, SP/SP/SP, and #IMM.

(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]**

```
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
OR.W:G    [R1],[[A0]]
```

115

(1) Mnemonic

Indicates the mnemonic explained in this page.

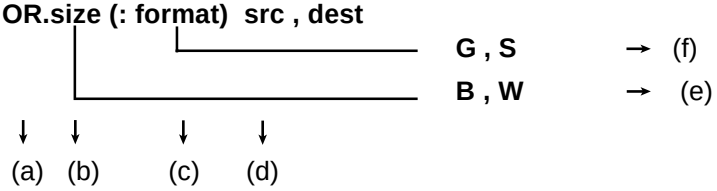
(2) Instruction code/number of cycles

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.

OR.size (: format) src , dest



(a) Mnemonic **OR**

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).

(1) **OR** *Logically OR* **OR**

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

(3) **[Syntax]**

OR.size (:format) src,dest Page=260

G , S (Can be specified)

W , B

(4) **[Operation]**

dest ← src ⋁ dest　　[dest] ← src ⋁ [dest]

dest ← [src] ⋁ dest　　[dest] ← [src] ⋁ [dest]

(5) [Function]

- This instruction logically ORs *dest* and *src* together and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and dest is the address register (A0, A1), the 8 high-order bits become 0. Also, when src is the address register, the 16 low-order bits of the address register are the data to be operated on.

(6) [Selectable *src/dest*]

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

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/-		R0L/R0/~~R2R0~~		R0H/R2/-	
R1L/R1/~~R3R1~~		R1H/R3/-		R1L/R1/~~R3R1~~		R1H/R3/-	
A0/A0/~~A0~~	A1/A1/~~A1~~	[A0]	[A1]	A0/A0/~~A0~~	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [src]and[dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, SP/SP/SP, and #IMM.

(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]

```
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
OR.W:G    [R1],[[A0]]
```

(4) Operation

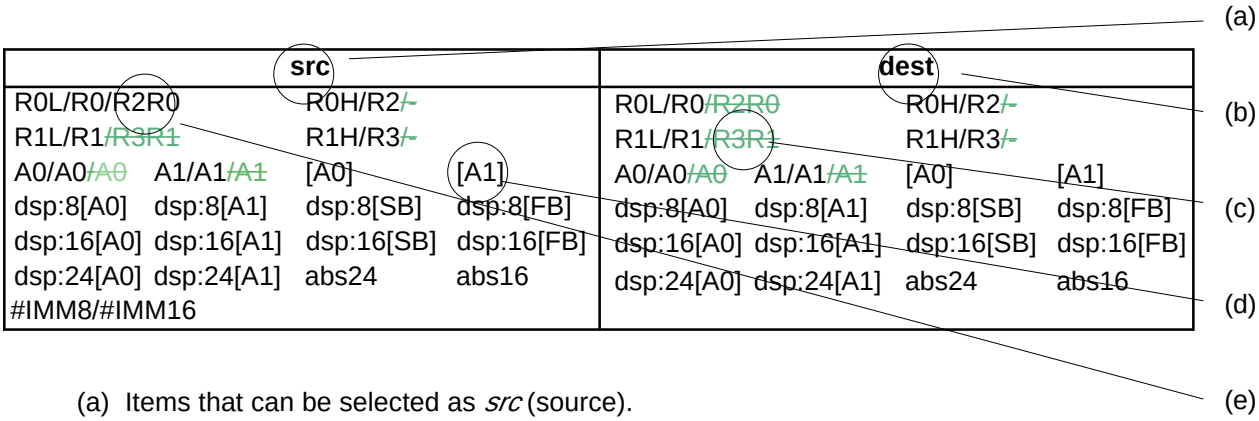
Explains the operation of the instruction using symbols.

(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/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/A1/A1	[A0]	[A1]	A0/A0/A0	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

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

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

(c) Addressing that cannot be selected.

(d) Addressing that can be selected.

(e) Shown on the left side of the slash (R0L) is the addressing when data is handled in bytes (8 bits).
Shown on the middle side of the slash (R0) is the addressing when data is handled in words (16 bits).
Shown on the right side of the slash (R2R0) is the addressing when data is handled in words (32 bits).

(7) Flag change

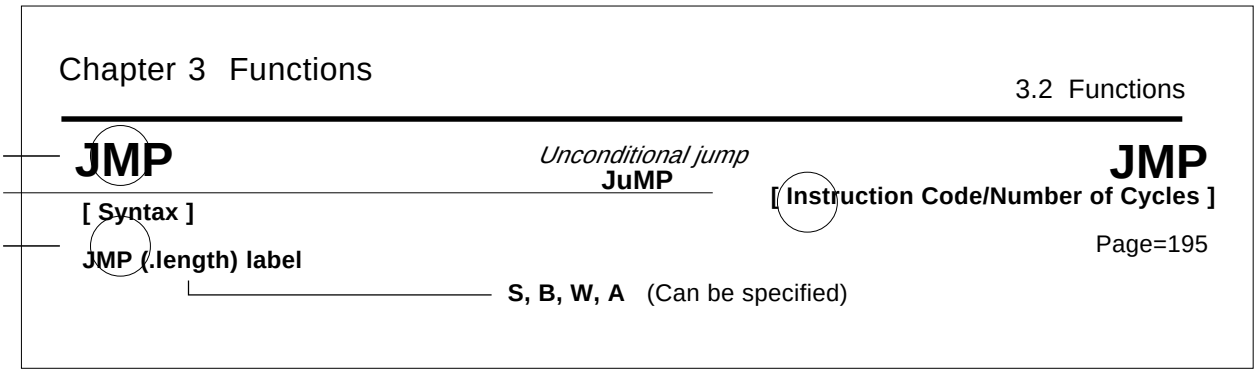
Indicates a flag change that occurs after the instruction is executed. The symbols in the table mean the following:

"—" The flag does not change.
"◯" The flag changes depending on condition.

(8) Description example

Shows a description example for the instruction.

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



(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 24-bit absolute

(c) Operand **label**

Describes the operand.

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

3.2 Functions

ABS

Absolute value
ABSolute

ABS

[Syntax]

ABS.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 174

[Operation]

dest ← | dest |

[dest] ← | [dest] |

[Function]

- This instruction takes on an absolute value of *dest* and stores it in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, R1H/R3~~/-~~.

[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 undefined.

[Description Example]

```
ABS.B   R0L
ABS.W   [A0]
ABS.W   [[A0]]
```

ADC

Add with carry

ADdition with Carry

ADC

[Syntax]

ADC.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 174

[Operation]

dest ← src + dest + C

[Function]

- This instruction adds *dest*, *src* and C flag together and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
A0/A0/~~A0~~*1	A1/A1/~~A1~~*1	[A0]	[A1]	A0/A0/~~A0~~*1	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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.
ADC.W   R1,[A1]
```

ADCF

Add carry flag
ADdition Carry Flag

ADCF

[Syntax]

ADCF.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 176

[Operation]

dest ← dest + C
[dest] ← [dest] + C

[Function]

- This instruction adds *dest* and C flag together and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, R1H/R3~~/-~~.

[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]
```

ADD

Add without carry
ADDition

ADD

[Syntax]

ADD.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page= 176

[Operation]

dest ← dest + src [dest] ← [dest] + src
dest ← dest + [src] [dest] ← [dest] + [src]

[Function]

- This instruction adds *dest* and *src* together and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-order bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.
- When (.L) is specified for the size specifier (.size) and *dest* is the address register, *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*. Also, when *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The flags also change states depending on the result of 32-bit operation.
- When (.L) is specified for the size specifier (.size) and *dest* is SP, *dest* is zero-extended to be treated as 32-bit data for the operation, and src is sign-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*. The flags also change states depending on the result of 32-bit operation.

[Selectable src/dest]*1 (See the next page for *src*/*dest* classified by format.)

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM				~~SP/SP/~~SP*3			

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, ~~SP/SP/~~SP, and #IMM.
*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/or A1 for *src* and *dest* simultaneously.
*3 Operation is performed on the stack pointer indicated by the U flag.

[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 +2147483647(.L) or -2147483648(.L), +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 +4294967295(.L) or +65535 (.W) or +255 (.B); otherwise cleared.

[Description Example]

```
ADD.B   [[A0]],abs16
```

[src/dest Classified by Format]

G format*4

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*5	A1/A1/A1*5	[A0]	[A1]	A0/A0/A0*5	A1/A1/A1*5	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32				~~SP/SP/~~SP*6			

*4 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, ~~SP/SP/~~SP, and #IMM.
*5 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/or A1 for *src* and *dest* simultaneously.
*6 Operation is performed on the stack pointer indicated by the U flag. You can choose only #IMM16 for *src*. You can choose only (.L) for the size specifier (.size).
In this case, you cannot use indirect instruction addressing mode.

Q format*7

src				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0/R2R0		R0H/R2/-	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1/R3R1		R1H/R3/-	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0/A0	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM3*9/#IMM4*10				~~SP/SP/~~SP*8			

*7 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and ~~SP/SP/~~SP.
*8 Operation is performed on the stack pointer indicated by the U flag. You can choose only #IMM3 for *src*.
*9 When *dest* is the SP, #IMM3 can be selected. The range of values is +1 ≤ #IMM3 ≤ +8.
*10 When *dest* is not the SP, #IMM4 can be selected. The range of values is -8 ≤ #IMM4 ≤ +7.

S format*11

src				dest			
~~R0L/R0~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~abs16~~	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16*12							
#1*13	#2*13			A0*10	A1*13		
#IMM8*13				SP*13			

*11 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0, and SP.
*12 You can choose the (.B) and (.W) for the size specifier (.size).
*13 You can choose only (.L) for the size specifier (.size). In this case, you cannot use indirect instruction addressing mode.

ADDX

Add extend sign without carry
ADDition eXtend sign

ADDX

[Syntax]

ADDX **src,dest**

[Instruction Code/Number of Cycles]

Page=183

[Operation]

dest ← dest + EXTS(src) [dest] ← [dest] + EXTS(src)
dest ← dest + EXTS([src]) [dest] ← [dest] + EXTS([src])

[Function]

- Sign-extend the 8-bit *src* to 32 bits which are added to the 32-bit *dest*, and the result is stored in *dest*.
- When *dest* is the address register (A0, A1) , *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*. The flags also change states depending on the result of 32-bit operation. Also, when *src* is the address register, *src* is zero-extended to be treated as 8 low-order-bit data for the operation.

[Selectable src/dest]*1

src				dest			
R0L/~~R0~~/R2R0		R0H/~~R2~~/-		~~R0L~~/R0/R2R0		~~R0H~~/R2/-	
R1L/~~R1~~/R3R1		R1H/~~R3~~/-		~~R1L~~/R1/R3R1		~~R1H~~/R3/-	
A0/~~A0~~/A0	A1/~~A1~~/A1	[A0]	[A1]	~~A0~~/A0/A0	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/~~R0~~/R2R0, R0H/R2/-, R1L/~~R1~~/R3R1, R1H/R3/-, and #IMM.

[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 +2147483647(.L) or -2147483648(.L); 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 +4294967295(.L); otherwise cleared.

[Description Example]

```
ADDX    R0L,A0
ADDX    RAM:8[SB],R2R0
ADDX    [A0],A1
```

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= 185

[Operation]

dest ← dest + src

if dest ≠ 0 **then** jump label

[Function]

- This instruction adds *dest* and *src* together and stores the result in *dest*.
- When the addition resulted in any value other than 0, control jumps to **label**. When the addition resulted in 0, the next instruction is executed.
- The op-code of this instruction is the same as that of SBJNZ.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), the 8 high-oreder bits become 0.

[Selectable src/dest/label]

src	dest				label
	R0L/R0/~~R2R0~~		R0H/R2/~~-~~		
	R1L/R1/~~R3R1~~		R1H/R3/~~-~~		
#IMM4*1	~~A0~~/A0/~~A0~~	~~A1~~/A1/~~A1~~	[A0]	[A1]	PC*2 -126≦label≦PC*2+129
	dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	
	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]	
	dsp:24[A0]	dsp:24[A1]	abs24	abs16	

*1 The range of values is - 8 ≤ #IMM4 ≤ +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
```

AND

Logically AND
AND

AND

[Syntax]

AND.size (:format) src,dest

G , S (Can be specified)
B , W

[Instruction Code/Number of Cycles]

Page=186

[Operation]

dest ← src ∧ dest [dest] ← src ∧ [dest]
dest ← [src] ∧ dest [dest] ← [src] ∧ [dest]

[Function]

- This instruction logically ANDs *dest* and *src* together and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest] *1

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

src				**dest**			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R2R0~~		R1H/R3/~~-~~		R1L/R1/~~R2R0~~		R1H/R3/~~-~~	
A0/A0/~~A0~~*2	A1/A1/~~A1~~*2	[A0]	[A1]	A0/A0/~~A0~~*2	A1/A1/~~A1~~*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/~~R2R0~~, R0H/R2/~~-~~, R1L/R1/~~R3R1~~, R1H/R3/~~-~~, and #IMM.

*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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
AND.W:G   [A0],[[A1]]
```

[src/dest Classified by Format]

G format*3

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R2R0		R1H/R3/-		R1L/R1/R2R0		R1H/R3/-	
A0/A0/A0*4	A1/A1/A1*4	[A0]	[A1]	A0/A0/A0*4	A1/A1/A1*4	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*3 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

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

S format*5

src				dest			
R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16							

*5 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0-, and #IMM.

BAND

Logically AND bits
Bit AND carry flag

BAND

[Syntax]

BAND **src**

[Instruction Code/Number of Cycles]

Page=188

[Operation]

C ← src ∧ C

[Function]

- This instruction logically ANDs the C flag and *src* together and stores the result in the C flag.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BAND    5,[A0]
```

BCLR

Clear bit
Bit CLeaR

BCLR

[Syntax]

BCLR **dest**

[Instruction Code/Number of Cycles]

Page= 188

[Operation]

dest ← 0

[Function]

- This instruction stores 0 in *dest*.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable dest]

dest			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[Flag Change]

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

[Description Example]

```
BCLR    flag
BCLR    4,Ram
BCLR    16,Ram:19[SB]
BCLR    5,[A0]
```

BITINDEX

Bit index
BIT INDEX

BITINDEX

[Syntax]

BITINDEX.size src

B , W

[Instruction Code/Number of Cycles]

Page= 189

[Operation]

[Function]

- This instruction modifies addressing of the next bit instruction.
- No interrupt request is accepted immediately after this instruction.
- The operand specified in *src* constitutes the *src* or *dest* index value for the next bit instruction.
- For details, refer to Section 3.3, "Index Instructions."

[Selectable src]

src			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
A0/A0~~/A0~~	A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs:24	abs:16

[Flag Change]

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

[Description Example]

```
BITINDEX R0
BITINDEX [A0]
```

BM*Cnd*

Conditional bit transfer

Bit Move Condition

BM*Cnd*

[Syntax]

BM*Cnd* **dest**

[Instruction Code/Number of Cycles]

Page=190

[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*. When the condition is true, 1 is transferred; when false, 0 is transferred.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.
- 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∧$\overline{Z}$=1	Greater than	<	LEU	C∧$\overline{Z}$=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,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19
C			

[Flag Change]

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

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

[Description Example]

```
BMN     3,Ram:11[SB]
BMZ     C
```

BNAND

Logically AND inverted bits

Bit Not AND carry flag

BNAND

[Syntax]

BNAND **src**

[Instruction Code/Number of Cycles]

Page=192

[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.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BNAND   5,[A0]
```

BNOR

Logically OR inverted bits
Bit Not OR carry flag

BNOR

[Syntax]

BNOR **src**

[Instruction Code/Number of Cycles]

Page= 192

[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.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BNOR    5,[A0]
```

BNOT

Invert bit
Bit NOT

BNOT

[Syntax]

BNOT **dest**

[Instruction Code/Number of Cycles]

Page=193

[Operation]

dest ← $\overline{\text{dest}}$

[Function]

- This instruction inverts *dest* and stores the result in *dest*.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable dest]

dest			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[Flag Change]

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

[Description Example]

```
BNOT    flag
BNOT    4,Ram
BNOT    16,Ram:19[SB]
BNOT    5,[A0]
```

BNTST

Test inverted bit
Bit Not TeST

BNTST

[Syntax]

BNTST **src**

[Instruction Code/Number of Cycles]

Page= 193

[Operation]

Z ← $\overline{\text{src}}$

C ← $\overline{\text{src}}$

[Function]

- This instruction transfers inverted *src* to the Z flag and inverted *src* to the C flag.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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
BNTST   16,Ram:19[SB]
BNTST   5,[A0]
```

BNXOR

Exclusive OR inverted bits

Bit Not eXclusive OR carry flag

BNXOR

[Syntax]

BNXOR **src**

[Instruction Code/Number of Cycles]

Page= 194

[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.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BNXOR   5,[A0]
```

BOR

Logically OR bits
Bit OR carry flag

BOR

[Syntax]

BOR src

[Instruction Code/Number of Cycles]

Page= 194

[Operation]

C ← src ∨ C

[Function]

- This instruction logically ORs the C flag and *src* together and stores the result in the C flag.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BOR     5,[A0]
```

BRK

Debug interrupt
BReaK

BRK

[Syntax]

BRK

[Instruction Code/Number of Cycles]

Page= 195

[Operation]

• When anything other than FF_{16} exists in the address $FFFFE7_{16}$

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*1 ← (PC + 1)H
SP ← SP - 2
M(SP) ← (PC + 1)ML
PC ← M($FFFFE4_{16}$)

*1 The 8 high-order bits become undefined.

• When FF_{16} exists in the address $FFFFE7_{16}$

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*2 ← (PC + 1)H
SP ← SP - 2
M(SP) ← (PC + 1)ML
PC ← M(IntBase)

*2 The 8 high-order bits become undefined.

[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
```

BRK2

Debug interrupt2
BReaK2

BRK2

[Syntax]

BRK

[Instruction Code/Number of Cycles]

Page=195

[Operation]

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*1 ← (PC + 1)H
SP ← SP - 2
M(SP) ← (PC + 1)ML
PC ← M(0020$_{16}$)

*1 The 8 high-order bits become undefined.

[Function]

- This instruction is provided for exclusive use in debuggers. Do not use it in user programs.
- A BRK2 interrupt is generated.
- The BRK2 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 BRK2 instruction is executed. After the interrupt, the flags change state as shown on the left.

[Description Example]

```
BRK2
```

BSET

Set bit
Bit SET

BSET

[Syntax]

BSET **dest**

[Instruction Code/Number of Cycles]

Page= 196

[Operation]

dest ← 1

[Function]

- This instruction stores 1 in *dest*.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable dest]

dest			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[Flag Change]

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

[Description Example]

```
BSET    flag
BSET    4,Ram
BSET    16,Ram:19[SB]
BSET    5,[A0]
```

BTST

Test bit
Bit TeST

BTST

[Syntax]

BTST (:format) src

G , S (Can be specified)

[Instruction Code/Number of Cycles]

Page= 196

[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.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

G format*1

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

S format

src
bit,base:19

[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
BTST    16,Ram:19[SB]
BTST    5,[A0]
```

BTSTC

Test bit & clear
Bit TeST & Clear

BTSTC

[Syntax]

BTSTC **dest**

[Instruction Code/Number of Cycles]

Page= 197

[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*.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.
- Do not use this instruction for dest in SFR area.

[Selectable dest]

dest			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BTSTC   5,[A0]
```

BTSTS

Test bit & set
Bit TeST & Set

BTSTS

[Syntax]

BTSTS **dest**

[Instruction Code/Number of Cycles]

Page= 198

[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*.
- When *dest* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.
- Do not use this instruction for dest in SFR area.

[Selectable dest]

dest			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BTSTS   5,[A0]
```

BXOR

Exclusive OR bits

Bit eXclusive OR carry flag

BXOR

[Syntax]

BXOR **src**

[Instruction Code/Number of Cycles]

Page= 198

[Operation]

C ← src ∀ C

[Function]

- This instruction exclusive ORs the C flag and *src* together and stores the result in the C flag.
- When *src* is the address register (A0, A1), you can specify the 8 low-order bits for the address register.

[Selectable src]

src			
bit,R0L	bit,R0H	bit,R1L	bit,R1H
bit,A0	bit,A1	bit,[A0]	bit,[A1]
bit,base:11[A0]	bit,base:11[A1]	bit,base:11[SB]	bit,base:11[FB]
bit,base:19[A0]	bit,base:19[A1]	bit,base:19[SB]	bit,base:19[FB]
bit,base:27[A0]	bit,base:27[A1]	bit,base:27	bit,base:19

[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:19[SB]
BXOR    5,[A0]
```

CLIP

CLIP
CLIP

CLIP

[Syntax]

CLIP.size src1, src2, dest

B , W

[Instruction Code/Number of Cycles]

Page= 199

[Operation]

if src1 > dest
then dest ← src1
if src2 < dest
then dest ← src2

[Function]

- Signed compares src1 and *dest* and stores the content of src1 in *dest* if src1 is greater than *dest*. Next, signed compares src2 and *dest* and stores the content of src2 in *dest* if src2 is smaller than *dest*. When src1 ≤ dest ≤ src2, dest is not changed.
- When (.W) is specified for the size specifier (.size), dest is the address register and writing to dest, the 8 high-order bits become 0.
- Src1 and src2 are set "src1<src2".

[Selectable src/dest/label]

src1, src2				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0~~/R2R0~~		R0H/R2/-	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1~~/R3R1~~		R1H/R3/-	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0~~/A0~~/A0~~	~~A1~~/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[Flag Change]

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

[Description Example]

```
CLIP.W    #5,#10,R1
CLIP.W    #-5,#5,[A0]
```

CMP

Compare
CoMPare

CMP

[Syntax]

CMP.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page= 200

[Operation]

dest - src [dest] - src
dest - [src] [dest] - [src]

[Function]

- Each flag bit of the flag register varies depending on the result of subtraction of *src* from *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.L) is specified for the size specifier (.size), and *src* or *dest* is the address register, address register is zero-extended to be treated as 32-bit data for the operation. The flags also change states depending on the result of 32-bit operation.

[Selectable src/dest]*1

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

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4/#IMM8/#IMM16/#IMM32							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

*2 If you specify (.B) for the size specifier (.size), you cannot choose A0 and/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 +2147483647(.L) or -2147483648(.L), +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*3

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*4	A1/A1/A1*4	[A0]	[A1]	A0/A0/A0*4	A1/A1/A1*4	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
~~#IMM4~~/#IMM8/#IMM16/#IMM32							

*3 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.
*4 If you specify (.B) for the size specifier (.size), you cannot choose A0 and/or A1 for *src* and *dest* simultaneously.

Q format*5*6

src				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0~~/A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4*7~~/#IMM8/#IMM16/#IMM32~~							

*5 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.
*6 You can only specify (.B) or (.W) for the size specifier (.size).
*7 The range of values is -8 ≤ #IMM4 ≤ +7.

S format*8*9

src				dest			
~~R0L/R0~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~abs16~~	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16							
~~R0L/R0~~	dsp:8[SB]	dsp:8[FB]	abs16	R0L/R0	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~abs16~~
~~#IMM~~							

*8 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0, and #IMM.
*9 You can only specify (.B) or (.W) for the size specifier (.size).

CMPX

Compare extended sign

CoMPare eXtend sign

CMPX

[Syntax]

CMPX **src,dest**

[Instruction Code/Number of Cycles]

Page=206

[Operation]

dest/[dest] - EXTS(src)

[Function]

- Each flag of the flag register changes state according to the result derived by subtracting the sign-extended 32-bit *src* from the 32-bit *dest*.
- When *dest* is the address register (A0, A1), it is zero-extended to be treated as 32-bit data for the operation and the flags change their states depending on the result.

[Selectable src/dest]*1

src				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0/~~A0	~~A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8							

*1 Indirect instruction addressing [dest] can be used in all addressing except ~~R0L/R0/~~R2R0, ~~R0H/R2/-~~, ~~R1L/R1/~~R3R1, and ~~R1H/R3/-~~.

[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 +2147483647(.L) or -2147483648(.L), 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]

```
CMPX    #10,R2R0
CMPX    #5,A0
```

DADC

Decimal add with carry

Decimal ADdition with Carry

DADC

[Syntax]

DADC.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 206

[Operation]

dest ← src + dest + C

[Function]

- This instruction adds *dest*, *src*, and C flag together in decimal and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0~~/A0/~~A0~~	~~A1~~/A1/~~A1~~	[A0]	[A1]	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[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
DADC.W   [A0],R2
```

DADD

Decimal add without carry
Decimal ADDition

DADD

[Syntax]

DADD.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 208

[Operation]

dest ← src + dest

[Function]

- This instruction adds *dest* and *src* together in decimal and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				**dest**			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0~~/A0/~~A0~~	~~A1~~/A1/~~A1~~	[A0]	[A1]	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[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
DADD.W   [A0],[A1]
```

DEC

Decrement
DECrement

DEC

[Syntax]

DEC.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 210

[Operation]

dest ← dest - 1 [dest] ← [dest] - 1

[Function]

- This instruction decrements 1 from *dest* and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
DEC.W   R0
```

DIV

Signed divide
DIVide

DIV

[Syntax]

DIV.size src

B , W , L

[Instruction Code/Number of Cycles]

Page= 210

[Operation]

- When the size specifier (.size) is (.L) R2R0 (quotient) ← R2R0 ÷ *src*
- When the size specifier (.size) is (.W) R0 (quotient), R2 (remainder) ← R2R0 ÷ src/[src]
- When the size specifier (.size) is (.B) R0L (quotient), R0H (remainder) ← R0 ÷ src/[src]

[Function]

- When (.B) is specified for the size specifier (.size), this instruction divides R0 by signed *src* and stores the quotient in R0L and the remainder in R0H. The remainder's sign is the same as the dividend's sign. When *src* is the address register (A0, A1), the 8 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 8 bits or the divider is 0. R0L and R0H is undefined.
- When (.W) is specified for the size specifier (.size), this instruction divides R2R0 by signed *src* and stores the quotient in R0 and the remainder in R2. The remainder's sign is the same as the divider's sign. When *src* is the address register, the 16 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 16 bits or the divider is 0. R0 and R2 is undefined.
- When (.L) is specified for the size specifier (.size), this instruction divides R2R0 by signed *src* and stores the quotient in R2R0. The remainder is not operated, but the remainder's sign is the same as the divider's sign. When *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The O flag is set when the divider is 0. R2R0 is undefined.

[Selectable src]

src*1			
R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/##IMM32			

*1 When (.B) and (.W) are specified for the size specifier (.size), indirect instruction addressing [src] can be used in all addressing except R0L/R0/-, R0H/R2/-, R1L/R1/-, R1/-, R1H/R3/-, and #IMM. When (.size) is (.L), indirect instruction addressing [src] cannot be used.

[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), 8 bits (.B) or the divider is 0; otherwise cleared.

[Description Example]

```
DIV.B   A0          ;A0's 8 low-order bits is the divider.
DIV.B   #4
DIV.W   R0
DIV.W   [[A1]]
DIV.L   R3R1        ;The remainder is not operated.
```

DIVU

Unsigned divide
DIVide Unsigned

DIVU

[Syntax]

DIVU.size src

B , W , L

[Instruction Code/Number of Cycles]

Page= 212

[Operation]

- When the size specifier (.size) is (.L) R2R0 (quotient) ← R2R0 ÷ src
- When the size specifier (.size) is (.W) R0 (quotient), R2 (remainder) ← R2R0 ÷ src/[src]
- When the size specifier (.size) is (.B) R0L (quotient), R0H (remainder) ← R0 ÷ src/[src]

[Function]

- When (.B) is specified for the size specifier (.size), this instruction divides R0 by unsigned *src* and stores the quotient in R0L and the remainder in R0H. When *src* is the address register (A0, A1), the 8 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 8 bits or the divider is 0. R0L and R0H is undefined.
- When (.W) is specified for the size specifier (.size), this instruction divides R2R0 by unsigned *src* and stores the quotient in R0 and the remainder in R2. When *src* is the address register, the 16 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 16 bits or the divider is 0. R0 and R2 is undefined.
- When (.L) is specified for the size specifier (.size), this instruction divides R2R0 by unsigned *src* and stores the quotient in R2R0. The remainder is not operated. When *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The O flag is set when the divider is 0. R2R0 is undefined.

[Selectable src]

src*1			
R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32			

*1 When (.B) and (.W) are specified for the size specifier (.size), indirect instruction addressing [src] can be used in all addressing except R0L/R0/-, R0H/R2/-, R1L/R1/-, R1/-, R1H/R3/-, and #IMM. When (.size) is (.L), indirect instruction addressing [src] cannot be used.

[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), 8 bits (.B) or the divider is 0; otherwise cleared.

[Description Example]

```
DIVU.B   A0          ;A0's 8 low-order bits is the divider.
DIVU.B   #4
DIVU.W   R0
DIVU.W   [[A0]]
DIVU.L   R3R1        ;The remainder is not operated.
```

DIVX

Singed divide
DIVide eXtension

DIVX

[Syntax]

DIVX.size src

B , W , L

[Instruction Code/Number of Cycles]

Page= 213

[Operation]

- When the size specifier (.size) is (.L) R2R0 (quotient) ← R2R0 ÷ src
- When the size specifier (.size) is (.W) R0 (quotient), R2 (remainder) ← R2R0 ÷ src/[src]
- When the size specifier (.size) is (.B) R0L (quotient), R0H (remainder) ← R0 ÷ src/[src]

[Function]

- When (.B) is specified for the size specifier (.size), this instruction divides R0 by signed *src* and stores the quotient in R0L and the remainder in R0H. The remainder's sign is the same as the divider's sign. When *src* is the address register (A0, A1), the 8 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 8 bits or the divider is 0. R0L and R0H is undefined.
- When (.W) is specified for the size specifier (.size), this instruction divides R2R0 by signed *src* and stores the quotient in R0 and the remainder in R2. The remainder's sign is the same as the divider's sign. When *src* is the address register, the 16 low-order bits of the address register are used as data to be operated on. The O flag is set when the operation resulted in the quotient exceeding 16 bits or the divider is 0. R0 and R2 is undefined.
- When (.L) is specified for the size specifier (.size), this instruction divides R2R0 by signed *src* and stores the quotient in R2R0. The remainder is not operated, but the remainder's sign is the same as the divider's sign. When *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The O flag is set when the divider is 0. R2R0 is undefined.

[Selectable src]

src*1			
R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/##IMM32			

*1 When (.B) and (.W) are specified for the size specifier (.size), indirect instruction addressing [src] can be used in all addressing except R0L/R0/-, R0H/R2/-, R1L/R1/-, R1/-, R1H/R3/-, and #IMM. When (.size) is (.L), indirect instruction addressing [src] cannot be used.

[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), 8 bits (.B) or the divider is 0; otherwise cleared.

[Description Example]

```
DIVX.B  A0          ; A0's 8 low-order bits is the divider.
DIVX.B  #4
DIVX.W  R0
DIVX.L  R3R1        ;The remainder is not operated
```

DSBB

Decimal subtract with borrow

Decimal SuBtract with Borrow

DSBB

[Syntax]

DSBB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 215

[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*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				dest			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~		R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0~~/A0~~/A0~~	~~A1~~/A1~~/A1~~	[A0]	[A1]	~~A0~~/A0~~/A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[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
DSBB.W   [A0],[A1]
```

DSUB

Decimal subtract without borrow

Decimal SUBtract

DSUB

[Syntax]

DSUB.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 217

[Operation]

dest ← dest - src

[Function]

- This instruction subtracts *src* from *dest* in decimal and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/-		R0L/R0/~~R2R0~~		R0H/R2/-	
R1L/R1/~~R3R1~~		R1H/R3/-		R1L/R1/~~R3R1~~		R1H/R3/-	
~~A0~~/A0/~~A0~~	~~A1~~/A1/~~A1~~	[A0]	[A1]	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[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
DSUB.W  [A0],[A1]
```

ENTER

Build stack frame
ENTER function

ENTER

[Syntax]

ENTER **src**

[Instruction Code/Number of Cycles]

Page= 219

[Operation]

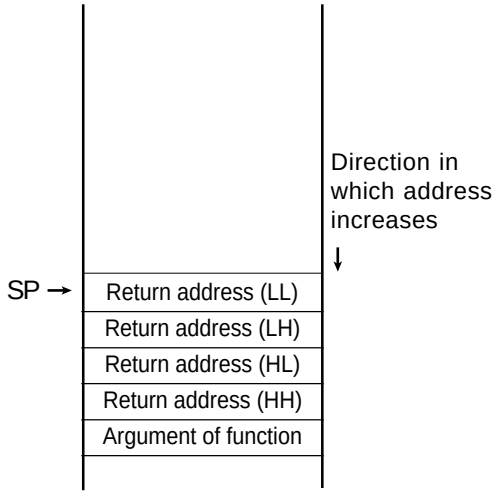
SP ← SP - 2
M(SP)*1 ← FBH
SP ← SP - 2
M(SP) ← FBL
FB ← SP
SP ← SP - *src*

*1 The 8 high-order bits become undefined.

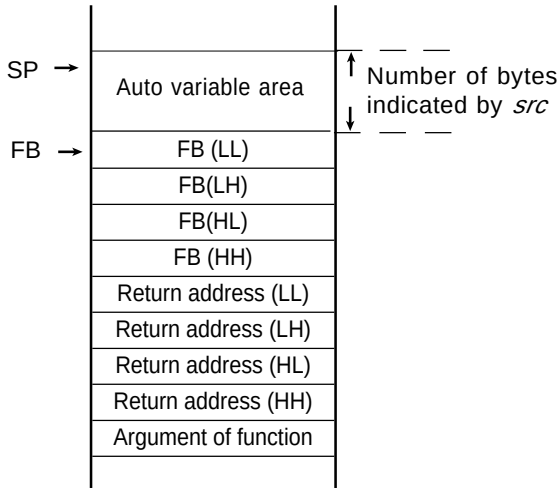
[Function]

- This instruction generates a stack frame. *src* represents the size of the stack frame. Set an even number for *src*. (You can set odd number, but it is more effective to set even number for operation.)
- The diagrams below show the stack area status before and after the ENTER instruction is executed at the beginning of a called subroutine.

Before instruction execution



After instruction execution



[Selectable src]

src
#IMM8

[Flag Change]

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

[Description Example]

```
ENTER   #4
```

EXITD

Deallocate stack frame

EXIT and Deallocate stack frame

EXITD

[Syntax]

EXITD

[Instruction Code/Number of Cycles]

Page=219

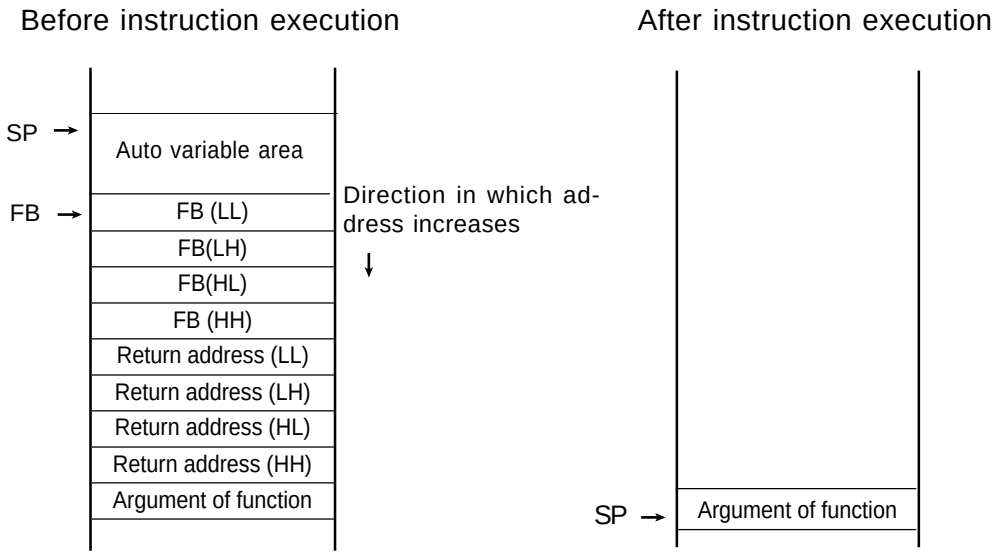
[Operation]

SP ← FB
FB_L ← M(SP)
SP ← SP + 2
FB_H ← M(SP)
SP ← SP + 2
PC_L ← M(SP)
SP ← SP + 2
PC_H ← M(SP)*1
SP ← SP + 2

*1 The 8 high-order bits become undefined.

[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

EXTS

Extend sign

EXTend Sign

EXTS

[Syntax]

EXTS.size dest

└ **B , W**

EXTS.size src,dest

└ **B**

[Instruction Code/Number of Cycles]

Page= 220

[Operation]

dest ← EXTS(dest)

dest ← EXTS(src)

[Function]

- This instruction sign extends *dest* and stores the result in *dest*.
- When you selected (.B) for the size specifier (.size), *src* or *dest* is sign extended to 16 bits. When dest is the address register(A0, A1), the 8 high-order bits become 0.
- When you selected (.W) for the size specifier (.size), *dest* is sign extended to 32 bits. When R0 is selected for *dest*, R2 is used for the upper byte; when R1 is selected, R3 is used for the upper byte. When dest is the address register, stores the 24 low-order bits of result in dest.

[Selectable src/dest]

dest[*1]			
R0L/R0~~/R2R0~~		~~R0H/R2/-~~	
R1L/R1~~/R3R1~~		~~R1H/R3/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

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

src[*2]				**dest**[*2]			
R0L~~/R0/R2R0~~		R0H~~/R2/-~~		~~R0L/~~R0~~/R2R0~~		~~R0H/~~R2~~/-~~	
R1L~~/R1/R3R1~~		R1H~~/R3/-~~		~~R1L/~~R1~~/R3R1~~		~~R1H/~~R3~~/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	[A0]	[A1]	~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16

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

```
EXTS.B  R0L
EXTS.W  R0
EXTS.W  [A0]
```

EXTZ

Extend zero
EXTend Zero

EXTZ

[Syntax]

EXTZ **src,dest**

[Instruction Code/Number of Cycles]

Page= 222

[Operation]

dest ← EXTZ(src)

[Function]

- This instruction zero-extends *src* to 16 bits and stores the result in *dest*. When dest is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable src/dest]

src				dest			
R0L/~~R0/R2R0~~		R0H/~~R2/-~~		~~R0L~~/R0/~~R2R0~~		~~R0H~~/R2/~~-~~	
R1L/~~R1/R3R1~~		R1H/~~R3/-~~		~~R1L~~/R1/~~R3R1~~		~~R1H~~/R3/~~-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	[A0]	[A1]	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
~~#IMM~~							

[Flag Change]

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

Conditions

S : The flag is always cleared to 0.

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

[Description Example]

```
EXTZ    R0L,R2
EXTZ    [A1],[A0]
```

FCLR

Clear flag register bit

Flag register CLeaR

FCLR

[Syntax]

FCLR **dest**

[Instruction Code/Number of Cycles]

Page= 223

[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
```

FREIT

Fast return from Interrupt

Fast REturn from InTerrupt

FREIT

[Syntax]

FREIT

[Instruction Code/Number of Cycles]

Page= 223

[Operation]

FLG ← SVF

PC ← SVP

[Function]

- Restores the contents of PC and FLG from the high-speed interrupt registers that had been saved when accepting a high-speed interrupt request upon returning 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 Becomes the content of SVF.

[Description Example]

FREIT

FSET

Set flag register bit

Flag register SET

FSET

[Syntax]

FSET **dest**

[Instruction Code/Number of Cycles]

Page=224

[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
```

INC

Increment
INCrement

INC

[Syntax]

INC.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 225

[Operation]

dest ← dest + 1 [dest] ← [dest] + 1

[Function]

- This instruction adds 1 to *dest* and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
INC.B   [[A1]]
```

INDEX*Type*

Index
INDEX Type

INDEX*Type*

[Syntax]

INDEX*Type*.size src

B , W

[Instruction Code/Number of Cycles]

Page= 225

[Operation]

[Function]

- This instruction modifies addressing of the next instruction.
- No interrupts are enabled until after the modifying instruction is executed.
- Use this instruction to access arrays.
- For details, refer to Section 3.3, "Index Instructions."
- There are following types for *Type:*

Type	Function
B BD BS	Modifies the addressing of the next instruction in units of bytes.
W WD WS	Modifies the addressing of the next instruction in units of words.
L LD LS	Modifies the addressing of the next instruction in units of long words.

[Selectable src]

src			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
A0/A0~~/A0~~	A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

[Flag Change]

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

[Description Example]

```
INDEXB.W    R0
INDEXLS.B   [A0]
```

INT

Interrupt by INT instruction

INTerrupt

INT

[Syntax]

INT **src**

[Instruction Code/Number of Cycles]

Page= 230

[Operation]

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*1 ← (PC + 2)H
SP ← SP - 2
M(SP) ← (PC + 2)L
PC ← M(IntBase + src × 4)

*1 The 8 high-order bits become undefined.

[Function]

- This instruction generates a software interrupt specified by *src*. *src* represents a software interrupt number.
- When *src* is 31 or smaller, the U flag is cleared to 0 and the interrupt stack pointer (ISP) is used.
- When *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.
- The interrupt number of *src* is 0 ≤ *src* ≤ 63.

[Selectable src]

src
#IMM6*1*2

*1 #IMM denotes a software interrupt number.
*2 The range of values is 0 ≤ #IMM6 ≤ 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 when the software interrupt number is 31 or smaller. The flag does not change when the software interrupt number is 32 or larger.
I : The flag is cleared.
D : The flag is cleared.

[Description Example]

```
INT     #0
```

INTO

Interrupt on overflow

INTerrupt on Overflow

INTO

[Syntax]

INTO

[Instruction Code/Number of Cycles]

Page= 230

[Operation]

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*1 ← (PC + 1)H
SP ← SP - 2
M(SP) ← (PC + 1)L
PC ← M(FFFFE016)

*1 The 8 high-order bits become undefined.

[Function]

- When the O flag is 1, this instruction generates an overflow interrupt. When 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

JCnd

Jump on condition

Jump on Condition

JCnd

[Syntax]

J*Cnd* **label**

[Instruction Code/Number of Cycles]

Page= 231

[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. When the condition indicated by *Cnd* is true, control jumps to **label**. When 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∧$\overline{Z}$=1	Greater than	<	LEU	C∧$\overline{Z}$=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, 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
```

JMP

Unconditional jump
JuMP

JMP

[Syntax]

JMP(.length) label

S , B , W , A

[Instruction Code/Number of Cycles]

Page=231

[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	abs24

*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
```

JMPI

Jump indirect
JuMP Indirect

JMPI

[Syntax]

JMPI.length src

W , A

[Instruction Code/Number of Cycles]

Page=233

[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*. When *src* is memory, specify the address at which the low-order address is stored.
- When 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). When *src* is memory, the required memory capacity is 2 bytes.
- When *src* is memory and (.A) is selected for the jump distance specifier (.length), the required memory capacity is 3 bytes.

[Selectable src]

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

src			
~~R0L/~~R0~~/R2R0~~		~~R0H/~~R2~~/-~~	
~~R1L/~~R1~~/R3R1~~		~~R1H/~~R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

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

src			
~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/~~A0	~~A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

[Flag Change]

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

[Description Example]

```
JMPI.A    A1
JMPI.W    R0
```

JMPS

Jump to special page
JuMP Special page

JMPS

[Syntax]

JMPS **src**

[Instruction Code/Number of Cycles]

Page= 234

[Operation]

$PC_H \leftarrow FF_{16}$

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

[Function]

- This instruction causes control to jump to the address set in each table of the special page vector table plus $FF0000_{16}$. The area across which control can jump is from address $FF0000_{16}$ to address $FFFFFF_{16}$.
- The special page vector table is allocated to an area from address $FFFE00_{16}$ to address $FFFFDB_{16}$.
- *src* represents a special page number. The special page number is 255 for address $FFFE00_{16}$, and 18 for address $FFFFDA_{16}$.

[Selectable src]

src
#IMM8*1*2

*1 #IMM denotes a special page number.

*2 The range of values is 18 ≤ #IMM8 ≤ 255.

[Flag Change]

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

[Description Example]

```
JMPS    #20
```

JSR

Subroutine call
Jump SubRoutine

JSR

[Syntax]

JSR(.length) **label**

W , A

[Instruction Code/Number of Cycles]

Page= 235

[Operation]

SP ← SP - 2
M(SP)*1 ← (PC + n*2)H
SP ← SP - 2
M(SP) ← (PC + n*2)ML
PC ← label

*1 The 8 high-oreder bits become 0.
*2 n denotes the number of bytes in the instruction.

[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	abs24

*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
```

JSRI

Indirect subroutine call

Jump SubRoutine Indirect

JSRI

[Syntax]

JSRI.length src

W , A

[Instruction Code/Number of Cycles]

Page=236

[Operation]

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

SP ← SP - 2
M(SP)*1 ← (PC + n*2)H
SP ← SP - 2
M(SP) ← (PC + n*2)ML
PC ← PC ± *src*

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

SP ← SP - 2
M(SP)*1 ← (PC + n*2)H
SP ← SP - 2
M(SP) ← (PC + n*2)H
PC ← *src*

*1 The 8 high-oreder bits become 0.
*2 n denotes the number of bytes in the instruction.

[Function]

- This instruction causes control to jump to a subroutine at the address indicated by *src*. When *src* is memory, specify the address at which the low-order address is stored.
- When 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). When *src* is memory, the required memory capacity is 2 bytes.
- When *src* is memory and (.A) is selected for the jump distance specifier (.length), the required memory capacity is 3 bytes.

[Selectable src]

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

src			
~~R0L/~~R0~~/R2R0~~		~~R0H/~~R2~~/-~~	
~~R1L/~~R1~~/R3R1~~		~~R1H/~~R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

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

src			
~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/~~A0	~~A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

[Flag Change]

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

[Description Example]

```
JSRI.A  A1
JSRI.W  R0
```

JSRS

Special page subroutine call

Jump SubRoutine Special page

JSRS

[Syntax]

JSRS **src**

[Instruction Code/Number of Cycles]

Page= 237

[Operation]

SP ← SP - 2
M(SP)*1 ← (PC + 2)H
SP ← SP - 2
M(SP) ← (PC + 2)ML
PCH ← FF16
PCML ← M (FFFE16 - *src* × 2)

*1 The 8 high-oreder bits become 0.

[Function]

- This instruction causes control to jump to a subroutine at the address set in each table of the special page vector table plus FF000016. The area across which program flow can jump to a subroutine is from address FF000016 to address FFFFFF16.
- The special page vector table is allocated to an area from address FFFE0016 to address FFFFDB16.
- *src* represents a special page number. The special page number is 255 for address FFFE0016, and 18 for address FFFFDA16.

[Selectable src]

src
#IMM8*1*2

*1 #IMM denotes a special page number.

*2 The range of values is 18 ≤ #IMM8 ≤ 255.

[Flag Change]

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

[Description Example]

JSRS #18

LDC

Transfer to control register

LoaD Control register

LDC

[Syntax]

LDC **src,dest**

[Instruction Code/Number of Cycles]

Page= 237

[Operation]

dest ← src

[Function]

- This instruction transfers *src* to the control register indicated by *dest*.
- When memory is specified for *src*, the following bytes of memory are required.

  2 bytes : DMD0*1, DMD1*1, FLG, DCT0, DCT1, DRC0, DRC1, SVF

  4 bytes*2 : FB, SB, SP*3, ISP*3, INTB*3, VCT, SVP, DMA0, DMA1, DRA0, DRA1, DSA0, DSA1

*1 The low-order 8 bit of src is transferred.

*2 The low-order 24 bit of src is transferred.

*3 Set even number for SP, ISP and INTB even though odd number can be set. It is more effective to set even number for operation.

[Selectable src/dest]

src				dest			
~~R0L/~~R0/~~R2R0~~		~~R0H/~~R2/~~-~~		DMD0	DMD1	DCT0	DCT1
~~R1L/~~R1/~~R3R1~~		~~R1H/~~R3/~~-~~		DRC0	DRC1	FLG	SVF
~~A0/~~A0/A0	~~A1/~~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]	dsp:16[FB]				
dsp:24[A0]	dsp:24[A1]	abs24	abs16				
#IMM16/#IMM24							
~~R0L/R0/~~R2R0		~~R0H/R2/~~-		FB	SB	SP*4	ISP
~~R1L/R1/~~R3R1		~~R1H/R3/~~-		INTB	VCT	SVP	
~~A0/A0/~~/A0	~~A1/A1/~~A1	[A0]	[A1]	DMA0	DMA1	DRA0	DRA1
dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]	DSA0	DSA1		
dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]				
dsp:24[A0]	dsp:24[A1]	abs24	abs16				
#IMM16/#IMM24							

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

[Flag Change]

Flag	U	I	O	B	S	Z	D	C
Change	*5	*5	*5	*5	*5	*5	*5	*5

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

[Description Example]

```
LDC     A0,FB
```

LDCTX

Restore context
LoaD ConTeXt

LDCTX

[Syntax]

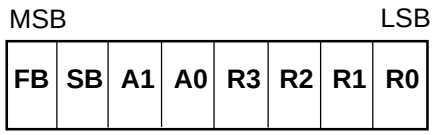
LDCTX **abs16,abs24**

[Instruction Code/Number of Cycles]

Page=240

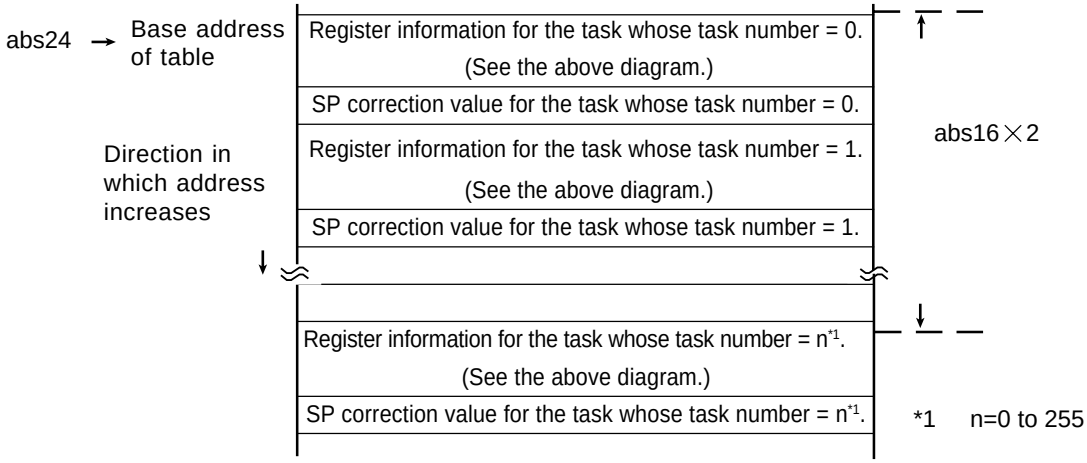
[Function]

- 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 abs24.
- 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. Calculated as 2 bytes when transferring the R0, R1, R2, or R3 registers. A0, A1, SB, and FB are calculated as 4 bytes.
- 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.



← Transferred sequentially beginning with R0

- The table data is comprised as shown below. The address indicated by abs24 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.



[Flag Change]

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

[Description Example]

```
LDCTX   Ram,Rom_TBL
```

LDIPL

Set interrupt enable level

LoaD Interrupt Permission Level

LDIPL

[Syntax]

LDIPL **src**

[Instruction Code/Number of Cycles]

Page= 241

[Operation]

IPL ← src

[Function]

- This instruction transfers *src* to IPL.

[Selectable src]

src
#IMM3*1

*1 The range of values is 0 ≤ #IMM3 ≤ 7.

[Flag Change]

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

[Description Example]

```
LDIPL   #2
```

MAX

Select maximum value

MAX select

MAX

[Syntax]

MAX.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 241

[Operation]

if (src > dest)

then dest ← src

[Function]

- Singed compares *src* and *dest* and transfers *src* to *dest* when *src* is greater than *dest*. No change occurs when *src* is smaller than or equal to *dest*.
- When (.W) is specified for the size specifier (.size), *dest* is the address register and writing to *dest*, the 8 high-order bits of the operation result are become 0. Also, when *src* is the address register, transfers the 16 low-order bits of the address register to *dest*.

[Selectable src/dest]

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0/~~A0/~~A0~~	~~A1/~~A1/~~A1~~	[A0]	[A1]	~~A0/~~A0/~~A0~~	~~A1/~~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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[Flag Change]

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

[Description Example]

```
MAX.B   #0ABH,R0L
MAX.W   #-1,R2
```

MIN

Select minimum value

MIN select

MIN

[Syntax]

MIN.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=243

[Operation]

if (src < dest)

then dest ← src

[Function]

- Signed compares *src* and *dest* and transfers *src* to *dest* when *src* is smaller than *dest*. No change occurs when *src* is greater than or equal to *dest*.
- When (.W) is specified for the size specifier (.size), *dest* is the address register and writing to *dest*, the 8 high-order bits of the operation result are become 0. Also, when *src* is the address register, transfers the 16 low-order bits of the address register to *dest*.

[Selectable src/dest]

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0~~/A0/~~A0~~	~~A1~~/A1/~~A1~~	[A0]	[A1]	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

[Flag Change]

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

[Description Example]

```
MIN.B   #0ABH,R0L
MIN.W   #-1,R2
```

MOV

Transfer
MOVe

MOV

[Syntax]

MOV.size (:format) src,dest

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

[Instruction Code/Number of Cycles]

Page=245

[Operation]

dest ← src　　[dest] ← src
dest ← [src]　　[dest] ← [src]

[Function]

- This instruction transfers *src* to *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-order bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.
- When (.L) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits of *src* is ignored and the 24 low-order bits of *src* is stored to *dest*. Also, when *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The flags also change states depending on the result of 32-bit operation.

[Selectable src/dest]*1

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

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM	dsp:8[SP]*3			dsp:8[SP]*3			

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, dsp:8[SP], and #IMM.
*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/or A1 for *src* and *dest* simultaneously.
*3 When *src* or *dest* is dsp:8[SP], you cannot choose indirect instruction addressing [src] nor [dest].

[Flag Change]

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

Conditions
S : The flag is set when the transfer resulted in MSB = 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
MOV.W    [A1],[[A0]]
```

[src/dest Classified by Format]

G format *4

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*5	A1/A1/A1*5	[A0]	[A1]	A0/A0/A0*5	A1/A1/A1*5	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32		dsp:8[SP]*6*8		dsp:8[SP]*6*7*8			

*4 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, dsp:8[SP], and #IMM.

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

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

*7 When you specify (.B) or (.W) for the size specifier (.size) and *src* is not #IMM, you can choose dsp:8 [SP] for *dest*.

*8 When *src* or *dest* is dsp:8[SP], you cannot choose indirect instruction addressing [src] nor [dest].

Q format *9*10

src				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0/~~A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4*11							

*9 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/~~R2R0~~, R0H/R2/~~-~~, R1L/R1/~~R3R1~~, R1H/R3/~~-~~, and #IMM .

*10 You can only specify (.B) or (.W) for the size specifier (.size).

*11 The range of values is - 8 ≤ #IMM4 ≤ +7.

S format *12

src			dest			
R0L/R0*13*14	dsp:8[SB]*14	dsp:8[FB]*14 abs16*14	R0L/R0*13*14	R1L/R1*14*15	dsp:8[SB]*14	dsp:8[FB]*14
#IMM8/#IMM16*14			abs16*14	~~A0~~	~~A1~~	
~~R0L/R0~~	dsp:8[SB]*17	dsp:8[FB]*17 abs16*17	~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~
#IMM16*16/#IMM24*17			~~abs16~~	~~A0/~~A0*16/A0*17	~~A1/~~A1*16/A1*17	

*12 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0, R1L/R1, and #IMM.

*13 You cannot choose the same registers for *src* and *dest* simultaneously.

*14 You can only specify (.B) or (.W) for the size specifier (.size).

*15 When *src* is not #IMM8/IMM16, you can only choose R1L/R1 for *dest* .

*16 You can specify (.W) for the size specifier (.size). In this case, you cannot use indirect instruction addressing mode for *dest*.

*17 You can specify (.L) for the size specifier (.size). In this case, you cannot use indirect instruction addressing mode for *dest*.

Z format *18

src				dest			
~~R0L~~	~~R0H~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
~~abs16~~	#0			~~A0~~	~~A1~~		

*18 You can specify (.B) or (.W) for the size specifier (.size).

MOVA

Transfer effective address

MOVe effective Address

MOVA

[Syntax]

MOVA **src,dest**

[Instruction Code/Number of Cycles]

Page= 254

[Operation]

dest ← EVA(src)

[Function]

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

[Selectable src/dest]

src				dest			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0/~~A0	~~A1/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]	dsp:16[FB]	~~dsp:16[A0]~~	~~dsp:16[A1]~~	~~dsp:16[SB]~~	~~dsp:16[FB]~~
dsp:24[A0]	dsp:24[A1]	abs24	abs16	~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~
~~#IMM~~							

[Flag Change]

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

[Description Example]

MOVA Ram:16[SB],A0

MOV*Dir*

Transfer 4-bit data
MOVe nibble

MOV*Dir*

[Syntax]

MOV*Dir* **src,dest**

[Instruction Code/Number of Cycles]

Page= 255

[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*(8 bits)'s 4 high-order bits to *dest*(8 bits)'s 4 high-order bits.
HL	Transfers *src*(8 bits)'s 4 high-order bits to *dest*(8 bits)'s 4 low-order bits.
LH	Transfers *src*(8 bits)'s 4 low-order bits to *dest*(8 bits)'s 4 high-order bits.
LL	Transfers *src*(8 bits)'s 4 low-order bits to *dest*(8 bits)'s 4 low-order bits.

[Selectable src/dest]

src				**dest**			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/A1/A1	[A0]	[A1]	A0/A0/A0	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM							
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0	A1/A1/A1	[A0]	[A1]	A0/A0/A0	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM							

[Flag Change]

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

[Description Example]

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

MOVX

Transfer extend sign
MOVe eXtend sign

MOVX

[Syntax]

MOVX **src,dest**

[Instruction Code/Number of Cycles]

Page= 257

[Operation]

dest/[dest] ← EXTS(src)

[Function]

- Sign-extends the 8-bit immediate to 32 bits before transferring it to *dest*.
- When *dest* is the address register (A0, A1), the 24 low-order bits are transferred. The flags also change state for the 32 bits transfers to be performed.

[Selectable src/dest]

src				**dest**[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0/~~A0	~~A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8*[2]							

*1 Indirect instruction addressing [dest] can be used in all addressing except ~~R0L/R0/~~R2R0, ~~R1L/R1/~~R3R1, and #IMM.

*2 The range of values is -128 ≤ #IMM8 ≤ +127

[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]

```
MOVX    #10,A0
MOVX    #5,[[A1]]
```

MUL

Signed multiply
MULtiple

MUL

[Syntax]

MUL.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page= 257

[Operation]

dest ← dest × src　　[dest] ← [dest] × src

dest ← dest × [src]　　[dest] ← [dest] × [src]

[Function]

- This instruction multiplies *src* and *dest* together including the sign bits and stores the result in *dest*.
- When you selected (.B) for the size specifier (.size), *src* and *dest* both are treated as 8-bit data for the operation and the result is stored in 16 bits. When you specified an address register (A0, A1) for either *src* or *dest*, operation is performed on the address register's 8 low-order bits. When *dest* is the address register, the 8 high-order bits become 0.
- When you selected (.W) for the size specifier (.size), *src* and *dest* both are treated as 16-bit data for the operation and the result is stored in 32 bits. R0, R1, A0, and A1 can be selected for *dest*. When you specified R0 or R1 for *dest*, the result is stored in R2R0 or R3R1 accordingly. When the address register is selected for *dest*, the 24 low-order bits of the 32-bit operation result is stored. When the address register is selected for *src*, operation is performed using the 16 low-order bits of the register.
- When you selected (.L) for the size specifier (.size), *src* and *dest* both are treated as 32-bit data for the operation and the result is stored in 32 bits. R2R0 is selected for *dest*. When the address register is selected for *src*, *src* is zero-extended to be treated as 32-bit data for the operation, and only 32-low-order bits are valid for the result.

[Selectable src/dest]*1

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32							

*1 When the size specifier (.size) is (.B), indirect instruction addressing [src] and [dest] can be used in all addressing except R0L, R0H, R1L, R1H, and #IMM.
When the size specifier (.size) is (.W), indirect instruction addressing [src] can be used in all addressing except R0, R1, and #IMM. Indirect instruction addressing [dest] cannot be used in any addressing.
When the size specifier (.size) is (.L), no indirect instruction addressing can be used.

*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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.L   A0,R2R0
```

MULEX

Multipl extend sign
MULtiple EXtend

MULEX

[Syntax]

MULEX **src**

[Instruction Code/Number of Cycles]

Page= 260

[Operation]

R1R2R0 ← R2R0 × src/[src]

[Function]

- Multiplies *src* (16-bit data) and R2R0 including the sign and stores the result in R1R2R0.

[Selectable src]

src[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/~~R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [src] can be used in all addressing except ~~R1H~~/R3/~~-~~.

[Flag Change]

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

[Description Example]

```
MULEX   A0
MULEX   R3
MULEX   Ram
MULEX   [[A0]]
```

MULU

Unsigned multiply
MULtiple Unsigned

MULU

[Syntax]

MULU.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=260

[Operation]

dest ← dest × src　　[dest] ← [dest] × src

dest ← dest × [src]　　[dest] ← [dest] × [src]

[Function]

- This instruction multiplies *src* and *dest* together not including the sign bits and stores the result in *dest*.
- When you selected (.B) for the size specifier (.size), *src* and *dest* both are treated as 8-bit data for the operation and the result is stored in 16 bits. When you specified an address register (A0, A1) for either *src* or *dest*, operation is performed on the address register's 8 low-order bits. When *dest* is the address register, the 8 high-order bits become 0.
- When you selected (.W) for the size specifier (.size), *src* and *dest* both are treated as 16-bit data for the operation and the result is stored in 32 bits. R0, R1, A0, and A1 can be selected for *dest*. When you specified R0 or R1 for *dest*, the result is stored in R2R0 or R3R1 accordingly. When the address register is selected for *dest*, the 24 low-order bits of the 32-bit operation result is stored. When the address register is selected for *src*, operation is performed using the 16 low-order bits of the register.
- When you selected (.L) for the size specifier (.size), *src* and *dest* both are treated as 32-bit data for the operation and the result is stored in 32 bits. R2R0 is selected for *dest*. When the address register is selected for *src*, *src* is zero-extended to be treated as 32-bit data for the operation, and only 32-low-order bits are valid for the result.

[Selectable src/dest] *1

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		~~R0H/R2/-~~	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/~~R3R1~~		~~R1H/R3/-~~	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/~~A0~~*2	A1/A1/~~A1~~*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/~~#IMM32~~							

*1 When the size specifier (.size) is (.B), indirect instruction addressing [src] and [dest] can be used in all addressing except R0L, R0H, R1L, R1H, and #IMM.
When the size specifier (.size) is (.W), indirect instruction addressing [src] can be used in all addressing except R0, R1, and #IMM. Indirect instruction addressing [dest] cannot be used in any addressing.
When the size specifier (.size) is (.L), no indirect instruction addressing can be used.

*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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
MUL.L     A0,R2F0
```

NEG

Two's complement
NEGate

NEG

[Syntax]

NEG.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 263

[Operation]

dest ← 0 - dest [dest] ← 0 - [dest]

[Function]

- This instruction takes the 2's complement of *dest* and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
NEG.W   [[A0]]
```

NOP

No operation

No OPeration

NOP

[Syntax]

NOP

[Instruction Code/Number of Cycles]

Page= 263

[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 dest

B , W

[Instruction Code/Number of Cycles]

Page= 264

[Operation]

dest ← $\overline{\text{dest}}$ [dest] ← [$\overline{\text{dest}}$]

[Function]

- This instruction inverts *dest* and stores the result in *dest*.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
```

OR

Logically OR
OR

OR

[Syntax]

OR.size (:format) src,dest

G , S (Can be specified)
B , W

[Instruction Code/Number of Cycles]

Page= 264

[Operation]

dest ← src ∨ dest　　[dest] ← src ∨ [dest]
dest ← [src] ∨ dest　　[dest] ← [src] ∨ [dest]

[Function]

- This instruction logically ORs *dest* and *src* together and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-order bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]*1

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

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

*2 If you specify (.B) for the size specifier (.size), you cannot choose A0 and/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
OR.W:G    [R1],[[A0]]
```

[src/dest Classified by Format]

G format*3

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*4	A1/A1/A1*4	[A0]	[A1]	A0/A0/A0*4	A1/A1/A1*4	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*3 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

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

S format*5

src				dest			
R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16							

*5 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0, and #IMM.

POP

Restore register/memory

POP

POP

[Syntax]

POP.size dest

B , W

[Instruction Code/Number of Cycles]

Page= 267

[Operation]

dest/[dest] ← M(SP)

SP ← SP + 2

*1 Even when (.B) is specified for the size specifier (.size), SP is increased by 2.

[Function]

- This instruction restores *dest* from the stack area.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*2			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*2 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[Flag Change]

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

[Description Example]

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

POPC

Restore control register

POP Control register

POPC

[Syntax]

POPC **dest**

[Instruction Code/Number of Cycles]

Page= 267

[Operation]

- When dest is DCT0, DCT1, DMD0, DMD1, DRC0, DRC1, SVF or FLG
  dest*1 ← M(SP)
  SP ← SP + 2

*1 The 8 low-order bytes are saved when dest is DMD0 or DMD1.

- When dest is FB, SB, SP, ISP or INTB
  dest*2 ← M(SP)
  SP*3 ← SP + 4

*2 The 3 low-order byte are saved.

*3 4 is not added to SP when dest is SP, or dest is ISP while U flag is "0".

[Function]

- This instruction restores from the stack area to the control register indicated by *dest*.
- Restored stack area is indicated by the U flag.

[Selectable dest]

dest			
FB	SB	SP*1	ISP
INTB			
DCT0	DCT1	DMD0	DMD1
DRC0	DRC1	SVF	FLG

*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]

```
POPC    SB
```

POPM

Restore multiple registers
POP Multiple

POPM

[Syntax]

POPM **dest**

[Instruction Code/Number of Cycles]

Page= 268

[Operation]

dest*3 ← M(SP)
SP ← SP + n1*1 × 2
SP ← SP + n2*2 × 4

*1 n1 denotes the number of R0, R1, R2 and R3 registers to be restored.
*2 n2 denotes the number of A0, A1, SB and FB registers to be restored.
*3 The 3 low-order bytes are saved when dest is A0, A1, SB and FB.

[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*3							
R0	R1	R2	R3	A0	A1	SB	FB

*3 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
```

PUSH

Save register/memory/immediate data

PUSH

PUSH

[Syntax]

PUSH.size src

B , W , L

[Instruction Code/Number of Cycles]

Page= 269

[Operation]

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

SP ← SP - 2

M(SP)*1 ← src/[src]

*1 The 8 high-order bits become undefined.

Even when (.B) is specified for the size specifier (.size) , SP is decreased by 2.

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

SP ← SP - 2

M(SP) ← src/[src]

• When the size specifier (.size) is (.L)

SP ← SP - 4

M(SP)*2 ← src/[src]

*2 When *src* is address register(A0, A1), the 8 high-order bits become 0.

[Function]

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

• When (.W) is specified for the size specifier (.size) and *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src]

src*3			
R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-	
~~A0/~~A0/A0	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32			

*3 Indirect instruction addressing [src] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

[Flag Change]

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

[Description Example]

```
PUSH.B   #5
PUSH.W   #100H
PUSH.L   R2R0
```

PUSHA

Save effective address
PUSH effective Address

PUSHA

[Syntax]

PUSHA **src**

[Instruction Code/Number of Cycles]

Page= 271

[Operation]

SP ← SP - 4

M(SP)*1 ← EVA(src)

*1 The 8 high-order bits become undefined.

[Function]

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

[Selectable src]

src			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

[Flag Change]

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

[Description Example]

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

PUSHC

Save control register
PUSH Control register

PUSHC

[Syntax]

PUSHC **src**

[Instruction Code/Number of Cycles]

Page= 271

[Operation]

- When *src* is DCT0, DCT1, DMD0, DMD1, DRC0, DRC1, SVF or FLG

SP ← SP - 2
M(SP)*1 ← src

*1 When *src* is DMD0 or DMD1, the 8 high-order bits become undefined.

- When *src* is FB, SB, SP, ISP or INTB

SP ← SP - 4
M(SP)*2 ← src*3

*2 The 8 high-order bits become 0.

*3 SP before 4 is subtracted is saved when src is SP, or src is ISP while U flag is "0".

[Function]

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

[Selectable src]

src			
FB	SB	SP*3	ISP
INTB			
DCT0	DCT1	DMD0	DMD1
DRC0	DRC1	SVF	FLG

*3 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
```

PUSHM

Save multiple registers
PUSH Multiple

PUSHM

[Syntax]

PUSHM **src**

[Instruction Code/Number of Cycles]

Page= 272

[Operation]

SP ← SP – n1*1 × 2
SP ← SP – n2*1 × 4
M(SP)*3 ← src

*1 n1 denotes the number of R0, R1, R2 and R3 registers to be saved.

*2 n2 denotes the number of A0, A1, SB and FB registers to be saved.

*3 When *src* is A0, A1, SB or FB, the 8 high-order bits become 0.

[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*4
R0 R1 R2 R3 A0 A1 SB FB

*4 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
```

REIT

Return from interrupt

REturn from InTerrupt

REIT

[Syntax]

REIT

[Instruction Code/Number of Cycles]

Page= 273

[Operation]

PCML ← M(SP)
SP ← SP + 2
PCH ← M(SP)*1
SP ← SP + 2
FLG ← M(SP)
SP ← SP + 2

*1 The 8 high-order bits are saved.

[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 Becomes the value in the stack.

[Description Example]

REIT

RMPA

Calculate sum-of-products

Repeat MultiPle & Addition

RMPA

[Syntax]

RMPA.size

B , W

[Instruction Code/Number of Cycles]

Page= 273

[Operation][*1]

Repeat

R1R2R0 ← R1R2R0 + M(A0) × M(A1)

A0 ← A0 + 2 (1)[*2]

A1 ← A1 + 2 (1)[*2]

R3 ← R3 - 1

Until R3 = 0

*1 When 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 R1R2R0 .
- The content of the address register when the instruction is completed indicates the next address of the last-read data.
- When 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 R1R2R0 has the initial value set.

[Fl ag Change]

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

Conditions

O : The flag is set when $+2^{31}-1$ or -2^{31} is exceeded during operation; otherwise cleared.

[Description Example]

RMPA.B

ROLC

Rotate left with carry

ROtate to Left with Carry

ROLC

[Syntax]

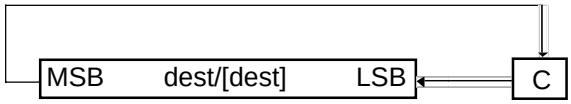
ROLC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=274

[Operation]



[Function]

- This instruction rotates *dest* one bit to the left including the C flag.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
ROLC.W  [[A0]]
```

RORC

Rotate right with carry

ROtate to Right with Carry

RORC

[Syntax]

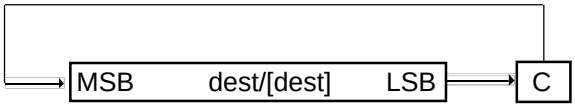
RORC.size dest

B , W

[Instruction Code/Number of Cycles]

Page=274

[Operation]



[Function]

- This instruction rotates *dest* one bit to the right including the C flag.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), the 8 high-order bits become 0.

[Selectable dest]

dest*1			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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
RORC.W  [[A0]]
```

ROT

Rotate
ROTate

ROT

[Syntax]

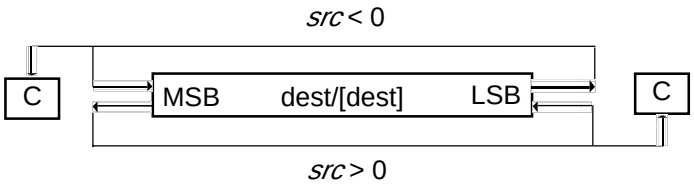
ROT.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page=275

[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*. When *src* is positive, bits are rotated left; when negative, bits are rotated right.
- When *src* is an immediate, the number of rotates is - 8 to +8(≠0). You cannot set values less than - 8, equal to 0, or greater than +8.
- When *src* is a register, the number of rotates is -16 to +16. Although you can set 0, no bits are rotated and no flags are changed. When you set a value less than -17 or greater than +17, the result of rotation is undefined.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register(A0, A1), the 8 high-order bits become 0.

[Selectable src/dest]

src				**dest**[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
~~R1L/R1/R3R1~~		R1H~~/R3/-~~		R1L/R1~~/R3R1~~[*2]		R1H/R3~~/-~~[*2]	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0~~/A0~~	~~A1/~~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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4[*3]							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

*2 When *src* is R1H, you cannot choose R1 or R1H for *dest*.

*3 The range of values is - 8 ≤ #IMM4 ≤ +8. However, you cannot set 0.

[Flag Change]

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

*4 When 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
```

RTS

Return from subroutine

ReTurn from Subroutine

RTS

[Syntax]

RTS

[Instruction Code/Number of Cycles]

Page=276

[Operation]

PCML ← M(SP)
SP ← SP + 2
PCH ← M(SP)*1
SP ← SP + 2

*1 The 8 low-order bits are saved.

[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= 277

[Operation]

dest ← dest - src - $\overline{C}$

[Function]

- This instruction subtracts *src* and inverted C flag from *dest* and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*1	A1/A1/A1*1	[A0]	[A1]	A0/A0/A0*1	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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]

```
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.
```

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= 279

[Operation]

dest ← dest - src

if dest ≠ 0 **then** jump label

[Function]

- This instruction subtracts *src* from *dest* and stores the result in *dest*.
- When the operation resulted in any value other than 0, control jumps to **label**. When the operation resulted in 0, the next instruction is executed.
- The op-code of this instruction is the same as that of ADJNZ.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable src/dest/label]

src	dest				label
#IMM4*1	R0L/R0/~~R2R0~~		R0H/R2/-		PC*2-126 ≤ label ≤ PC*2+129
	R1L/R1/~~R3R1~~		R1H/R3/-		
	~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]	
	dsp:24[A0]	dsp:24[A1]	abs24	abs16	

*1 The range of values is -7 ≤ #IMM4 ≤ +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
SBJNZ.W   #2,[A1],label
```

SC*Cnd*

Store on condition

Store Condition on Condition

SC*Cnd*

[Syntax]

SC*Cnd* **label**

[Instruction Code/Number of Cycles]

Page=280

[Operation]

if true then dest ← 1
else dest ← 0

if true then [dest] ← 1
else [dest] ← 0

[Function]

- When the condition specified by *Cnd* is true, this instruction stores a 1 in *dest*; when the condition is false, it stores a 0 in *dest*.
- When *dest* is the address register(A0, A1), the 8 high-order bits of the address register become 0.
- There are following types 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∧$\overline{Z}$=1	Greater than	<	LEU	C∧$\overline{Z}$=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*1			
~~R0L~~/R0/~~R2R0~~		~~R0H~~/R2/~~-~~	
~~R1L~~/R1/~~R3R1~~		~~R1H~~/R3/~~-~~	
~~A0~~/A0/~~A0~~	~~A1~~/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]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 Indirect instruction addressing [dest] can be used in all addressing except ~~R0L~~/R0/~~R2R0~~, ~~R0H~~/R2/~~-~~, ~~R1L~~/R1/~~R3R1~~, and ~~R1H~~/R3/~~-~~.

[Flag Change]

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

[Description Example]

```
SCC     R0L
SCC     [dsp:8[A0]]
```

SCMPU

String compare unequal

String CoMPare Unequal

SCMPU

[Syntax]

SCMPU.size

B , W

[Instruction Code/Number of Cycles]

Page=281

[Operation]

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

Repeat

M(A0) – M(A1) (compared by byte)
tmp0 ← M(A0)
tmp2 ← M(A1)
A0 ← A0 + 1
A1 ← A1 + 1

Until (tmp0=0) ıı (tmp0≠tmp2)

tmp0, tmp2: temporary registers

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

Repeat

M(A0) – M(A1) (compared by byte)
If M(A0)=M(A1) **and** M(A0)≠0 **then** M(A0+1)–M(A1+1)
(compared by byte)
tmp0 ← M(A0)
tmp1 ← M(A0+1)
tmp2 ← M(A1)
tmp3 ← M(A1+1)
A0 ← A0 + 2
A1 ← A1 + 2

Until (tmp0=0) ıı (tmp1=0) ıı (tmp0≠tmp2) ıı (tmp1≠tmp3)

tmp0, tmp1, tmp2, tmp3: temporary registers

[Function]

- This instruction compares strings until contents do not match when compared in the address incrementing direction from the comparison address (A0) to the compared address (A1), until M(A0) = 0 or M(A0+1)=0 (when (.W) is specified for the size specifier (.size)) .
- The contents of the address register (A0, A1) when the instruction is terminated become undefined.
- When an interrupt is requested during instruction execution, the interrupt is accepted after comparison of one data is completed.

[Flag Change]

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

Conditions

O : The flag is set when a signed operation of M(A0)–M(A1) resulted in exceeding +127 or -128; otherwise cleared.

S : The flag is set when the operation of M(A0)–M(A1) resulted in MSB = 1; otherwise cleared.

Z : The flag is set when fined 0 in M(A0) and terminated, or M(A0)–M(A1)=0 (when compared result is matched); the flag is cleared when M(A0)–M(A1)≠0 (when compared result is not matched).

C : The flag is set when an unsigned operation of M(A0)–M(A1) resulted in any value equal to or greater then 0; otherwise cleared.

[Description Example]

SCMPU.W

SHA

Shift arithmetic

SHift Arithmetic

SHA

[Syntax]

SHA.size src,dest

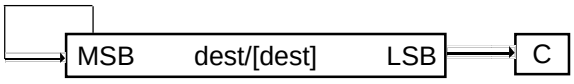
B , W , L

[Instruction Code/Number of Cycles]

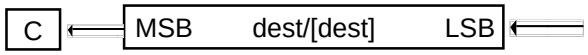
Page= 282

[Operation]

When *src* < 0



When *src* > 0



[Function]

- This instruction arithmetically 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*. When *src* is positive, bits are shifted left; when negative, bits are shifted right.
- When *src* is an immediate and you selected (.B) or (.W) for the size specifier (.size), the number of shifts is -8 to +8(≠0). You cannot set values less than -8, equal to 0, or greater than +8. When you selected (.L) for the size specifier (.size), the number of shifts is -32 to +32(≠0). You cannot set the value 0.
- When *src* is a register, the number of shifts is -32 to +32. Although you can set 0, no bits are shifted and no flags are changed. When you set a value less than -32 or greater than +32, the result of shift is undefined.
- When (.L) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*.

[Selectable src/dest]

src				**dest*1**			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0/R2R0		R0H/R2/-	
~~R1L/R1/R3R1~~		R1H*2/~~R3/-~~		R1L/R1/R3R1*2		R1H/R3/-*2	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0/A0	~~A1/~~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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4/#IMM8*3							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, and R1H/R3/-.

*2 When *src* is R1H, you cannot choose R1, R1H or R3R1 for *dest*.

*3 When (.B) or (.W) is selected for the size specifier (.size), the range of values is -8 ≤ #IMM4 ≤ +8(≠0). When (.L) is selected for the size specifier (.size), the range of values is -32 ≤ #IMM8 ≤ +32 (≠0).

[Flag Change]*4

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

*4 When the number of shifts is 0, no flags are changed.

Conditions

O*5 : The flag is cleared when all the shift resulted in MSB and shift out bit are the same value; otherwise set.

S*5 : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z*5 : The flag is set when the operation resulted in 0; otherwise cleared.

C*5 : The flag is set when the bit at last shifted out is 1; otherwise cleared.

*5 When (.L) is specified for the sign specifier (.size) and dest is the address register(A0, A1), the flag become undefined.

[Description Example]

```
SHA.B   #3,R0L          ; Arithmetically shifted left
SHA.B   #-3,R0L         ; Arithmetically shifted right
SHA.L   R1H,Ram:8[A1]
SHA.W   R1H,[[A1]]
```

SHANC

Shift arithmetic

SHift Arithmetic Non Carry

SHANC

[Syntax]

SHANC.size src,dest

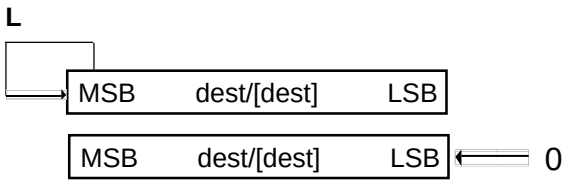
[Instruction Code/Number of Cycles]

Page= 284

[Operation]

When *src* < 0

When *src* > 0



[Function]

- This instruction arithmetically shifts *dest* left or right the number of bits indicated by *src*.
- The direction of shift is determined by the sign of *src*. When *src* is positive, bits are shifted left; when negative, bits are shifted right. Data which are compensated for shift are the sign of MSB when *src* > 0 (negative), or 0 when *src* < 0 (positive).
- The number of shifts is -32 to +32. You cannot set values less than -32, equal to 0, or greater than +32.
- When *dest* is the address register (A0, A1), *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*.

[Selectable src/dest]

src				dest*1			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0/~~A0	~~A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
~~#IMM4~~/#IMM8*2							

*1 Indirect instruction addressing [dest] can be used in all addressing except R2R0 and R3R1.

*2 The range of values is -32 ≤ #IMM8 ≤ +32(≠0).

[Flag Change]

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

Conditions

S*3 : The flag is set when the operation resulted in MSB = 1; otherwise cleared.

Z*3 : The flag is set when the operation resulted in 0; otherwise cleared.

*3 When dest is the address register (A0, A1), the flag become undefined.

[Description Example]

```
SHANC.L   #3,R2R0          ; Arithmetically shifted left
SHANC.L   #-3,R2R0         ; Arithmetically shifted right
SHANC.L   #10,Ram:8[A1]    ; Arithmetically shifted left
SHANC.L   #11,[[A1]]       ; Arithmetically shifted right
```

SHL

Shift logical
SHift Logical

SHL

[Syntax]

SHL.size src,dest

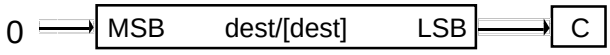
B , W , L

[Instruction Code/Number of Cycles]

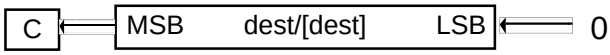
Page=285

[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*. When *src* is positive, bits are shifted left; when negative, bits are shifted right.
- When *src* is an immediate and (.B) or (.W) is specified for the size specifier (.size), the number of shifts is -8 to +8(≠0). You cannot set values less than -8, equal to 0, or greater than +8. When (.L) is specified for the size specifier (.size), the number of shifts is -32 to +32 (≠0). You cannot set the value 0.
- When *src* is a register, the number of shifts is -32 to +32. Although you can set 0, no bits are shifted and no flags are changed. When you set a value less than -32 or greater than +32, the result of shift is undefined.

[Selectable src/dest]

src				**dest**[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0/R2R0		R0H/R2/-	
~~R1L/R1/R3R1~~		R1H[*2]~~/R3/-~~		R1L/R1/R3R1[*2]		R1H/R3/-[*2]	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/~~A0/A0	~~A1/~~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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM4/#IMM8[*3]							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, and R1H/R3/-.

*2 When *src* is R1H, you cannot choose R1, R1H or R3R1 for *dest*.

*3 When (.B) or (.W) is selected for the size specifier (.size), the range of values is -8 ≤ #IMM4 ≤ +8(≠0). When (.L) is selected for the size specifier (.size), the range of values is -32 ≤ #IMM8 ≤ +32 (≠0).

[Flag Change][*4]

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

*4 When the number of shifts is 0, no flags are changed.

Conditions

S[*5]: The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z[*5]: The flag is set when the operation resulted in 0; otherwise cleared.
C[*5]: The flag is set when the bit shifted out last is 1; otherwise cleared.

*5 When (.L) is specified for the sign specifier (.size) and dest is the address register(A0, A1), the flag become undefined.

[Description Example]

```
SHL.B   #3,R0L          ; Logically shifted left
SHL.B   #-3,R0L         ; Logically shifted right
SHL.L   R1H,Ram:8[A1]
SHL.W   R1H,[[A0]]
```

SHLNC

Shift logical
SHift Logical Non Carry

SHLNC

[Syntax]

SHLNC.size src,dest

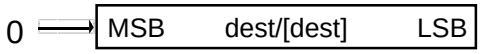
L

[Instruction Code/Number of Cycles]

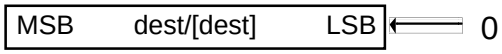
Page=288

[Operation]

When *src* < 0



When *src* > 0



[Function]

- This instruction logically shifts *dest* left or right the number of bits indicated by *src*.
- The direction of shift is determined by the sign of *src*. When *src* is positive, bits are shifted left; when negative, bits are shifted right. Data which are compensated for shift are 0, regardless of the sign of *src*.
- The number of shifts is -32 to +32. You cannot set values less than -32, equal to 0, or greater than +32.
- When *dest* is the address register (A0, A1), *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*.

[Selectable src/dest]

src				**dest**[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	~~A0/A0/~~A0	~~A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
~~#IMM4~~/#IMM8[*2]							

*1 Indirect instruction addressing [dest] can be used in all addressing except R2R0 and R3R1.
*2 The range of values is -32 ≤ #IMM8 ≤ +32(≠0).

[Flag Change]

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

Conditions

S*3 : The flag is set when the operation resulted in MSB = 1; otherwise cleared.
Z*3 : The flag is set when the operation resulted in 0; otherwise cleared.
*3 When dest is the address register (A0, A1), the flag become undefined.

[Description Example]

```
SHLNC.L   #3,R2R0          ; Logically shifted left
SHLNC.L   #-3,R2R0         ; Logically  shifted right
SHLNC.L   #10,Ram:8[A1]    ; Logically shifted left
SHLNC.L   #11,[[A0]]       ; Logically shifted right
```

SIN

String input

String INput

SIN

[Syntax]

SIN.size

B , W

[Instruction Code/Number of Cycles]

Page=288

[Operation]*1

• When size specifier (.size) is (.B)

While R3≠0 **Do**

M(A1) ← M(A0)
A1 ← A1 + 1
R3 ← R3 - 1

End

• When size specifier (.size) is (.W)

While R3≠0 **Do**

M(A1) ← M(A0)
A1 ← A1 + 2
R3 ← R3 - 1

End

*1 When you set a value 0 in R3, this instruction is ignored.

[Function]

- This instruction transfers string from the fixed source address indicated by A0 to the destination address indicated by A1 in the address incrementing direction as many times as specified by R3.
- Set the source of transfer address in A0, the destination address in A1, and the transfer count in R3.
- The content of A1, when the instruction is terminated, indicates the next address of the last-transferred data.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SIN.W

SMOVB

Transfer string backward

String MOVe Backward

SMOVB

[Syntax]

SMOVB.size

B , W

[Instruction Code/Number of Cycles]

Page=289

[Operation]*1

• When size specifier (.size) is (.B)

While R3≠0 **Do**

M(A1) ← M(A0)
A0 ← A0 - 1
A1 ← A1 - 1
R3 ← R3 - 1

End

• When size specifier (.size) is (.W)

While R3≠0 **Do**

M(A1) ← M(A0)
A0 ← A0 - 2
A1 ← A1 - 2
R3 ← R3 - 1

End

*1 When you set a value 0 in R3, this instruction is ignored.

[Function]

- This instruction transfers string in successively address decrementing direction from the source address indicated by A0 to the destination address indicated by A1.
- Set the transfer count in R3.
- The address register (A0, A1), when the instruction is completed, contains the next address of the last-read data.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SMOVB.B

SMOVF

Transfer string forward
String MOVe Forward

SMOVF

[Syntax]

SMOVF.size
B , W

[Instruction Code/Number of Cycles]

Page=289

[Operation][*1]

• When size specifier (.size) is (.B)

While R3≠0 **Do**
M(A1) ← M(A0)
A0 ← A0 + 1
A1 ← A1 + 1
R3 ← R3 - 1
End

• When size specifier (.size) is (.W)

While R3≠0 **Do**
M(A1) ← M(A0)
A0 ← A0 + 2
A1 ← A1 + 2
R3 ← R3 - 1
End

*1 When you set a value 0 in R3, this instruction is ignored.

[Function]

- This instruction transfers string in successively address incrementing direction from the source address indicated by A0 to the destination address indicated by A1.
- Set the transfer count in R3.
- The address register (A0, A1) when the instruction is completed contains the next address of the last-read data.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SMOVF.W

SMOVU

Transfer string
String MOVe Unequal

SMOVU

[Syntax]

SMOVU.size

B , W

[Instruction Code/Number of Cycles]

Page=290

[Operation]

• When size specifier (.size) is (.B)

Repeat

M(A1) ← M(A0) (transfered by byte)
tmp0 ← M(A0)
A0 ← A0 + 1
A1 ← A1 + 1

Until tmp0 = 0

tmp0: temporary register

• When size specifier (.size) is (.W)

Repeat

M(A1) ← M(A0) (transfered by word)
tmp0 ← M(A0)
tmp1 ← M(A0 + 1)
A0 ← A0 + 2
A1 ← A1 + 2

Until (tmp0 = 0) ıı (tmp1 = 0)

tmp0, tmp1: temporary registers

[Function]

- This instruction transfers strings from the source address indicated by A0 to the destination address indicated by A1 in the address incrementing direction until 0 is detected.
- The contents of the address register (A0, A1), when the instruction is terminated, become undefined.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SMOVU.B

SOUT

Store string output

String OUTput

SOUT

[Syntax]

SOUT.size

B , W

[Instruction Code/Number of Cycles]

Page= 290

[Operation][*1]

• When size specifier (.size) is (.B)

While R3≠0 **Do**

M(A1) ← M(A0)
A0 ← A0 + 1
R3 ← R3 - 1

End

• When size specifier (.size) is (.W)

While R3≠0 **Do**

M(A1) ← M(A0)
A0 ← A0 + 2
R3 ← R3 - 1

End

*1 When you set a value 0 in R3, this instruction is ignored.

[Function]

- This instruction transfers string from the source address indicated by A0 to the fixed destination address indicated by A1 in the address incrementing direction as many times as specified by R3.
- Set the source of transfer address in A0, the destination address in A1, and the transfer count in R3.
- The content of A0, when the instruction is terminated, indicates the next address of the last-transferred data.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SOUT.W

SSTR

Store string
String SToRe

SSTR

[Syntax]

SSTR.size
└ **B , W**

[Instruction Code/Number of Cycles]
Page= 291

[Operation][*1]

• When size specifier (.size) is (.B)

While R3≠0 **Do**
M(A1) ← R0L
A1 ← A1 + 1
R3 ← R3 - 1
End

• When size specifier (.size) is (.W)

While R3≠0 **Do**
M(A1) ← R0
A1 ← A1 + 2
R3 ← R3 - 1
End

*1 When you set a value 0 in R3, this instruction is ignored.

[Function]

- This instruction stores string, with the store data indicated by R0L/R0, the transfer address indicated by A1, and the transfer count indicated by R3.
- The content of A1, when the instruction is completed, indicates the next address of the last-written data.
- If an interrupt is requested in the middle of one transfer, the interrupt is acknowledged as soon as the one transfer is completed.

[Flag Change]

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

[Description Example]

SSTR.B

STC

Transfer from control register

STore from Control register

STC

[Syntax]

STC src,dest

[Instruction Code/Number of Cycles]

Page= 291

[Operation]

dest ← src

[Function]

- This instruction transfers the control register indicated by *src* to *dest*. When *dest* is memory, specify the address in which to store the low-order address.
- When memory is specified for *dest*, the following bytes of memory are required.
  2 bytes : DMD0*1, DMD1*1, FLG, DCT0, DCT1, DRC0, DRC1, SVF
  4 bytes : FB*1, SB*1, SP*1, ISP*1, INTB*1, VCT*1, SVP*1, DMA0*1, DMA1*1, DRA0*1, DRA1*1, DSA0*1, DSA1*1

*1 The 1 high-order byte of dest becomes undefined.

[Selectable src/dest]

src				dest			
DMD0	DMD1	DCT0	DCT1	~~R0L/~~R0~~/R2R0~~		~~R0H/~~R2~~/-~~	
DRC0	DRC1	FLG	SVF	~~R1L/~~R1~~/R3R1~~		~~R1H/~~R3~~/-~~	
				~~A0/~~A0~~/A0~~	~~A1/~~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]	dsp:16[FB]
				dsp:24[A0]	dsp:24[A1]	abs24	abs16
FB	SB	SP*2	ISP	~~R0L/R0/~~R2R0		~~R0H/R2/~~-	
INTB	VCT	SVP		~~R1L/R1/~~R3R1		~~R1H/R3/~~-	
DMA0	DMA1	DRA0	DRA1	~~A0/A0/~~A0	~~A1/A1/~~A1	[A0]	[A1]
DSA0	DSA1			dsp:8[A0]	dsp:8[A1]	dsp:8[SB]	dsp:8[FB]
				dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
				dsp:24[A0]	dsp:24[A1]	abs24	abs16

*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]

```
STC     FLG,R0
STC     FB,A0
```

STCTX

Save context
STore ConTeXt

STCTX

[Syntax]

STCTX **abs16,abs24**

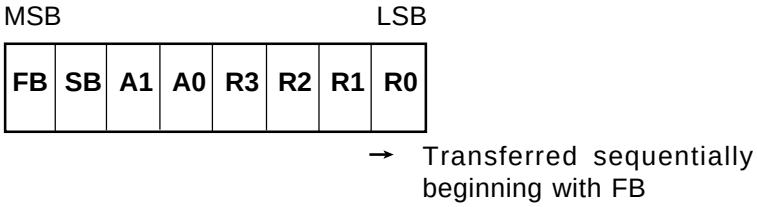
[Instruction Code/Number of Cycles]

Page= 293

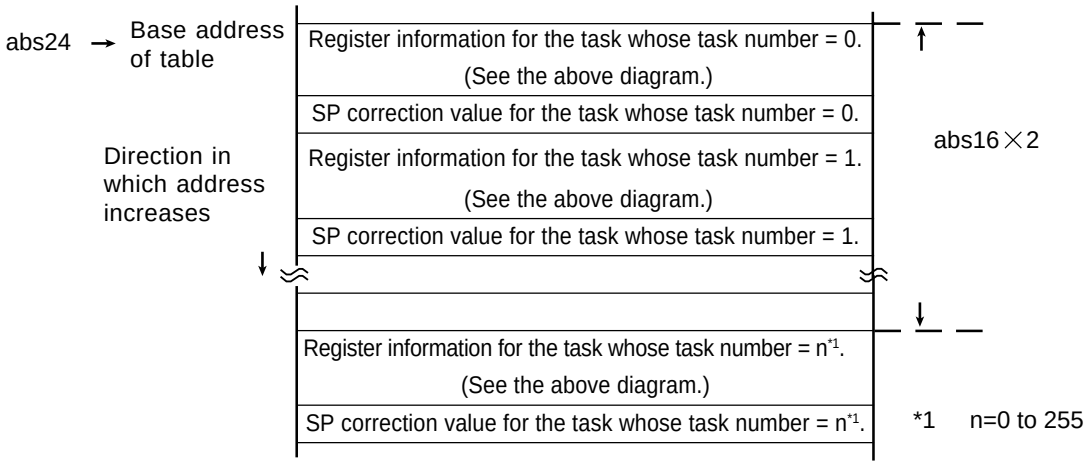
[Operation]

[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 abs24.
- 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 be transferred. Calculated as 2 bytes when transferring the R0, R1, R2, or R3 registers. A0, A1, SB, and FB are calculated as 4 bytes.
- 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.



- The table data is comprised as shown below. The address indicated by abs24 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.



[Flag Change]

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

[Description Example]

```
STCTX   Ram,Rom_TBL
```

STNZ

Conditional transfer

STore on Not Zero

STNZ

[Syntax]

STNZ.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 293

[Operation]

if Z = 0 **then** dest/[dest] ← src

[Function]

- This instruction transfers *src* to *dest* when the Z flag is 0. *dest* is not changed when the Z flag is 1.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable src/dest]

src				**dest*1**			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0/~~A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/~~R2R0~~, R0H/R2/~~-~~, R1L/R1/~~R3R1~~, and R1H/R3/~~-~~.

[Flag Change]

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

[Description Example]

```
STNZ.B    #5,Ram:8[SB]
STNZ.W    #15,[[A1]]
```

STZ

Conditional transfer

STore on Zero

STZ

[Syntax]

STZ.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 294

[Operation]

if Z = 1 **then** dest/[dest] ← src

[Function]

- This instruction transfers *src* to *dest* when the Z flag is 1. *dest* is not changed when the Z flag is 0.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable src/dest]

src				**dest**[*1]			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0~~/A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[Flag Change]

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

[Description Example]

```
STZ.B   #5,Ram:8[SB]
STZ.W   #10,[[A0]]
```

STZX

Conditional transfer

STore on Zero eXtention

STZX

[Syntax]

STZX.size **src1,src2,dest**

B , W

[Instruction Code/Number of Cycles]

Page= 294

[Operation]

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

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*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0.

[Selectable src/dest]

src				**dest*1**			
~~R0L/R0/R2R0~~		~~R0H/R2/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
~~R1L/R1/R3R1~~		~~R1H/R3/-~~		R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/A0/A0~~	~~A1/A1/A1~~	~~[A0]~~	~~[A1]~~	A0/A0~~/A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[Flag Change]

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

[Description Example]

```
STZX.B   #1,#2,Ram:8[SB]
STZX.W   #5,#10,[R0]
```

SUB

Subtract without borrow
SUBtract

SUB

[Syntax]

SUB.size (:format) src,dest

G , S (Can be specified)
B , W , L

[Instruction Code/Number of Cycles]

Page=295

[Operation]

dest	←	dest	-	src	[dest]	←	[dest]	- src
dest	←	dest	-	[src]	[dest]	←	[dest]	- [src]

[Function]

- This instruction subtracts *src* from *dest* and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.
- When (.L) is specified for the size specifier (.size) and *dest* is the address register, *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*. When *src* is the address register, *src* is zero-extended to be treated as 32-bit data for the operation. The flags also change states depending on the result of 32-bit operation.

[Selectable src/dest]*1

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

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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 +2147483647(.L) or -2147483648(.L), +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]
```

[src/dest Classified by Format]

G format*3

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*4	A1/A1/A1*4	[A0]	[A1]	A0/A0/A0*4	A1/A1/A1*4	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16/#IMM32							

*3 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

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

S format

src				dest*5			
~~R0L/R0~~	~~dsp:8[SB]~~	~~dsp:8[FB]~~	~~abs16~~	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16*6							

*5 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, SP/SP/SP, dsp:8[SP], and #IMM.

*6 You can specify only (.B) or (.W) for the size specifier (.size).

SUBX

Subtract extend without borrow

SUBtract eXtend

SUBX

[Syntax]

SUBX src,dest

[Instruction Code/Number of Cycles]

Page= 299

[Operation]

dest ← dest - EXT(src)　　[dest] ← [dest] - EXT(src)

dest ← dest - EXT([src])　　[dest] ← [dest] - EXT([src])

[Function]

- This instruction subtracts 8-bit *src* from *dest* (32 bits) after sign-extending *src* to 32 bits and stores the result in *dest*.
- When *dest* is the address register (A0, A1), *dest* is zero-extended to be treated as 32-bit data for the operation. The 24 low-order bits of the operation result are stored in *dest*. The flags also change states depending on the result of 32-bit operation.

[Selectable src/dest]*1

src				dest			
R0L/~~R0/R2R0~~		R0H/~~R2/-~~		~~R0L/R0/~~R2R0		~~R0H/R2/~~-	
R1L/~~R1/R3R1~~		R1H/~~R3/-~~		~~R1L/R1/~~R3R1		~~R1H/R3/~~-	
A0/~~A0/A0~~	A1/~~A1/A1~~	[A0]	[A1]	~~A0/A0/~~A0	~~A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/R2R0, R0H/~~R2~~/-, R1L/R1/R3R1, R1H/R3/-, and #IMM.

[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 +2147483647(.L) or -2147483648(.L); 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]

```
SUBX    R0L,A0
SUBX    Ram:8[SB],R2R0
SUBX    #2,[A0]
```

TST

Test

TeST

TST

[Syntax]

TST.size(:format) src,dest

- **G , S** (Can be specified)
- **B , W**

[Instruction Code/Number of Cycles]

Page= 301

[Operation]

dest ∧ src

[Function]

- Each flag in the flag register changes state depending on the result of logical AND of *src* and *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]

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

src				**dest**			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
A0/A0/~~A0~~*1	A1/A1/~~A1~~*1	[A0]	[A1]	A0/A0/~~A0~~*1	A1/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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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 R0L are operated on.
TST.B   R0L,A0      ; R0L is zero-expanded and operated on with A0.
```

[src/dest Classified by Format]

G format

src				dest			
R0L/R0/R2R0		R0H/R2/-		R0L/R0/R2R0		R0H/R2/-	
R1L/R1/R3R1		R1H/R3/-		R1L/R1/R3R1		R1H/R3/-	
A0/A0/A0*2	A1/A1/A1*2	[A0]	[A1]	A0/A0/A0*2	A1/A1/A1*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

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

S format

src				dest			
R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16	R0L/R0	dsp:8[SB]	dsp:8[FB]	abs16
#IMM8/#IMM16							

UND

Interrupt for undefined instruction

UNDefined instruction

UND

[Syntax]

UND

[Instruction Code/Number of Cycles]

Page= 303

[Operation]

SP ← SP - 2
M(SP) ← FLG
SP ← SP - 2
M(SP)*1 ← (PC + 1)H
SP ← SP - 2
M(SP) ← (PC + 1)L
PC ← M(FFFFDC16)

*1 The 8 high-order bits become undefined.

[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	○	○	–	–	–	–	○	–

*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.

Conditions

U : The flag is cleared.
I : The flag is cleared.
D : The flag is cleared.

[Description Example]

UND

WAIT

Wait
WAIT

WAIT

[Syntax]

WAIT

[Instruction Code/Number of Cycles]

Page= 303

[Operation]

[Function]

- Stops program execution. Program execution is restarted when an interrupt whose priority level is higher than bits RLVL2 to RLVL0 in the RLVL register is accepted 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= 304

[Operation]

dest/[dest] ←→ src

[Function]

- This instruction exchanges contents between *src* and *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is address register(A0, A1), 24 bits of zero-expanded *src* data are placed in the address register and the 8 low-order bits of the address register are placed in *src*.
- When (.W) is specified for the size specifier (.size) and *dest* is address register, 24 bits of zero- expanded *src* data are placed in the address register and the 16 low-order bits of the address register are placed in *src*. When *src* is address register, 24 bits data are placed in the address register and the 16 low-order bits of the address register are placed in *dest*.

[Selectable src/dest]

src				**dest*1**			
R0L/R0~~/R2R0~~		R0H/R2~~/-~~		R0L/R0~~/R2R0~~		R0H/R2~~/-~~	
R1L/R1~~/R3R1~~		R1H/R3~~/-~~		R1L/R1~~/R3R1~~		R1H/R3~~/-~~	
~~A0/~~A0~~/A0~~	~~A1/~~A1~~/A1~~	~~[A0]~~	~~[A1]~~	A0/A0~~/A0~~	A1/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]~~	~~dsp:16[FB]~~	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
~~dsp:24[A0]~~	~~dsp:24[A1]~~	~~abs24~~	~~abs16~~	dsp:24[A0]	dsp:24[A1]	abs24	abs16
~~#IMM~~							

*1 Indirect instruction addressing [dest] can be used in all addressing except R0L/R0~~/R2R0~~, R0H/R2~~/-~~, R1L/R1~~/R3R1~~, and R1H/R3~~/-~~.

[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]
```

XOR

Exclusive OR
eXclusive OR

XOR

[Syntax]

XOR.size src,dest

B , W

[Instruction Code/Number of Cycles]

Page= 304

[Operation]

dest ← dest ∀ src
dest ← dest ∀ [src]
[dest] ← [dest] ∀ src
[dest] ← [dest] ∀ [src]

[Function]

- This instruction exclusive ORs *src* and *dest* together and stores the result in *dest*.
- When (.B) is specified for the size specifier (.size) and *dest* is the address register (A0, A1), *src* is zero-extended to be treated as 16-bit data for the operation. In this case, the 8 high-oreder bits become 0. Also, when *src* is the address register, the 8 low-order bits of the address register are used as data to be operated on.
- When (.W) is specified for the size specifier (.size) and *dest* is the address register, the 8 high-order bits become 0. Also, when *src* is the address register, the 16 low-order bits of the address register are the data to be operated on.

[Selectable src/dest]*1

src				dest			
R0L/R0/~~R2R0~~		R0H/R2/~~-~~		R0L/R0/~~R2R0~~		R0H/R2/~~-~~	
R1L/R1/~~R3R1~~		R1H/R3/~~-~~		R1L/R1/~~R3R1~~		R1H/R3/~~-~~	
A0/A0/~~A0~~*2	A1/A1/~~A1~~*2	[A0]	[A1]	A0/A0/~~A0~~*2	A1/A1/~~A1~~*2	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16
#IMM8/#IMM16							

*1 Indirect instruction addressing [src] and [dest] can be used in all addressing except R0L/R0/~~R2R0~~, R0H/R2/~~-~~, R1L/R1/~~R3R1~~, R1H/R3/~~-~~, and #IMM.

*2 When you specify (.B) for the size specifier (.size), you cannot choose A0 and/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
XOR.W   [A0],[[A1]]
```

3.3 Index instructions

This section explains each INDEX instruction individually.
The INDEX instructions are provided for use on arrays. The effective addresses are derived by unsigned adding the addresses indicated by src and dest of the next instruction to be executed after the INDEX instruction to the content of src of the INDEX instruction.
The modifiable size is from 0 to 65535(64KB).
No interrupt request is not accepted immediately after the INDEX instruction.
The 10 types of INDEX instructions shown below are supported.

(1) INDEXB.size src

The INDEXB (INDEX Byte) instruction is used for arrays arranged in bytes.
The effective addresses for the INDEXB instruction are derived by unsigned adding the src content of the INDEXB instruction to the addresses indicated by src and dest of the next instruction to be executed.
For the next instruction executed after the INDEXB instruction, be sure to choose memory for both src and dest. Also, specify .B for the size specifier.

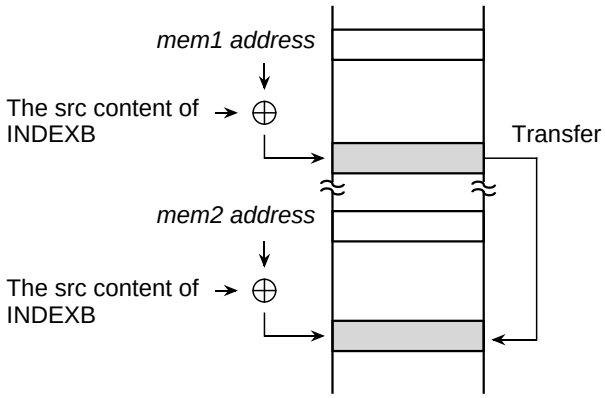
Example:

```
INDEXB.B    src
MOV.B:G     mem1,mem2
      Specify .B      Memory
```

Operation in C language

```
char    src;
char    mem1[],mem2[];

mem2[src] = mem1[src];
```



Instruction which is modified by INDEXB

The src and dest of
ADC, ADD:G*1*2, AND, CMP:G*1, MAX, MIN, MOV:G*1*3, MUL, MULU, OR, SBB, SUB,
TST, XOR.
*1 You can only specify G format.
*2 The SP can not be used in dest of ADD instruction.
*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXB instruction.

(2) INDEXBD.size src

The INDEXBD (INDEX Byte Dest) instruction is used for arrays arranged in bytes.

The effective addresses for the INDEXBD instruction are derived by unsigned adding the src content of the INDEXBD instruction to the addresses indicated by dest(some instructions are src) of the next instruction to be executed.

For the next instruction executed after the INDEXBD instruction, be sure to choose memory for dest(some instructions are src). Also, specify .B for the size specifier.

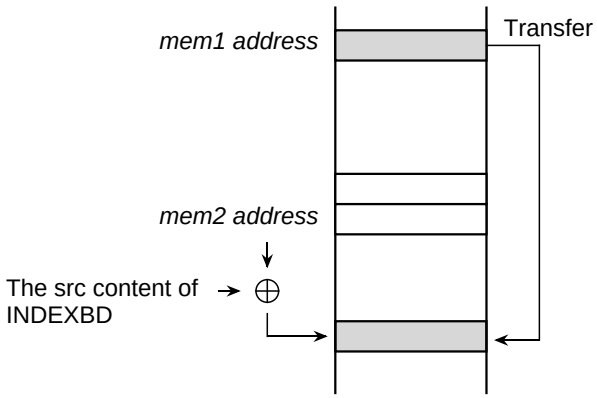
Example:

```
INDEXBD.B       src
MOV.B:G         mem1,mem2
        Specify .B      Memory
```

Operation in C language

```
char            src,mem1;
char            mem2[];

mem2[src] = mem1;
```



Instruction which is modified by INDEXBD

The dest of

ABS, ADC, ADCF, ADD:G*1*2, AND, CLIP, CMP:G*1, DEC, INC, MAX, MIN, MOV:G*1*3, MUL, MULU, NEG, NOT, OR, POP, ROLC, RORC, ROT, SBB, SHA, SHL, STNZ, STZ, STZX, SUB, TST, XCHG, XOR.

The src of

DIV, DIVU, DIVX, PUSH

*1 You can only specify G format.

*2 The SP can not be used in dest of ADD instruction.

*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXBD instruction.

(3) INDEXBS.size src

The INDEXBS (INDEX Byte Src) instruction is used for arrays arranged in bytes.
The effective addresses for the INDEXBS instruction are derived by unsigned adding the src content of the INDEXBS instruction to the addresses indicated by src of the next instruction to be executed.
For the next instruction executed after the INDEXBS instruction, be sure to choose memory for src.
Also, specify .B for the size specifier.

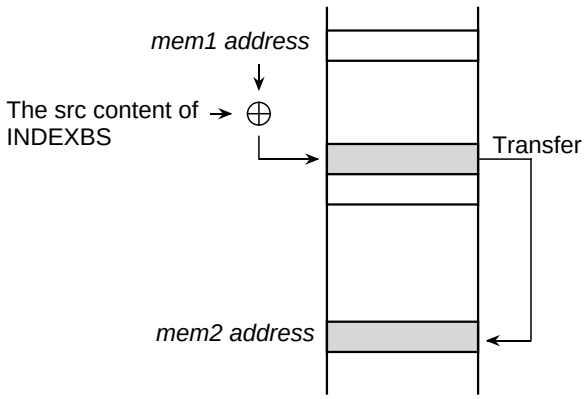
Example:

```
INDEXBS.B       src
MOV.B:G         mem1,mem2
        Specify .B      Memory
```

Operation in C language

```
char            src,mem2;
char            mem1[];

mem2 = mem1[src];
```



Instruction which is modified by INDEXBS

The src of
ADC, ADD:G*1*2, AND, CMP:G*1, MAX, MIN, MOV:G*1*3, MUL, MULU, OR, SBB, SUB, TST, XOR

*1 You can only specify G format.
*2 The SP can not be used in dest of ADD instruction.
*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXBS instruction.

(4)INDEXW.size src

The INDEXW (INDEX Word) is used for arrays arranged in words.

The effective addresses for the INDEXW instruction are derived by unsigned adding twice the src content of the INDEXW instruction to the addresses indicated by src and dest of the next instruction to be executed. The range of src of INDEXW instruction is from 0 to 32767. You can not set otherwise.

For the next instruction executed after the INDEXW instruction, be sure to choose memory for both src and dest. Also, specify .W for the size specifier.

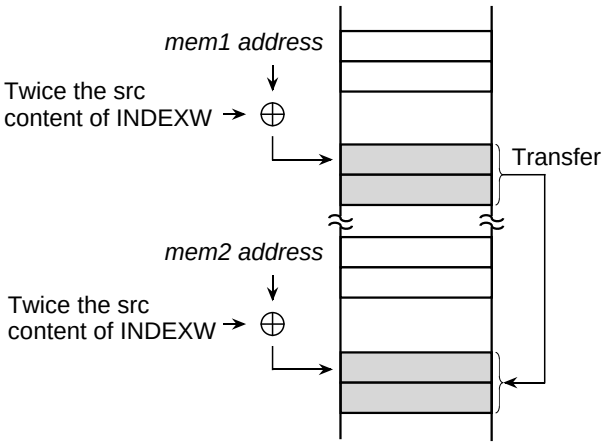
Example:

```
INDEXW.B        src
MOV.W:G         mem1,mem2
      Specify .W        Memory
```

Operation in C language

```
char            src;
char            mem1[],mem2[];

mem2[src] = mem1[src];
```



Instruction which is modified by INDEXW

The src and dest of
ADC, ADD:G*1*2, AND, CMP:G*1, MAX, MIN, MOV:G*1*3, MUL, MULU, OR, SBB, SUB, TST, XOR.

*1 You can only specify G format.
*2 The SP can not be used in dest of ADD instruction.
*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXW instruction.

(5) INDEXWD.size src

The INDEXWD (INDEX Word Dest) is used for arrays arranged in words.

The effective addresses for the INDEXWD instruction are derived by unsigned adding twice the src content of the INDEXWD instruction to the addresses indicated by dest (some instructions are src) of the next instruction to be executed.

The range of src of INDEXWD instruction is from 0 to 32767. You cannot set otherwise.

For the next instruction executed after the INDEXWD instruction, be sure to choose memory for dest(some instructions are src). Also, specify .W for the size specifier.

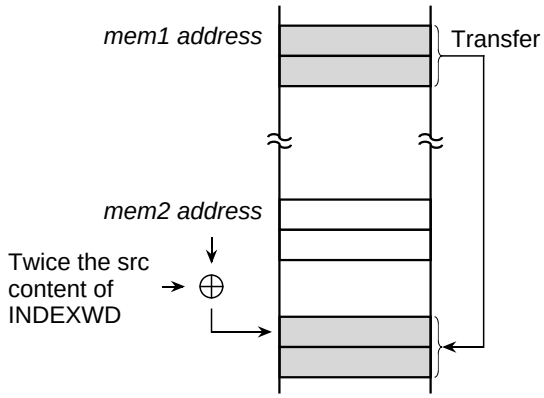
Example:

```
INDEXWD.B       src
MOV.W:G         mem1,mem2
        Specify .W      Memory
```

Operation in C language

```
char            src;
int             mem1;
int             mem2[];

mem2[src] = mem1;
```



Instruction which is modified by INDEXWD

The dest of

ABS, ADC, ADCF, ADD:G*1*2, AND, CLIP, CMP:G*1, DEC, INC, MAX, MIN, MOV:G*1*3, MUL, MULU, NEG, NOT, OR, POP, ROLC, RORC, ROT, SBB, SCcnd, SHA, SHL, STNZ, STZ, STZX, SUB, TST, XCHG, XOR.

The src of

DIV, DIVU, DIVX, PUSH, JMPI, JSRI.

*1 You can only specify G format.

*2 The SP can not be used in dest of ADD instruction.

*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXWD instruction.

(6) INDEXWS.size src

The INDEXWS (INDEX Word Src) is used for arrays arranged in words.

The effective addresses for the INDEXWS instruction are derived by unsigned adding twice the src content of the INDEXWS instruction to the addresses indicated by src of the next instruction to be executed. The range of src of INDEXWS instruction is from 0 to 32767. You can not set otherwise.

For the next instruction executed after the INDEXWS instruction, be sure to choose memory for src.

Also, specify .W for the size specifier.

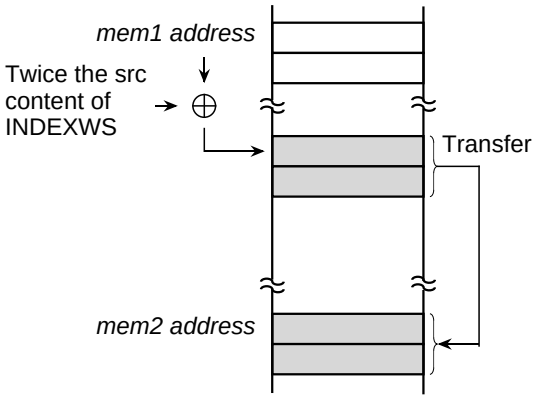
Example:

```
INDEXWS.B       src
MOV.W:G         mem1,mem2
        Specify .W      Memory
```

Operation in C language

```
char            src;
int             mem1[];
int             mem2[];

mem2 = mem1[src];
```



Instruction which is modified by INDEXWS

The src of

ADC, ADD:G*1*2, AND, CMP:G*1, MAX, MIN, MOV:G*1*3, MUL, MULU, OR, SBB, SUB, TST, XOR.

*1 You can only specify G format.

*2 The SP can not be used in dest of ADD instruction.

*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXWS instruction.

(7) INDEXL.size src

The INDEXL (INDEX Long word) is used for arrays arranged in long words.
The effective addresses for the INDEXL instruction are derived by unsigned adding four times the src content of the INDEXL instruction to the addresses indicated by src and dest of the next instruction to be executed. The range of src of INDEXL instruction is from 0 to 16383. You can not set otherwise.
For the next instruction executed after the INDEXL instruction, be sure to choose memory for both src and dest. Also, specify .L for the size specifier.

Example:

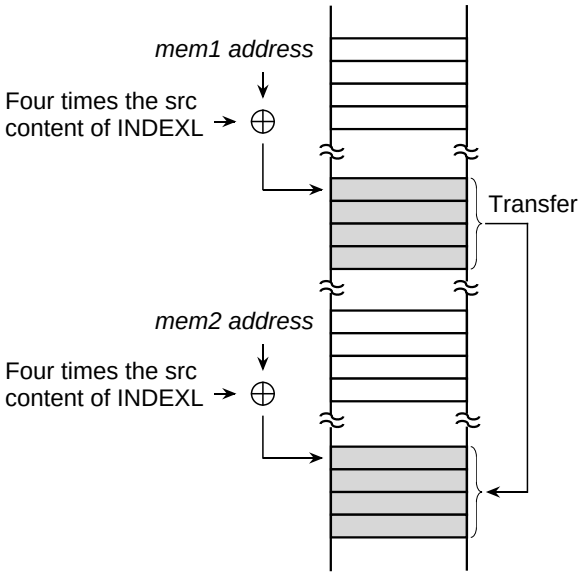
```
INDEXL.B        src
MOV.L:G         mem1,mem2
```

Specify .L — Memory

Operation in C language

```
char            src;
long            mem1[],mem2[];

mem2[src] = mem1[src];
```



Instruction which is modified by INDEXL

The src and dest of
ADD:G*1*2, CMP:G*1, MOV:G*1*3, SUB.

*1 You can only specify G format.
*2 The SP can not be used in dest of ADD instruction.
*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXL instruction.

(8) INDEXLD.size src

The INDEXLD (INDEX Long word Dest) is used for arrays arranged in long words.

The effective addresses for the INDEXLD instruction are derived by unsigned adding four times the src content of the INDEXLD instruction to the addresses indicated by dest (some instructions are src) of the next instruction to be executed. The range of src of INDEXLD instruction is from 0 to 16383. You can not set otherwise.

For the next instruction executed after the INDEXLD instruction, be sure to choose memory for dest (some instructions are src). Also, specify .L for the size specifier.

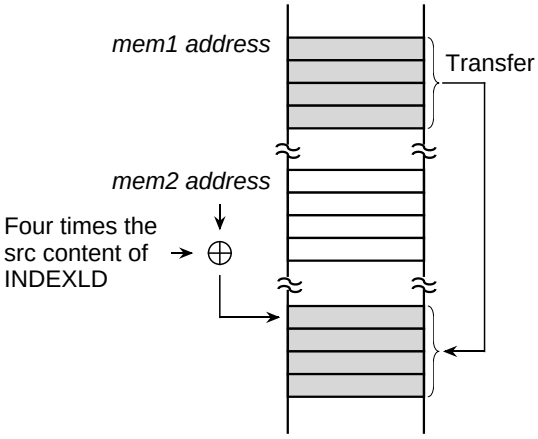
Example:

```
INDEXLD.B       src
MOV.L:G         mem1,mem2
        Specify .L      Memory
```

Operation in C language

```
char    src;
long    mem1;
long    mem2[];

mem2[src] = mem1;
```



Instruction which is modified by INDEXLD

The dest of ADD:G*1*2, CMP:G*1, MOV:G*1*3, SUB, SHA, SHANC, SHL, SHLNC.

The src of DIV, DIVL, DIVX, MUL, MULU, JMPI, JSRI.

*1 You can only specify G format.

*2 The SP can not be used in dest of ADD instruction.

*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXLD instruction.

(9) INDEXLS.size src

The INDEXLS (INDEX Long word Src) is used for arrays arranged in long words.

The effective addresses for the INDEXLS instruction are derived by unsigned adding four times the src content of the INDEXLS instruction to the addresses indicated by src of the next instruction to be executed. The range of src of INDEXLS instruction is from 0 to 16383. You cannot set otherwise.

For the next instruction executed after the INDEXLS instruction, be sure to choose memory for src. Also, specify .L for the size specifier.

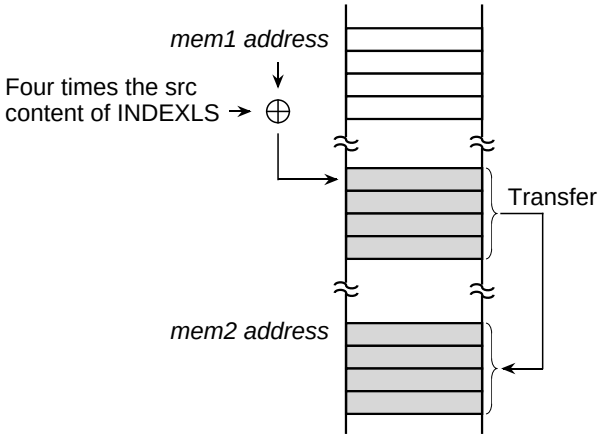
Example:

```
INDEXLS.B       src
MOV.L:G         mem1,mem2
        Specify .L      Memory
```

Operation in C language

```
char            src;
long            mem1[];
long            mem2;

mem2 = mem1[src];
```



Instruction which is modified by INDEXLS

The src of ADD:G*1*2, CMP:G*1, MOV:G*1*3, SUB.

*1 You can only specify G format.
*2 The SP can not be used in dest of ADD instruction.
*3 The dsp:8[SP] can not be used in src or dest of MOV instruction.

Only above instructions can be used next to INDEXLS instruction.

(10) BITINDEX.size src

The BITINDEX instruction is operated on the bit that is apart from bit 0 of the address indicated by dest as many bits as indicated by src of BITINDEX.

Make sure the next instruction to be executed after BITINDEX is a bit instruction. Also, be sure to specify memory for src or dest.

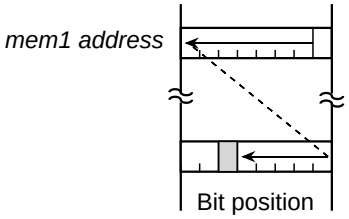
Example:

```
BITINDEX.B/W    src
BSET            3,mem1
```

Bit instruction

Becomes invalid.

Memory



Instruction which is modified by BITINDEX

The src of BAND, BNAND, BNOR, BNTST, BNXOR, BOR, BTST:G*1, BXOR.

The dest of BCLR, BMcnd, BNOT, BSET, BTSTC, BTSTS.

*1 You can only specify G format.

(11) Next instructions that can be executed after INDEX

The table below lists the next instructions that can be executed after each INDEX instruction.

	Valid instruction	
INDEXB.B/.W*2	ADC, ADD:G*4, AND, CMP:G, MAX, MIN, MOV:G*3, MUL, MULU, OR, SBB, SUB,TST,XOR The src and dest of above instructions.	
INDEXBD.B/.W*2	ABS, ADC, ADCF, ADD:G*4, AND, CLIP, CMP:G, DEC, INC, MAX, MIN, MOV:G*3, MUL, MULU, NEG, NOT, OR, POP, ROLC, RORC, ROT, SBB, SCcnd, SHA, SHL, STNZ, STZ, STZX, SUB, TST, XCHG, XOR The dest of above instructions.	DIV, DIVU, DIVX, PUSH The src of above instructions.
INDEXBS.B/.W*2	ADC, ADD:G*4, AND, CMP:G, MAX, MIN, MOV:G*3, MUL, MULU, OR, SBB, SUB, TST, XOR The src of above instructions.	
INDEXW.B/.W*2	ADC, ADD:G*4, AND, CMP:G, MAX, MIN, MOV:G*3, MUL, MULU, OR, SBB, SUB, TST, XOR The src and dest of above instructions.	
INDEXWD.B/.W*2	ABS, ADC, ADCF, ADD:G*4, AND, CLIP, CMP:G, DEC, INC, MAX, MIN, MOV:G*3, MUL, MULU, NEG, NOT, OR, POP, ROLC, RORC, ROT, SBB, SHA, SHL, STNZ, STZ, STZX, SUB, TST, XCHG, XOR The dest of above instructions.	DIV, DIVU, DIVX, PUSH, JMPI, JSRI The src of above instructions.
INDEXWS.B/.W*2	ADC, ADD:G*4, AND, CMP:G, MAX, MIN, MOV:G*3, MUL, MULU, OR, SBB, SUB, TST, XOR The src of above instructions.	
INDEXL.B/.W*2	ADD:G*4, CMP:G, MOV:G*3, SUB The src and dest of above instructions.	
INDEXLD.B/.W*2	ADD:G*4, CMP:G, MOV:G*3, SHA, SHL, SUB The dest of above instructions.	JMPI*1, JSRI*1 The src of above instructions.
INDEXLS.B/.W*2	ADD:G*4, CMP:G, MOV:G*3, SUB The src of above instructions.	
BITINDEX.B/.W	BAND, BNAND, BNOR, BNTST, BNXOR, BOR, BTST:G, BXOR The src of above instructions.	BCLR, BMcnd, BNOT, BSET, BTSTC, BTSTS The dest of above instructions.

*1 Since the size is specified for .A(3 bytes) by .L(4 bytes), care must be taken when using the data table.

*2 The ADD, CMP, and MOV instructions are valid in only the G format.

*3 The dsp:8[SP] cannot be used in src or dest of MOV instruction.

*4 The SP cannot be used in src or dest of ADD instruction.

(12) Addressing modes

The table below lists the addressing modes that become valid in the next instructions that can be executed after INDEX. Indirect instruction addressing modes can be used in each instruction.

src				dest			
[A0]	[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]	dsp:16[FB]	dsp:16[A0]	dsp:16[A1]	dsp:16[SB]	dsp:16[FB]
dsp:24[A0]	dsp:24[A1]	abs24	abs16	dsp:24[A0]	dsp:24[A1]	abs24	abs16

*1 For the MOV instruction you cannot use dsp8:[SP].

*2 The SP in the ADD instruction cannot be used.

*3 You cannot use R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, SP/SP/SP, dsp:8[SP], and #IMM.

Chapter 4

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.

(1) **LDIPL**

(2) **(1) LDIPL #IMM**

(3)

b7				b0	b7				b0
1	1	0	1	0	1	0	1	1	1	1	0	1	#IMM

[Number of Bytes/Number of Cycles]

(4)

Bytes/Cycles	2/2

(1) **MAX**

(2) **(1) MAX.size #IMM,dest**

(3)

b7							b0	b7							b0
0000	0001	1	0	0	0	d4	d3	d2	SIZE	d1	d0	1	1	1	1	1	1

dest code: dsp8 / dsp16/abs16 / dsp24/abs24 — #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

(4) [Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/3	4/3	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5

(1) Mnemonic

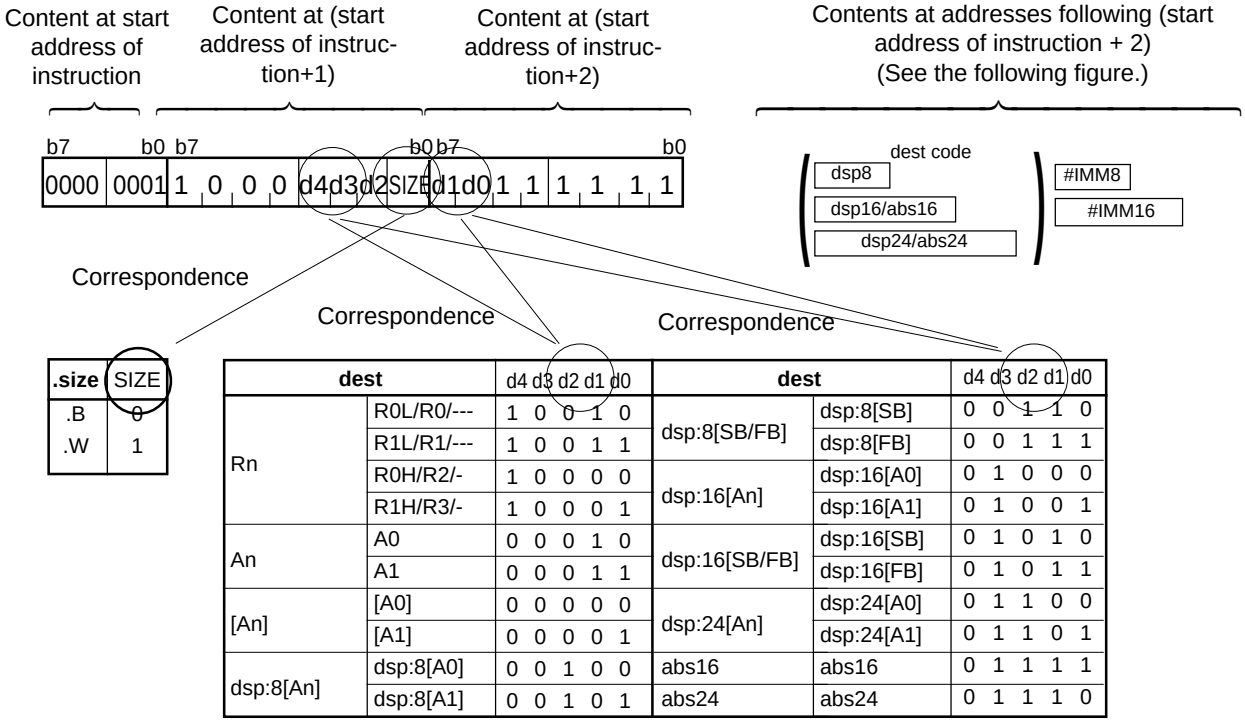
Shows the mnemonic explained in this page.

(2) Syntax

Shows an instruction syntax using symbols.

(3) Instruction code

Shows instruction code. Entered in ( ) are omitted depending on src/dest you selected.

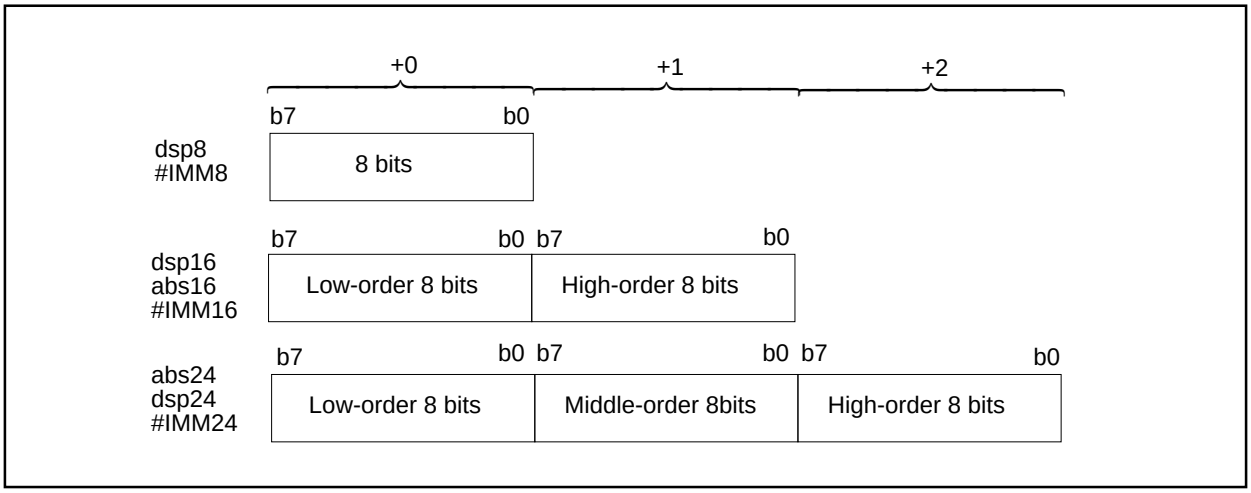


.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0
	R1L/R1/---	1 0 0 1 1
	R0H/R2/-	1 0 0 0 0
	R1H/R3/-	1 0 0 0 1
An	A0	0 0 0 1 0
	A1	0 0 0 1 1
[An]	[A0]	0 0 0 0 0
	[A1]	0 0 0 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0
	dsp:8[A1]	0 0 1 0 1

dest		d4 d3 d2 d1 d0
dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	dsp:8[FB]	0 0 1 1 1
dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	dsp:16[A1]	0 1 0 0 1
dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	dsp:16[FB]	0 1 0 1 1
dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	dsp:24[A1]	0 1 1 0 1
abs16	abs16	0 1 1 1 1
abs24	abs24	0 1 1 1 0

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 bytes in the instruction.

The number of cycles shown are the minimum possible, and they vary depending on the following conditions:

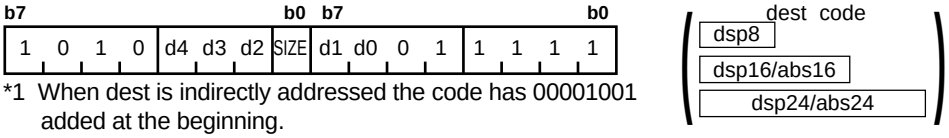
- Number of bytes that have been loaded in the instruction queue buffer
- Accessing of an external memory using 8-bit external bus
- Whether a wait is inserted in the bus cycle

Instruction bytes are indicated on the left side of the slash and execution cycles are indicated on the right side.

4.2 Instruction Code/Number of Cycles

ABS

(1) ABS.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

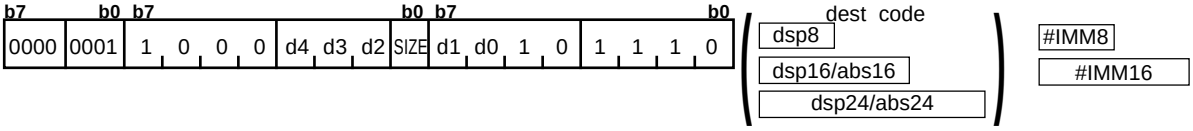
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3, respectively.

ADC

(1) ADC.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

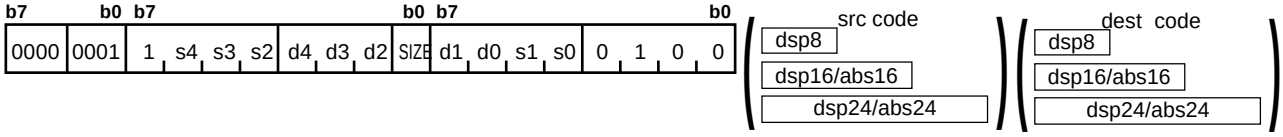
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/1	4/1	4/3	5/3	5/3	6/3	6/3	7/3	6/3	7/3

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

ADC

(2) ADC.size src, dest



.size	SIZE
.B	0
.W	1

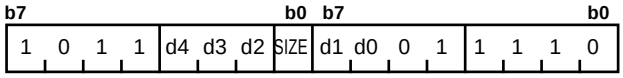
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
An	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:8[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:16[SB/FB]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:24[An]	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4
abs16	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs24	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

ADCF

(1) ADCF.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest code

dest code
dsp8
dsp16/abs16
dsp24/abs24

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

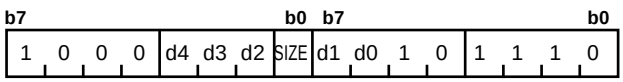
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ADD

(1) ADD.size:G #IMM,dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest code

dest code
dsp8
dsp16/abs16
dsp24/abs24

#IMM8
#IMM16

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

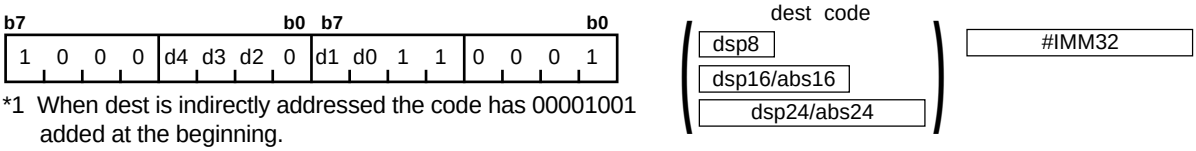
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

ADD

(2) ADD.L:G #IMM,dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

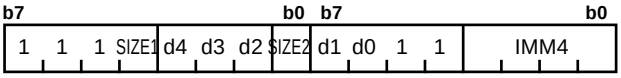
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	6/2	6/2	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

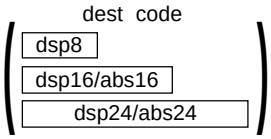
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ADD

(3) ADD.size:Q #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE1	SIZE2
.B	0	0
.W	0	1
.L	1	0

#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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

When (.B) and (.W) is specified for the size specifier (.size)

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

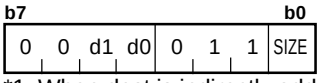
When (.L) is specified for the size specifier (.size)

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*3 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ADD

(4) ADD.size:S #IMM, dest



dest code

dsp8
abs16

#IMM8
#IMM16

*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

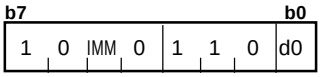
dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size) the number of bytes in the table is increased by 1.

ADD

(5) ADD.L:S #IMM, A0/A1



#IMM	IMM
#1	0
#2	1

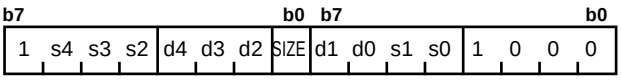
A0/A1	d0
A0	0
A1	1

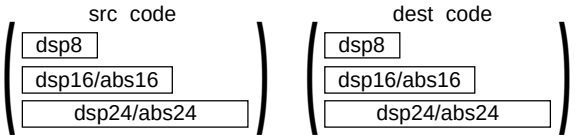
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/2

ADD

(6) ADD.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

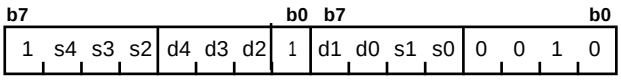
[Number of Bytes/Number of Cycles]

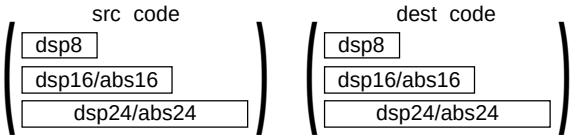
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

ADD

(7) ADD.L:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

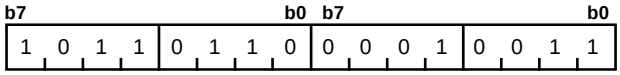
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
An	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
[An]	2/4	2/4	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

ADD

(8) ADD.L:G #IMM16, SP



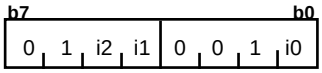
#IMM16

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/2

ADD

(9) ADD.L:Q #IMM3, SP

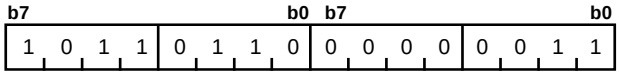


#IMM3	i2 i1 i0	#IMM3	i2 i1 i0
+1	0 0 0	+5	1 0 0
+2	0 0 1	+6	1 0 1
+3	0 1 0	+7	1 1 0
+4	0 1 1	+8	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/1

(10) ADD.L:S #IMM8, SP

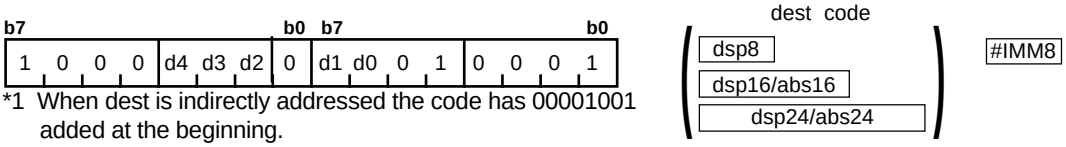


[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/2

ADDX

(1) ADDX #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

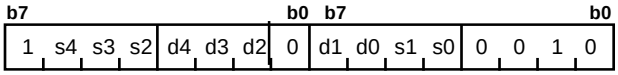
[Number of Bytes/Number of Cycles]

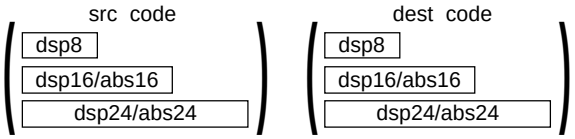
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/5	4/4	4/4	5/4	5/4	6/4	5/4	6/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ADDX

(2) ADDX src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

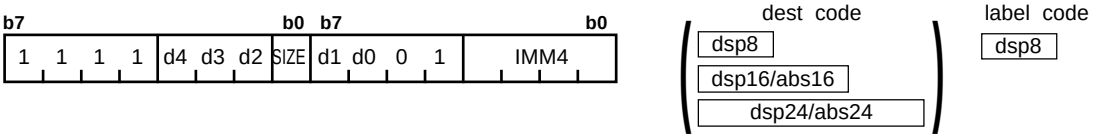
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
An	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
[An]	2/4	2/4	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

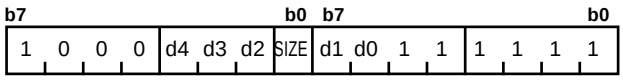
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

*1 When branched to label, the number of cycles in the table is increased by 2.

AND

(1) AND.size:G #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest code: dsp8 / dsp16/abs16 / dsp24/abs24 — #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

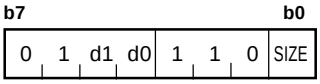
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size) the number of bytes in the table is increased by 1.

AND

(2) AND.size:S #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest code: dsp8 / abs16 — #IMM8 / #IMM16

.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

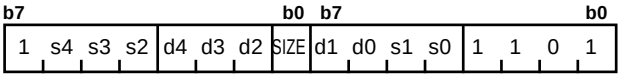
[Number of Bytes/Number of Cycles]

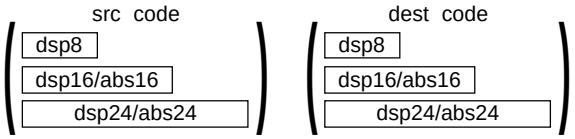
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size) the number of bytes in the table is increased by 1.

AND

(3) AND.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

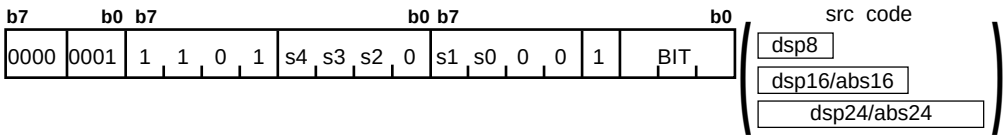
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

BAND

(1) BAND src



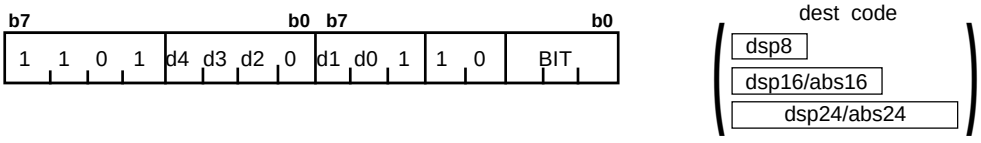
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BCLR

(1) BCLR dest



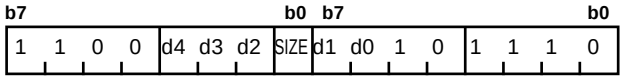
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:27[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

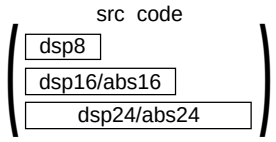
[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

BITINDEX

(1) BITINDEX.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

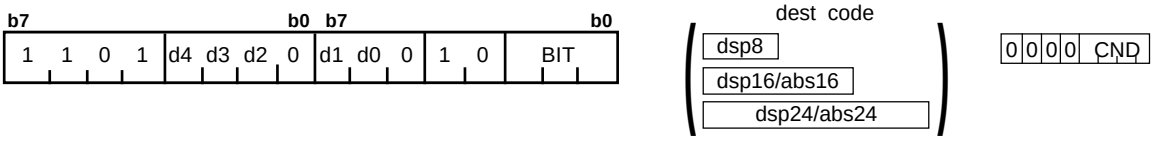
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/4	2/4	2/6	3/3	3/6	4/6	4/6	5/6	4/6	5/6

*1 The cycles of next instruction to be executed is increased by 1.

BM*cnd*

(1) BM*cnd* dest



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

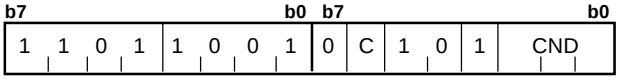
Cnd	CND	***Cnd***	CND
LTU/NC	0 0 0 0	GEU/C	1 0 0 0
LEU	0 0 0 1	GTU	1 0 0 1
NE/NZ	0 0 1 0	EQ/Z	1 0 1 0
PZ	0 0 1 1	N	1 0 1 1
NO	0 1 0 0	O	1 1 0 0
GT	0 1 0 1	LE	1 1 0 1
GE	0 1 1 0	LT	1 1 1 0

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BM*cnd*

(2) BM*cnd* C



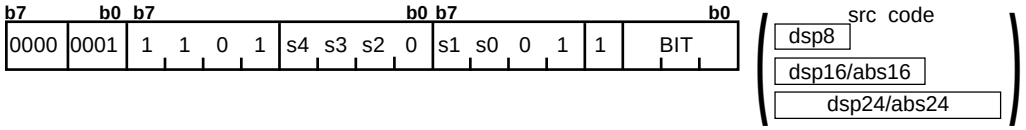
Cnd	C	CND	*Cnd*	C	CND
LTU/NC	0	0 0 0	GEU/C	1	0 0 0
LEU	0	0 0 1	GTU	1	0 0 1
NE/NZ	0	0 1 0	EQ/Z	1	0 1 0
PZ	0	0 1 1	N	1	0 1 1
NO	0	1 0 0	O	1	1 0 0
GT	0	1 0 1	LE	1	1 0 1
GE	0	1 1 0	LT	1	1 1 0

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

BNAND

(1) BNAND src



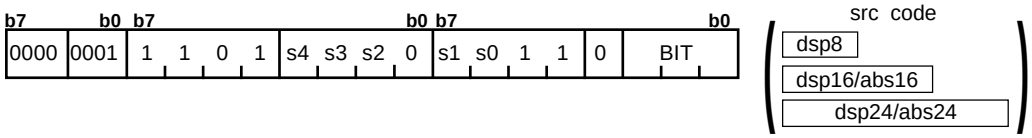
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BNOR

(1) BNOR src



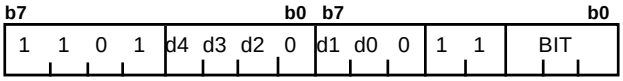
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

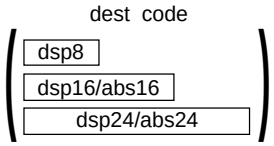
[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BNOT

(1) BNOT dest





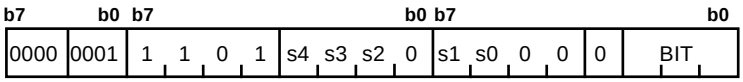
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

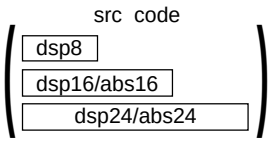
[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

BNTST

(1) BNTST src





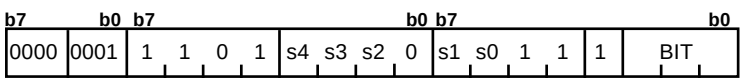
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

BNXOR

(1) BNXOR src



src code
- dsp8
- dsp16/abs16
- dsp24/abs24

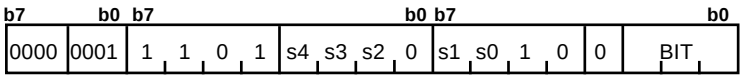
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BOR

(1) BOR src



src code
- dsp8
- dsp16/abs16
- dsp24/abs24

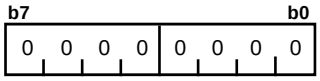
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

BRK

(1) BRK



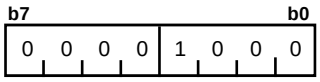
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/17

*1 When you specify the target address of the BRK interrupt by use of the interruput table register (INTB) the number of cycles shown in the table increases by 2. At this time, set FF_{16} in address $FFFFE7_{16}$.

BRK2

(1) BRK2

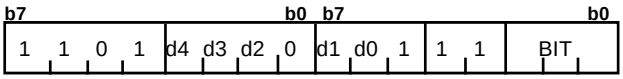


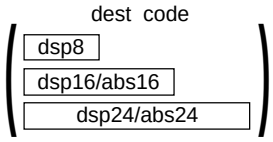
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/19

BSET

(1) BSET dest





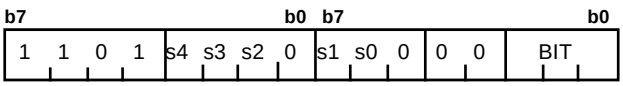
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

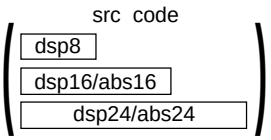
[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

BTST

(1) BTST:G src





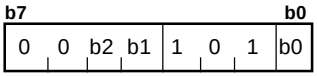
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

BTST

(2) BTST:S src



src code

abs16

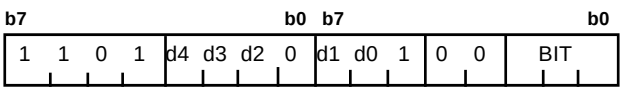
src	bit,base:19

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/3

BTSTC

(1) BTSTC dest



dest code

dsp8
dsp16/abs16
dsp24/abs24

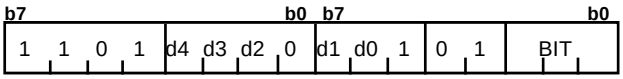
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

BTSTS

(1) BTSTS dest



dest code
- dsp8
- dsp16/abs16
- dsp24/abs24

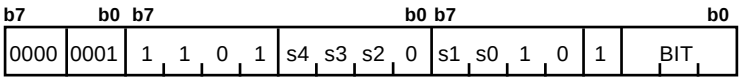
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

BXOR

(1) BXOR src



src code
- dsp8
- dsp16/abs16
- dsp24/abs24

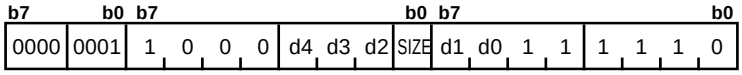
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	bit,R0L	1 0 0 1 0	bit,base:11[SB/FB]	bit,base:11[SB]	0 0 1 1 0
	bit,R0H	1 0 0 0 0		bit,base:11[FB]	0 0 1 1 1
	bit,R1L	1 0 0 1 1	bit,base:19[An]	bit,base:19[A0]	0 1 0 0 0
	bit,R1H	1 0 0 0 1		bit,base:19[A1]	0 1 0 0 1
An	bit,A0	0 0 0 1 0	bit,base:19[SB/FB]	bit,base:19[SB]	0 1 0 1 0
	bit,A1	0 0 0 1 1		bit,base:19[FB]	0 1 0 1 1
[An]	bit,[A0]	0 0 0 0 0	bit,base:27[An]	bit,base:27[A0]	0 1 1 0 0
	bit,[A1]	0 0 0 0 1		bit,base:27[A1]	0 1 1 0 1
bit,base:11[An]	bit,base:11[A0]	0 0 1 0 0	bit,base:19	bit,base:19	0 1 1 1 1
	bit,base:11[A1]	0 0 1 0 1	bit,base:27	bit,base:27	0 1 1 1 0

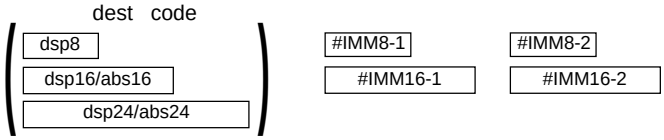
[Number of Bytes/Number of Cycles]

src	bit,Rn	bit,An	bit,[An]	bit,base:11 [An]	bit,base:11 [SB/FB]	bit,base:19 [An]	bit,base:19 [SB/FB]	bit,base:27 [An]	bit,base:19	bit,base:27
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

CLIP

(1) CLIP.size #IMM1, #IMM2, dest





.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

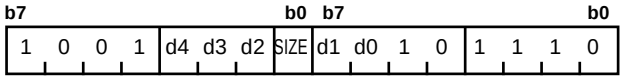
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	5/6	5/6	5/8	6/8	6/8	7/8	7/8	8/8	7/8	8/8

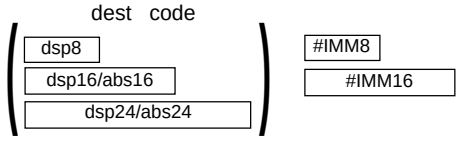
*1 When (.W) is specified for the size specifier (.size) the numberof bytes in the table is increased by 2.

CMP

(1) CMP.size:G #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

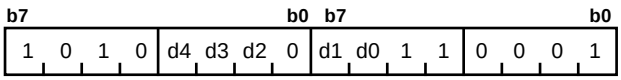
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

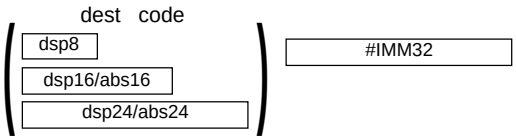
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size), the number of bytes in the table is increased by 1.

CMP

(2) CMP.L:G #IMM32, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

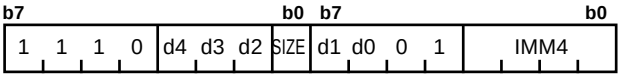
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	6/2	6/2	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

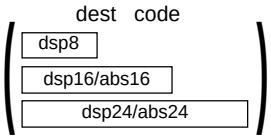
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

CMP

(3) CMP.size:Q #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



.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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

CMP

(4) CMP.size:S #IMM, dest

b7							b0
0	1	d1	d0	0	1	1	SIZE

*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

dest code

dsp8	#IMM8
abs16	#IMM16

.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

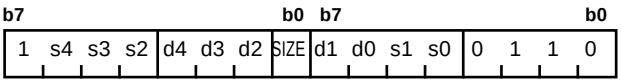
[Number of Bytes/Number of Cycles]

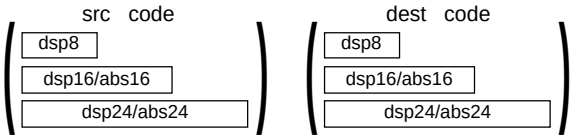
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size), the number of bytes in the table is increased by 1.

CMP

(5) CMP.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

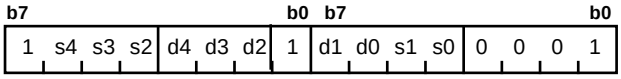
[Number of Bytes/Number of Cycles]

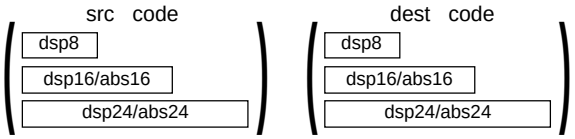
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

CMP

(6) CMP.L:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
An	2/2	2/2	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
[An]	2/5	2/5	2/8	3/8	3/8	4/8	4/8	5/8	4/8	5/8
dsp:8[An]	3/5	3/5	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8
dsp:8[SB/FB]	3/5	3/5	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8
dsp:16[An]	4/5	4/5	4/8	5/8	5/8	6/8	6/8	7/8	6/8	7/8
dsp:16[SB/FB]	4/5	4/5	4/8	5/8	5/8	6/8	6/8	7/8	6/8	7/8
dsp:24[An]	5/5	5/5	5/8	6/8	6/8	7/8	7/8	8/8	7/8	8/8
abs16	4/5	4/5	4/8	5/8	5/8	6/8	6/8	7/8	6/8	7/8
abs24	5/5	5/5	5/8	6/8	6/8	7/8	7/8	8/8	7/8	8/8

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

CMP

(7) CMP.size:S src, R0/R0L

b7							b0
0	1	d1	d0	0	0	0	SIZE

src code

dsp8

abs16

*1 When src is indirectly addressed, the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

src		d1	d0
dsp:8[SB/FB]	dsp:8[SB]	1	0
	dsp:8[FB]	1	1
abs16	abs16	0	1

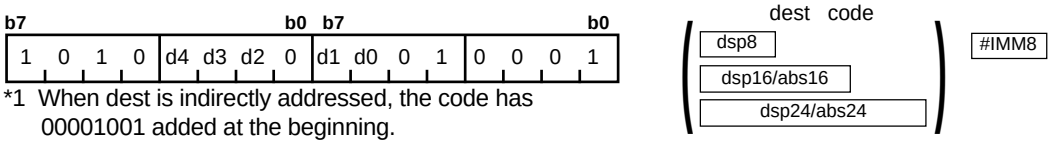
[Number of Bytes/Number of Cycles]

src	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/3	3/3

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

CMPX

(1) CMPX #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	--- / --- /R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	--- / --- /R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	--- / --- /-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	--- / --- /-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

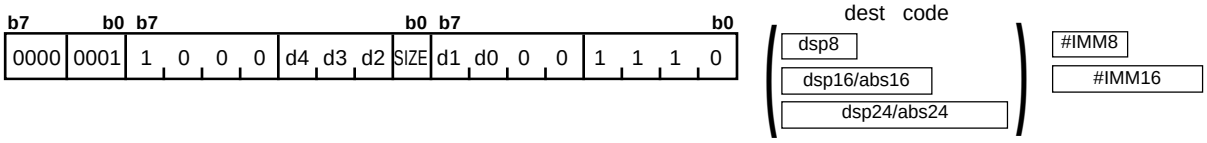
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

DADC

(1) DADC.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

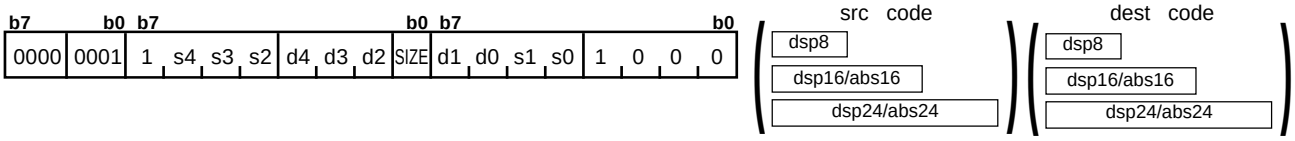
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6

*1 When (.W)is specified for the size specifier(.size), the numberof bytes in the table is increased by 1.

DADC

(2) DADC.size src, dest



.size	SIZE
.B	0
.W	1

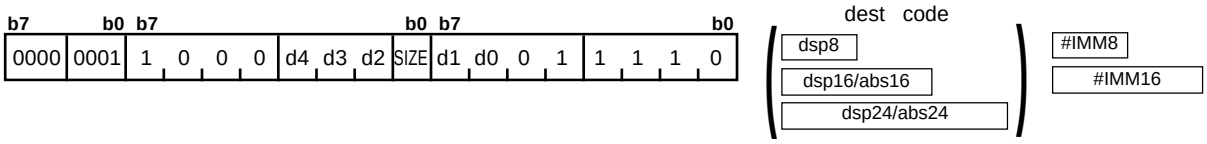
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
An	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
[An]	3/6	3/6	3/7	4/7	4/7	5/7	5/7	6/7	5/7	6/7
dsp:8[An]	4/6	4/6	4/7	5/7	5/7	6/7	6/7	7/7	6/7	7/7
dsp:8[SB/FB]	4/6	4/6	4/7	5/7	5/7	6/7	6/7	7/7	6/7	7/7
dsp:16[An]	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
dsp:16[SB/FB]	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
dsp:24[An]	6/6	6/6	6/7	7/7	7/7	8/7	8/7	9/7	8/7	9/7
abs16	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
abs24	6/6	6/6	6/7	7/7	7/7	8/7	8/7	9/7	8/7	9/7

DADD

(1) DADD.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

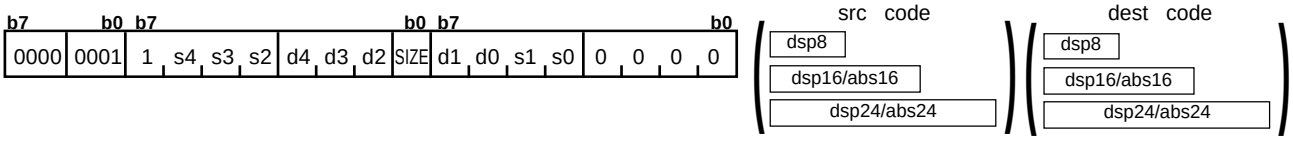
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6

*1 When (.W) is specified for the size specifier (.size), the number of bytes in the table is increased by 1.

DADD

(2) DADD.size src, dest



.size	SIZE
.B	0
.W	1

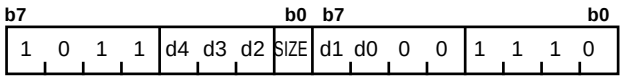
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

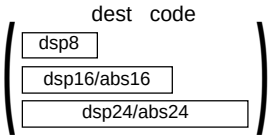
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
An	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
[An]	3/6	3/6	3/7	4/7	4/7	5/7	5/7	6/7	5/7	6/7
dsp:8[An]	4/6	4/6	4/7	5/7	5/7	6/7	6/7	7/7	6/7	7/7
dsp:8[SB/FB]	4/6	4/6	4/7	5/7	5/7	6/7	6/7	7/7	6/7	7/7
dsp:16[An]	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
dsp:16[SB/FB]	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
dsp:24[An]	6/6	6/6	6/7	7/7	7/7	8/7	8/7	9/7	8/7	9/7
abs16	5/6	5/6	5/7	6/7	6/7	7/7	7/7	8/7	7/7	8/7
abs24	6/6	6/6	6/7	7/7	7/7	8/7	8/7	9/7	8/7	9/7

DEC

(1) DEC.size dest



*1 When dest is indirectly addressed,the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

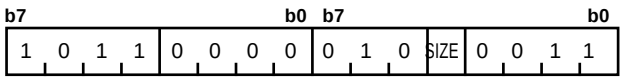
[Number of Bytes/Number of Cycles]

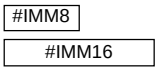
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

DIV

(1) DIV.size #IMM





.size	SIZE
.B	0
.W	1

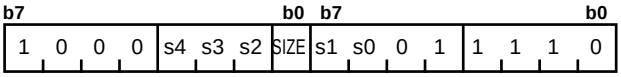
[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/18

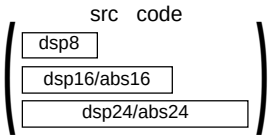
*1 When (.W) is specified for the size specifier (.size), the number of bytes and cycles in the table are increased by 1 and 6, respectively.

DIV

(2) DIV.size src



*1 When src is indirectly addressed, the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

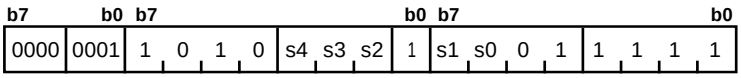
[Number of Bytes/Number of Cycles]

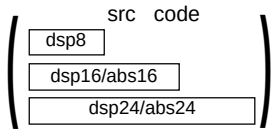
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/18	2/18	2/20	3/20	3/20	4/20	4/20	5/20	4/20	5/20

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size), the number of bytes in the table is increased by 6.

DIV

(3) DIV.L src





src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/ ---/---	- - - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/---	- - - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

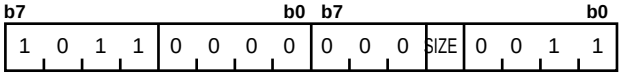
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/40	3/40	3/42	4/42	4/42	5/42	5/42	6/42	5/42	6/42

*1 Indirect instruction addressing cannot be used since op-code is 3 bytes.

DIVU

(1) DIVU.size #IMM



#IMM8
#IMM16

.size	SIZE
.B	0
.W	1

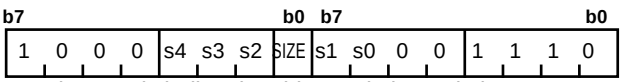
[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/18

*1 When (.W) is specified for the size specifier (.size), the number of bytes and cycles in the table are increased by 1 and 5, respectively.

DIVU

(2) DIV.size src



*1 When src is indirectly addressed, the code has 00001001 added at the beginning.

src code
dsp8
dsp16/abs16
dsp24/abs24

.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

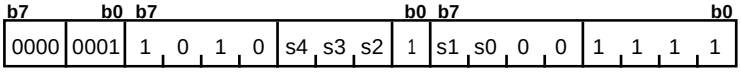
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/18	2/18	2/20	3/20	3/20	4/20	4/20	5/20	4/20	5/20

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size), the number of cycles in the table is increased by 5.

DIVU

(3) DIV.L #IMM



src code
- dsp8
- dsp16/abs16
- dsp24/abs24

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	----------	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	----------	- - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

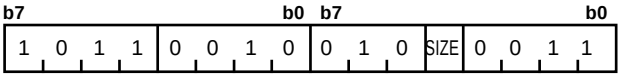
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/40	3/40	3/42	4/42	4/42	5/42	5/42	6/42	5/42	6/42

*1 Indirect instruction addressing cannot be used since op-code is 3 bytes.

DIVX

(1) DIVX.size #IMM



- #IMM8
- #IMM16

.size	SIZE
.B	0
.W	1

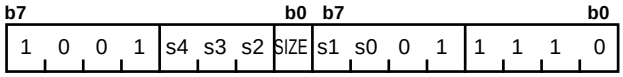
[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/18

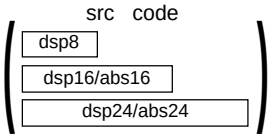
*1 When (.W) is specified for the size specifier (.size), the number of bytes and cycles in the table are increased by 1 and 6, respectively.

DIVX

(2) DIVX.size src



*1 When src is indirectly addressed,the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

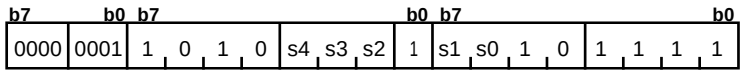
[Number of Bytes/Number of Cycles]

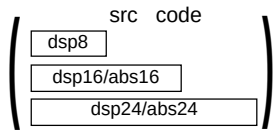
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/18	2/18	2/20	3/20	3/20	4/20	4/20	5/20	4/20	5/20

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier (.size), the number of cycles in the table is increased by 6.

DIVX

(3) DIVX.L src





src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	----------	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	----------	- - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

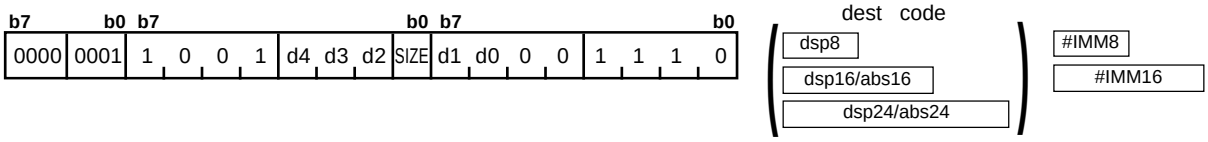
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/40	3/40	3/42	4/42	4/42	5/42	5/42	6/42	5/42	6/42

*1 Indirect instruction addressing cannot be used since op-code is 3 bytes.

DSBB

(1) DSBB.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

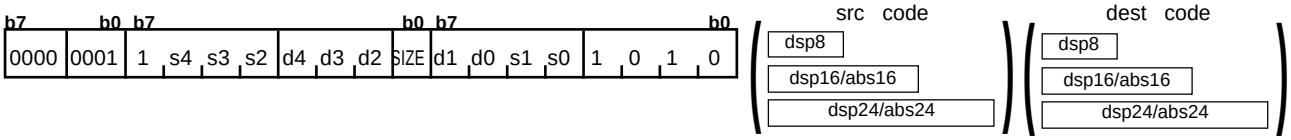
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Btyes/Cycles	4/2	4/2	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4

*1 When (.W) is specified for the size specifier (.size), the numberof bytes in the table is increased by 1.

DSBB

(2) DSBB.size src, dest



.size	SIZE
.B	0
.W	1

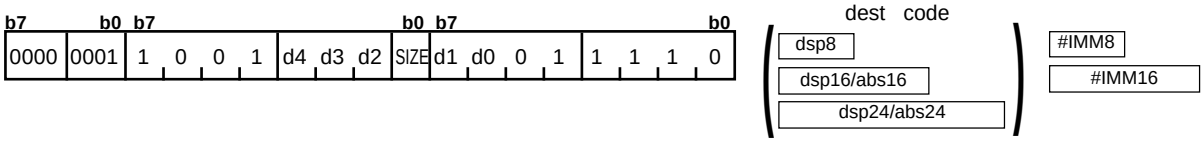
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
An	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
[An]	3/4	3/4	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
dsp:8[An]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:8[SB/FB]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:16[An]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:16[SB/FB]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:24[An]	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5
abs16	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
abs24	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5

DSUB

(1) DSUB.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

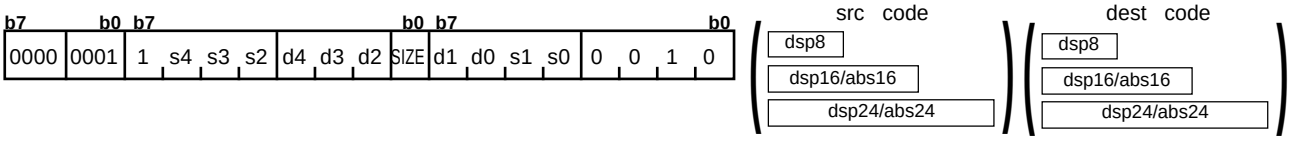
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/2	4/2	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4

*1 When (.W) is specified for the size specifier (.size), the numberof bytes in the table is increased by 1.

DSUB

(2) DSUB.size src, dest



.size	SIZE
.B	0
.W	1

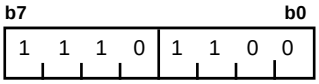
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
An	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
[An]	3/4	3/4	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
dsp:8[An]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:8[SB/FB]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:16[An]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:16[SB/FB]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:24[An]	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5
abs16	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
abs24	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5

ENTER

(1) ENTER #IMM



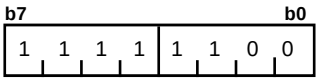
#IMM8

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

EXITD

(1) EXITD

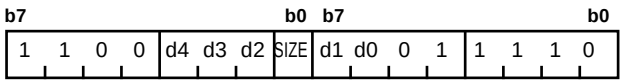


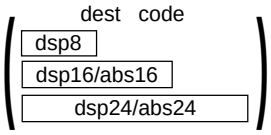
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/8

EXTS

(1) EXTS.size dest





.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

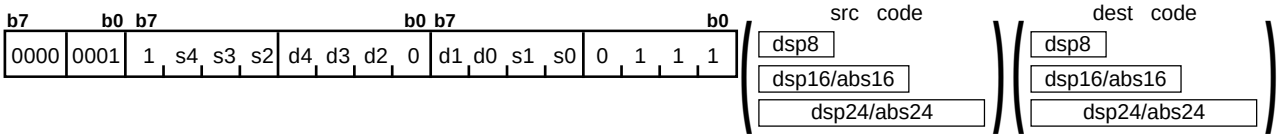
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	5/3	5/3

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

EXTS

(2) EXTS.B src,dest



src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1s0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	---	- - - - - - - -	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---	- - - - - - - -		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

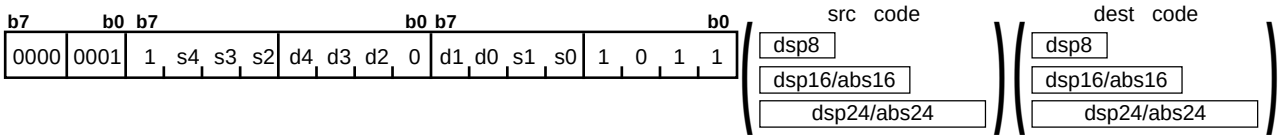
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:8[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:16[SB/FB]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:24[An]	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4
abs16	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs24	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

EXTZ

(1) EXTZ src,dest



src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1s0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	---	- - - - - - - -	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---	- - - - - - - -		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

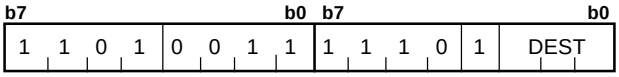
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:8[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:16[SB/FB]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:24[An]	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4
abs16	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs24	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

FCLR

(1) FCLR dest



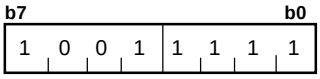
dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

FREIT

(1) FREIT

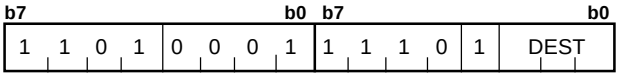


[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

FSET

(1) FSET dest



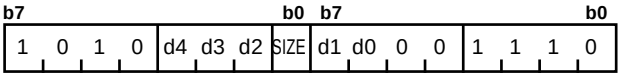
dest	DEST
C	0 0 0
D	0 0 1
Z	0 1 0
S	0 1 1
B	1 0 0
O	1 0 1
I	1 1 0
U	1 1 1

[Number of Bytes/Number of Cycles]

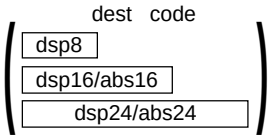
Bytes/Cycles	2/1

INC

(1) INC.size dest



*1 When dest is indirectly addressed,the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

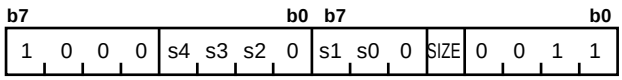
[Number of Bytes/Number of Cycles]

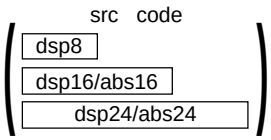
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

INDEXB

(1) INDEXB.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

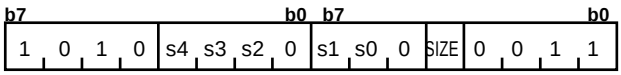
[Number of Bytes/Number of Cycles]

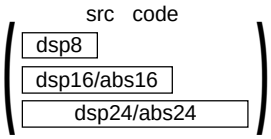
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 2.

INDEXBD

(1) INDEXBD.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

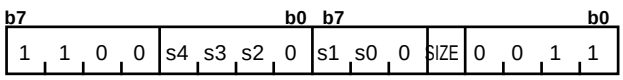
[Number of Bytes/Number of Cycles]

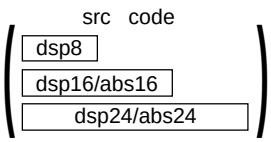
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INDEXBS

(1) INDEXBS.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

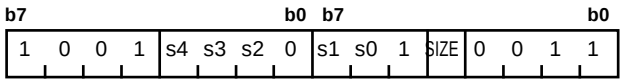
[Number of Bytes/Number of Cycles]

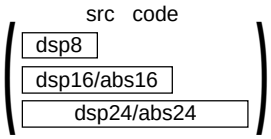
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INDEXL

(1) INDEXL.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

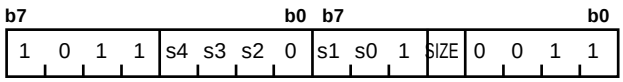
[Number of Bytes/Number of Cycles]

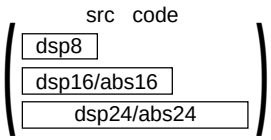
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/4	2/4	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 2.

INDEXLD

(1) INDEXLD.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

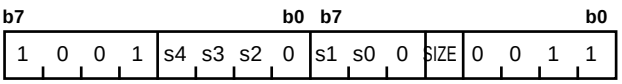
[Number of Bytes/Number of Cycles]

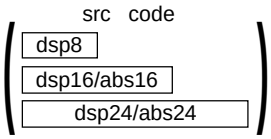
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INDEXLS

(1) INDEXLS.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

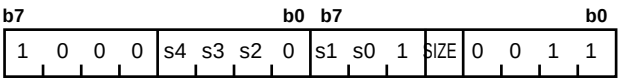
[Number of Bytes/Number of Cycles]

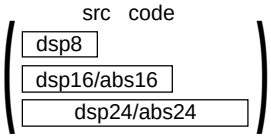
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INDEXW

(1) INDEXW.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

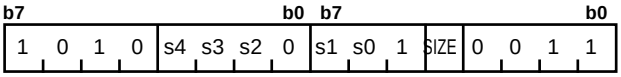
[Number of Bytes/Number of Cycles]

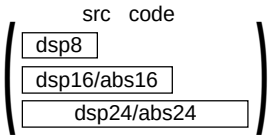
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 2.

INDEXWD

(1) INDEXWD.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

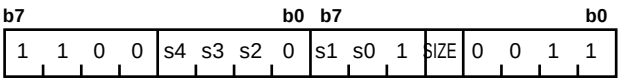
[Number of Bytes/Number of Cycles]

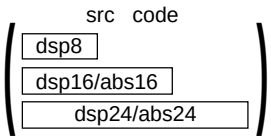
src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INDEXWS

(1) INDEXWS.size src





.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

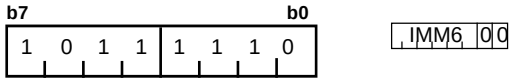
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*1 When (.W) is specified for the size specifier(.size) the number of cycles in the table is increased by 1.

INT

(1) INT #IMM

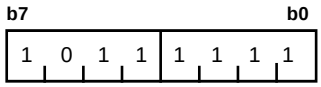


[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/ 12

INTO

(1) INTO



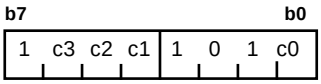
[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/ 1

*1 When O flag is 1, the number of cycles in the table is increased by 13.

J*cnd*

(1) J*cnd* label



labe code

dsp8

dsp8 = address indicated by label - (start address of instruction +1)

Cnd	c3 c2 c1 c0	*Cnd*	c3 c2 c1 c0
LTU/NC	0 0 0 0	GEU/C	1 0 0 0
LEU	0 0 0 1	GTU	1 0 0 1
NE/NZ	0 0 1 0	EQ/Z	1 0 1 0
PZ	0 0 1 1	N	1 0 1 1
NO	0 1 0 0	O	1 1 0 0
GT	0 1 0 1	LE	1 1 0 1
GE	0 1 1 0	LT	1 1 1 0

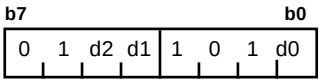
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 When branched to label the number of cycles in the table is increased by 2.

JMP

(1) JMP.S label



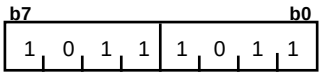
label	d2 d1 d0	label	d2 d1 d0
PC + 2	0 0 0	PC + 6	1 0 0
PC + 3	0 0 1	PC + 7	1 0 1
PC + 4	0 1 0	PC + 8	1 1 0
PC + 5	0 1 1	PC + 9	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/3

JMP

(2) JMP.B label



label code

dsp8

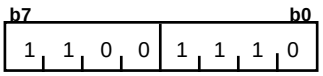
dsp8 = address indicated by label - (start address of instruction +1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

JMP

(3) JMP.W label



label code

dsp16

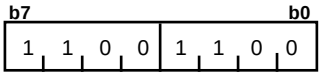
dsp16 = address indicated by label - (start address of instruction +1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/3

JMP

(4) JMP.A label



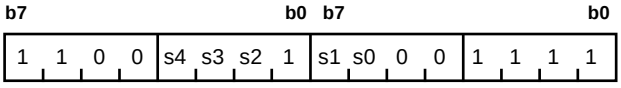
label code
abs24

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/3

JMPI

(1) JMPI.W src



src code
dsp8
dsp16/abs16
dsp24/abs24

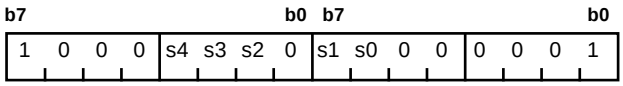
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

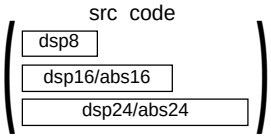
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/7	2/7	2/8	3/8	3/8	4/8	4/8	5/8	4/8	5/8

JMPI

(2) JMPI.A src





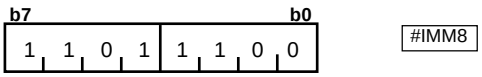
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycle	2/5	2/5	2/7	3/7	3/7	4/7	4/7	5/7	4/7	5/7

JMPS

(1) JMPS #IMM8

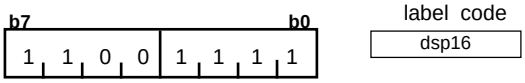


[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/8

JSR

(1) JSR.W label



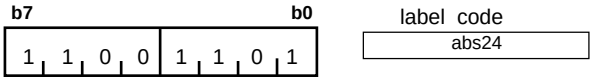
dsp16 = address indicated by label - (start address of instruction +1)

[Number of Bytes/Number of Cycles]

Bytes/Cycles	3/3

JSR

(2) JSR.A label

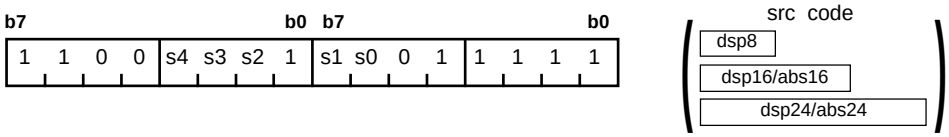


[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/3

JSRI

(1) JSRI.W src



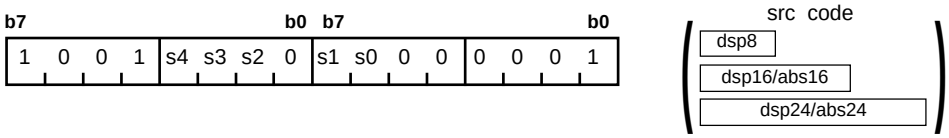
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/7	2/7	2/8	3/8	3/8	4/8	4/8	5/8	4/8	5/8

JSRI

(2) JSRI.A src



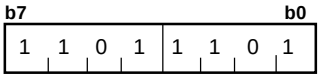
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/5	2/5	2/7	3/7	3/7	4/7	4/7	5/7	4/7	5/7

JSRS

(1) JSRS #IMM8

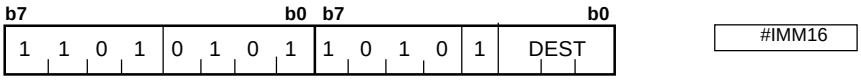


[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/8

LDC

(1) LDC #IMM16, dest



dest	DEST
DCT0	0 0 0
DCT1	0 0 1
FLG	0 1 0
SVF	0 1 1
DRC0	1 0 0
DRC1	1 0 1
DMD0	1 1 0
DMD1	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	4/1

LDC

(2) LDC #IMM24, dest

#IMM24

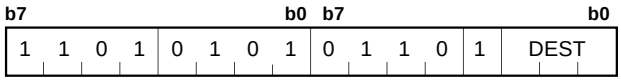
dest	DEST
INTB	0 0 0
SP	0 0 1
SB	0 1 0
FB	0 1 1
SVP	1 0 0
VCT	1 0 1
---	1 1 0
ISP	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	5/2

LDC

(3) LDC #IMM24, dest



#IMM24

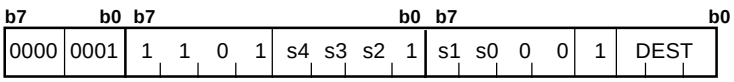
dest	DEST
---	0 0 0
---	0 0 1
DMA0	0 1 0
DMA1	0 1 1
DRA0	1 0 0
DRA1	1 0 1
DSA0	1 1 0
DSA1	1 1 1

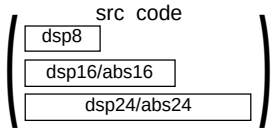
[Number of Bytes/Number of Cycles]

Bytes/Cycles	5/2

LDC

(4) LDC src, dest





src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

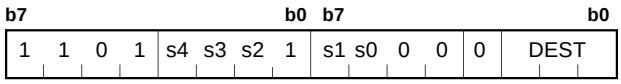
dest	DEST
DCT0	0 0 0
DCT1	0 0 1
FLG	0 1 0
SVF	0 1 1
DRC0	1 0 0
DRC1	1 0 1
DMD0	1 1 0
DMD1	1 1 1

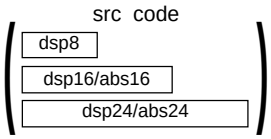
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cyclse	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

LDC

(5) LDC src, dest





src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

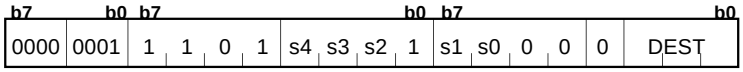
dest	DEST
INTB	0 0 0
SP	0 0 1
SB	0 1 0
FB	0 1 1
SVP	1 0 0
VCT	1 0 1
---	1 1 0
ISP	1 1 1

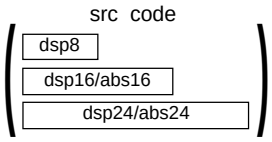
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6

LDC

(6) LDC src, dest





src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

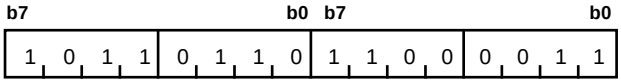
dest	DEST
---	0 0 0
---	0 0 1
DMA0	0 1 0
DMA1	0 1 1
DRA0	1 0 0
DRA1	1 0 1
DSA0	1 1 0
DSA1	1 1 1

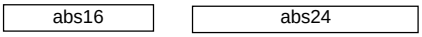
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cyclse	3/2	3/2	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6

LDCTX

(1) LDCTX abs16,abs24





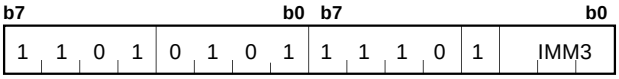
[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/10 + m

*1 m denotes the number of transfers performed.
m = (Number of R0,R1,R2,R3) + 2 x (Number of A0,A1,FB,SB)

LDIPL

(1) LDIPL #IMM

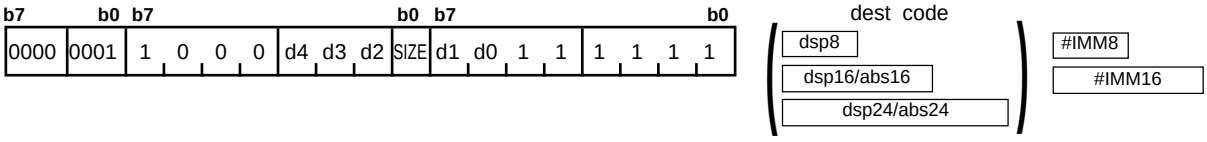


[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2

MAX

(1) MAX.size #IMM,dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

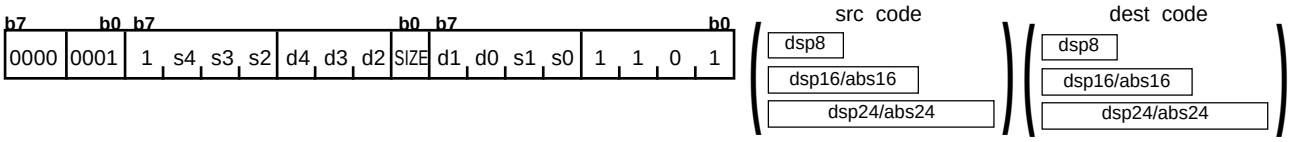
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/3	4/3	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

MAX

(2) MAX.size src, dest



.size	SIZE
.B	0
.W	1

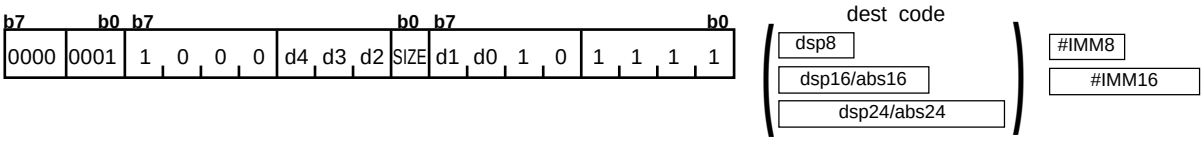
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
An	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
[An]	3/4	3/4	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
dsp:8[An]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:8[SB/FB]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:16[An]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:16[SB/FB]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:24[An]	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5
abs16	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
abs24	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5

MIN

(1) MIN.size #IMM,dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

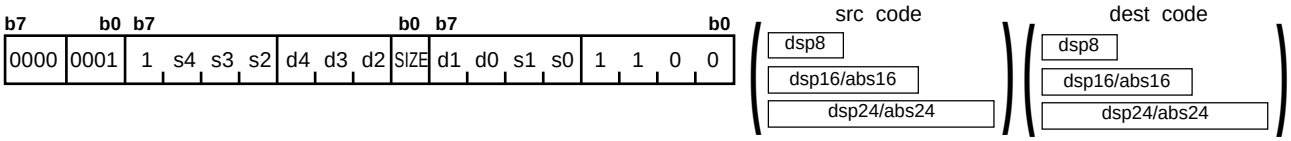
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/3	4/3	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

MIN

(2) MIN.size src, dest



.size	SIZE
.B	0
.W	1

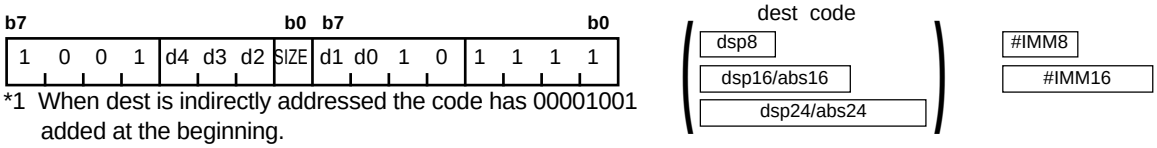
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
An	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
[An]	3/4	3/4	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
dsp:8[An]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:8[SB/FB]	4/4	4/4	4/5	5/5	5/5	6/5	6/5	7/5	6/5	7/5
dsp:16[An]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:16[SB/FB]	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
dsp:24[An]	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5
abs16	5/4	5/4	5/5	6/5	6/5	7/5	7/5	8/5	7/5	8/5
abs24	6/4	6/4	6/5	7/5	7/5	8/5	8/5	9/5	8/5	9/5

MOV

(1) MOV.size:G #IMM,dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

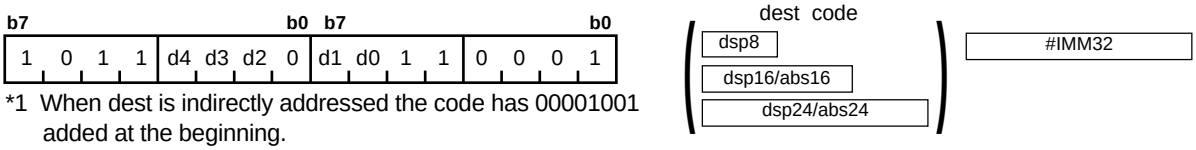
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/2	4/2	4/2	5/2	5/2	6/2	5/2	6/2

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

MOV

(2) MOV.L:G #IMM,dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

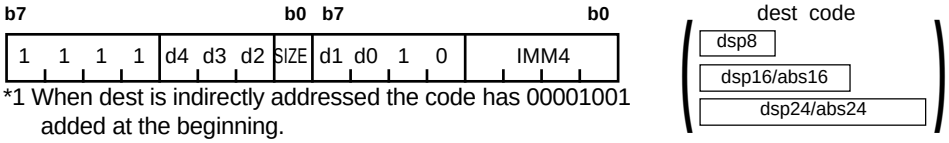
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	6/2	6/2	6/3	7/3	7/3	8/3	8/3	9/3	8/3	9/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(3) MOV.size:Q #IMM4, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

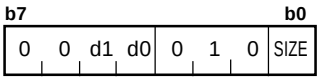
[Number of Bytes/Number of Cycles]

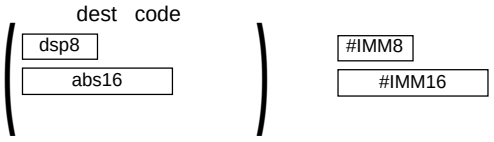
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/1	3/1	3/1	4/1	4/1	5/1	4/1	5/1

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(4) MOV.size:S #IMM, dest





*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

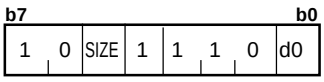
[Number of Bytes/Number of Cycles]

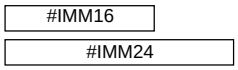
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/2	4/2

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

MOV

(5) MOV.size:S #IMM,A0/A1





.size	SIZE
.W	0
.L	1

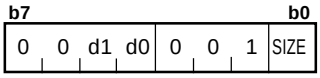
A0/A1	d0
A0	0
A1	1

[Number of Bytes/Number of Cycles]

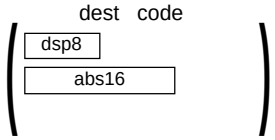
#IMM	An
#IMM16	3/1
#IMM24	4/2

MOV

(6) MOV.size:Z #0, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d1	d0
Rn	R0L/R0	0	0
dsp:8[SB/FB]	dsp:8[SB]	1	0
	dsp:8[FB]	1	1
abs16	abs16	0	1

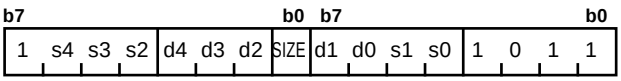
[Number of Bytes/Number of Cycles]

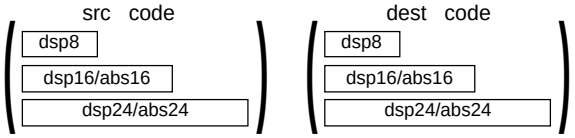
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/1	2/1	3/1

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(7) MOV.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

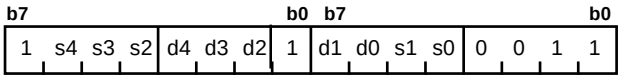
[Number of Bytes/Number of Cycles]

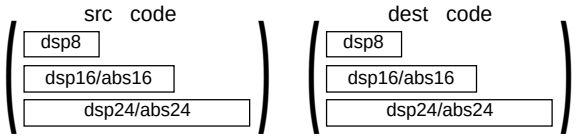
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/1	3/1	3/1	4/1	4/1	5/1	4/1	5/1
An	2/1	2/1	2/1	3/1	3/1	4/1	4/1	5/1	4/1	5/1
[An]	2/3	2/3	2/3	3/2	3/2	4/2	4/2	5/2	4/2	5/2
dsp:8[An]	3/3	3/3	3/3	4/2	4/2	5/2	5/2	6/2	5/2	6/2
dsp:8[SB/FB]	3/3	3/3	3/3	4/2	4/2	5/2	5/2	6/2	5/2	6/2
dsp:16[An]	4/3	4/3	4/3	5/2	5/2	6/2	6/2	7/2	6/2	7/2
dsp:16[SB/FB]	4/3	4/3	4/3	5/2	5/2	6/2	6/2	7/2	6/2	7/2
dsp:24[An]	5/3	5/3	5/3	6/2	6/2	7/2	7/2	8/2	7/2	8/2
abs16	4/3	4/3	4/3	5/2	5/2	6/2	6/2	7/2	6/2	7/2
abs24	5/3	5/3	5/3	6/2	6/2	7/2	7/2	8/2	7/2	8/2

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3, respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

MOV

(8) MOV.L:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

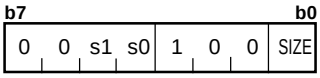
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/2	3/2	3/2	4/2	4/2	5/2	4/2	5/2
An	2/2	2/2	2/2	3/2	3/2	4/2	4/2	5/2	4/2	5/2
[An]	2/4	2/4	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/4	3/4	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/4	3/4	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/4	4/4	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/4	4/4	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/4	5/4	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/4	4/4	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/4	5/4	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

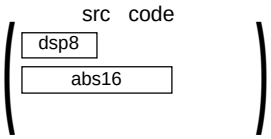
*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

MOV

(9) MOV.size:S src, R0L/R0



*1 When src is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

src		s1	s0
dsp:8[SB/FB]	dsp:8[SB]	1	0
	dsp:8[FB]	1	1
abs16	abs16	0	1

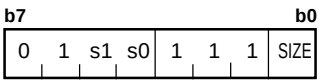
[Number of Bytes/Number of Cycles]

src	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/2	3/2

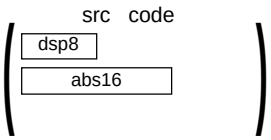
*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(10) MOV.size:S src, R1L/R1



*1 When src is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

src		s1	s0
Rn	R0L/R0	0	0
dsp:8[SB/FB]	dsp:8[SB]	1	0
	dsp:8[FB]	1	1
abs16	abs16	0	1

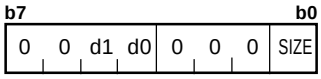
[Number of Bytes/Number of Cycles]

src	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	1/2	2/2	3/2

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3, respectively.

MOV

(11) MOV.size:S R0L/R0, dest



dest code
dsp8
abs16

*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d1 d0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

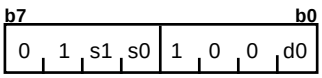
[Number of Bytes/Number of Cycles]

dest	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/1

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(12) MOV.L:S src, A0/A1



src code
dsp8
abs16

*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

src		s1 s0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

A0/A1	d0
A0	0
A1	1

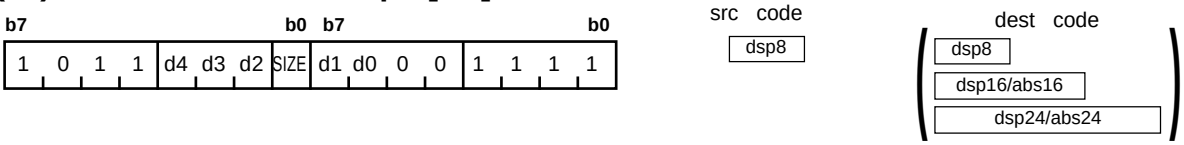
[Number of Bytes/Number of Cycles]

src	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/3	3/3

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MOV

(13) MOV.size:G dsp:8[SP], dest



.size	SIZE
.B	0
.W	1

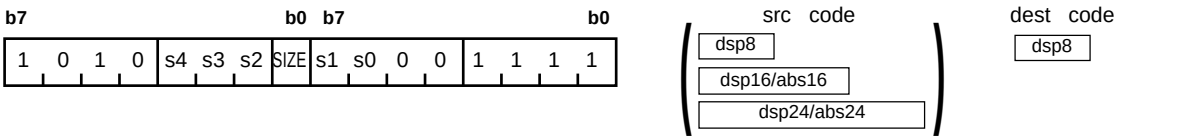
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

MOV

(14) MOV.size:G src, dsp:8[SP]



.size	SIZE
.B	0
.W	1

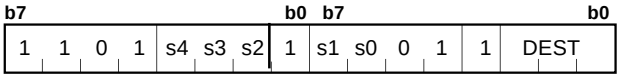
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

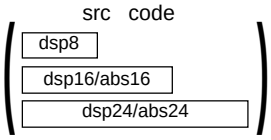
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

MOVA

(1) MOVA src, dest





dest	DEST
R2R0	0 0 0
R3R1	0 0 1
A0	0 1 0
A1	0 1 1

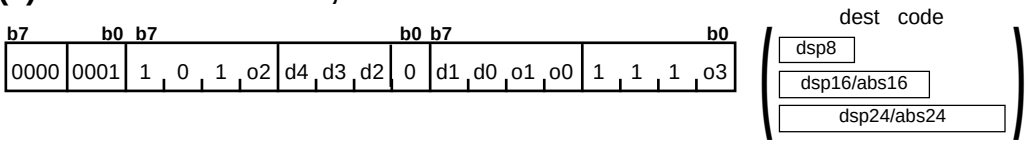
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	dsp:8[A1]	0 0 1 0 1		dsp:16[FB]	0 1 0 1 1
dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	dsp:8[FB]	0 0 1 1 1		dsp:24[A1]	0 1 1 0 1
dsp:16[An]	dsp:16[A0]	0 1 0 0 0	abs16	abs16	0 1 1 1 1
	dsp:16[A1]	0 1 0 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	4/2	4/2	5/2	4/2	5/2

MOV*Dir*

(1) MOV*Dir* R0L, dest



Dir	o3 o2 o1 o0
LL	0 1 0 0
HL	0 1 0 1
LH	0 1 1 0
HH	0 1 1 1

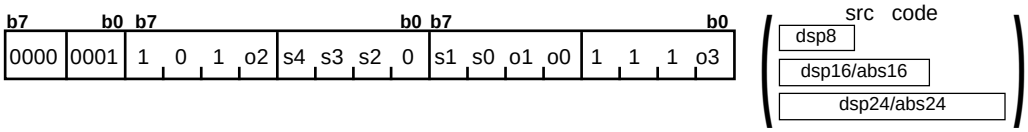
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	---	- - - - - - - -	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---	- - - - - - - -		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
MOVHH, MOVLL	3/3	3/3	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
MOVHL, MOVLH	3/6	3/6	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8

MOV*Dir*

(2) MOV*Dir* src, R0L



Dir	o3 o2 o1 o0
LL	0 0 0 0
HL	0 0 0 1
LH	0 0 1 0
HH	0 0 1 1

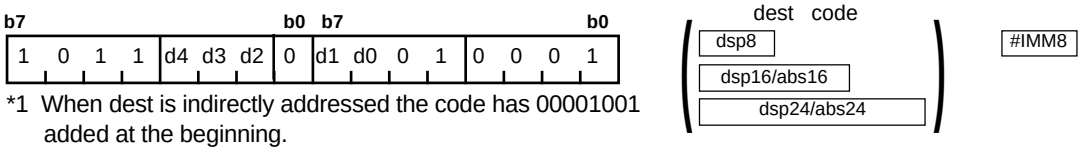
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	---	- - - - - - - -	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---	- - - - - - - -		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
MOVHH, MOVLL	3/3	3/3	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5
MOVHL, MOVLH	3/6	3/6	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8

MOVX

(1) MOVX #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

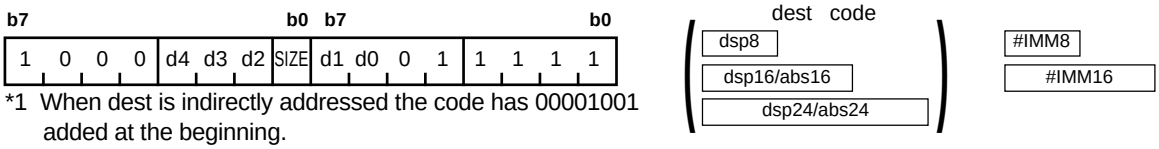
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/2	4/2	4/2	5/2	5/2	6/2	5/2	6/2

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MUL

(1) MUL.size #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

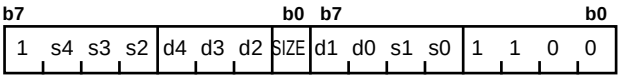
[Number of Bytes/Number of Cycles]

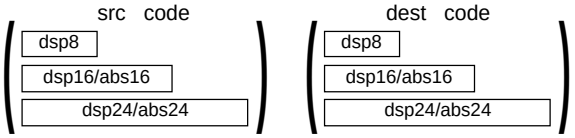
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3, respectively.
*3 When (.W) is specified for the size specifier(.size), the number of bytes in the table is increased by 1.
*4 When (.W) is specified for the size specifier(.size), only Rn and An can be selected for *dest*.

MUL

(2) MUL.size src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

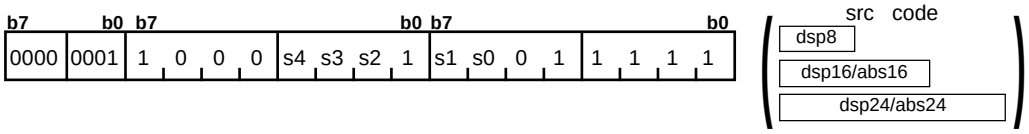
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
An	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
[An]	2/5	2/5	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/5	3/5	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/5	3/5	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/5	5/5	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/5	5/5	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

*3 When (.W) is specified for the size specifier(.size), only Rn and An can be selected for *dest*.

MUL

(3) MUL.L src, R2R0



src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

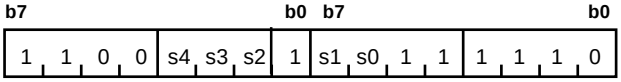
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/8	3/8	3/9	4/9	4/9	5/9	5/9	6/9	5/9	6/9

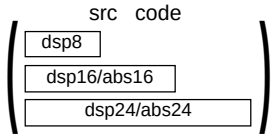
*1 Indirect instruction addressing cannot be used since op-code is 3 bytes.

MULEX

(1) MULEX src



*1 When src is indirectly addressed the code has 00001001 added at the beginning.



src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/---	- - - - -	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/---	- - - - -		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

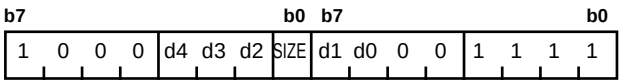
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/8	2/8	2/10	3/10	3/10	4/10	4/10	5/10	4/10	5/10

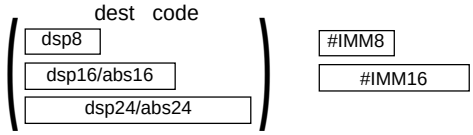
*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

MULU

(1) MULU.size #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

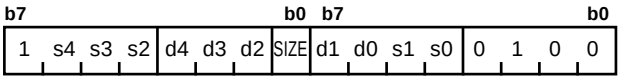
[Number of Bytes/Number of Cycles]

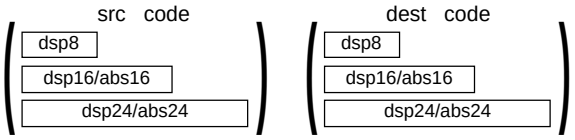
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	3/5	4/5	4/5	5/5	5/5	6/5	5/5	6/5

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3, respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.
*4 When (.W) is specified for the size specifier(.size), only Rn and An can be selected for *dest*.

MULU

(2) MULU.size src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

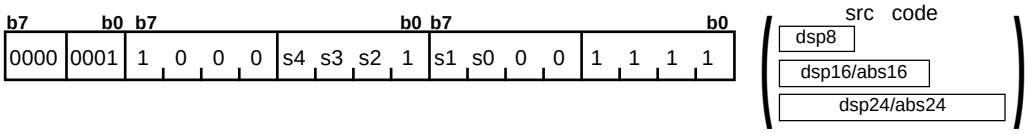
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
An	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5
[An]	2/5	2/5	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/5	3/5	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/5	3/5	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/5	5/5	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/5	4/5	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/5	5/5	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

*3 When (.W) is specified for the size specifier(.size), only Rn and An can be selected for *dest*.

MUL

(3) MULU.L src, R2R0



src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	----------	- - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	----------	- - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

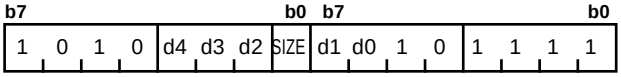
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/8	3/8	3/9	4/9	4/9	5/9	5/9	6/9	5/9	6/9

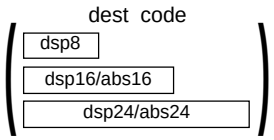
*1 Indirect instruction addressing cannot be used since op-code is 3 bytes.

NEG

(1) NEG.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

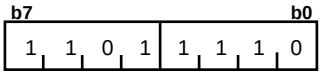
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

NOP

(1) NOP

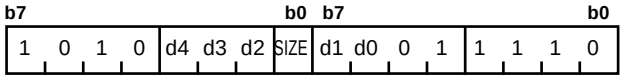


[Number of Bytes/Number of Cycles]

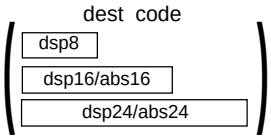
Bytes/Cycles	1/1

NOT

(1)NOT .size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

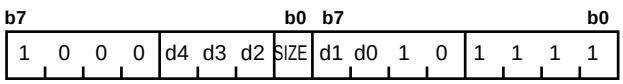
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

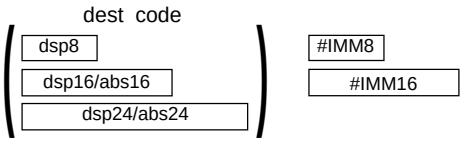
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

OR

(1) OR.size:G #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

OR

(2) OR.size:S #IMM, dest

b7							b0
0	1	d1	d0	0	1	0	SIZE

*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest code

dest	code
dsp8	#IMM8
abs16	#IMM16

.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

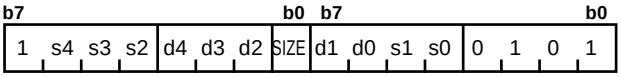
[Number of Bytes/Number of Cycles]

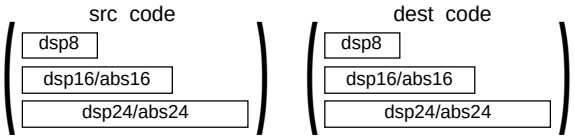
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

OR

(3) OR.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

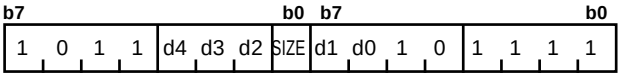
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

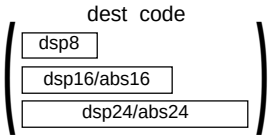
*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

POP

(1) POP.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

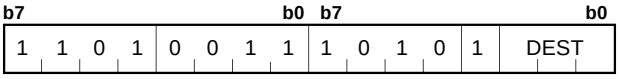
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/3	2/3	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

POPC

(1) POPC dest



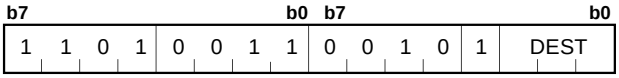
dest	DEST	dest	DEST
DCT0	0 0 0	DRC0	1 0 0
DCT1	0 0 1	DRC1	1 0 1
FLG	0 1 0	DMD0	1 1 0
SVF	0 1 1	DMD1	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/3

POPC

(2) POPC dest



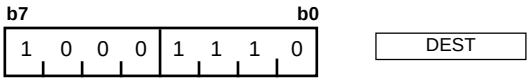
dest	DEST	dest	DEST
INTB	0 0 0	---	1 0 0
SP	0 0 1	---	1 0 1
SB	0 1 0	---	1 1 0
FB	0 1 1	ISP	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

POPM

(1)POPM dest



DEST

dest							
FB	SB	A1	A0	R3	R2	R1	R0
DEST*1							

*1 The bit for a selected register is 1.
The bit for a non-selected register is 0.

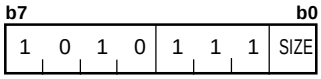
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+m

*2 m denotes the number of register to be restored.
m = (number of R0, R1,R2,R3)+ 2 x (number of A0,A1,FB,SB)

PUSH

(1) PUSH.size #IMM



#IMM8
#IMM16

.size	SIZE
.B	0
.W	1

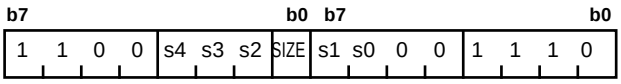
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

PUSH

(2) PUSH.size src



*1 When src is indirectly addressed the code has 00001001 added at the beginning.

src code
dsp8
dsp16/abs16
dsp24/abs24

.size	SIZE
.B	0
.W	1

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

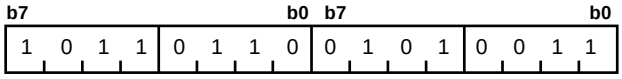
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

PUSH

(3) PUSH.L #IMM32



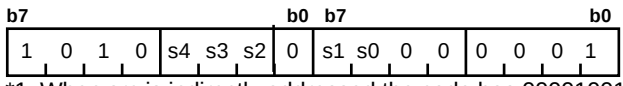
#IMM32

[Number of Bytes/Number of Cycles]

Bytes/Cycles	6/3

PUSH

(4) PUSH.L src



*1 When src is indirectly addressed the code has 00001001 added at the beginning.

src code

dsp8

dsp16/abs16

dsp24/abs24

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1 s0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

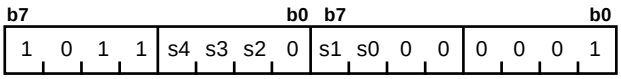
[Number of Bytes/Number of Cycles]

src	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5

*2 When src is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

PUSHA

(1) PUSHA src



src code
- dsp8
- dsp16/abs16
- dsp24/abs24

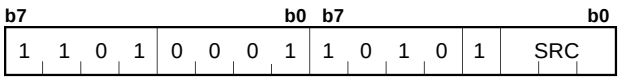
src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1s0
Rn	---/---/---	- - - - -	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/---	- - - - -		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - -		dsp:16[A1]	0 1 0 0 1
An	---	- - - - -	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---	- - - - -		dsp:16[FB]	0 1 0 1 1
[An]	---	- - - - -	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	---	- - - - -		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	4/3	4/3	5/3	4/3	5/3

PUSHC

(1) PUSHC src



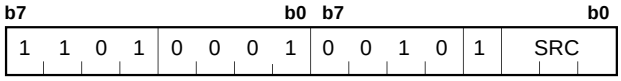
src	SRC	src	SRC
DCT0	0 0 0	DRC0	1 0 0
DCT1	0 0 1	DRC1	1 0 1
FLG	0 1 0	DMD0	1 1 0
SVF	0 1 1	DMD1	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1

PUSHC

(2) PUSHC src



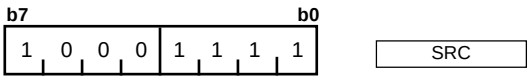
src	SRC	src	SRC
INTB	0 0 0	---	1 0 0
SP	0 0 1	---	1 0 1
SB	0 1 0	---	1 1 0
FB	0 1 1	ISP	1 1 1

[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/4

PUSHM

(1) PUSHM src



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.

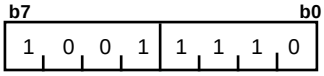
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/m

*2 m denotes the number of registers to be saved.
m = (number of R0,R1,R2,R3)+2x(number of A0,A1,FB,SB)

REIT

(1) REIT

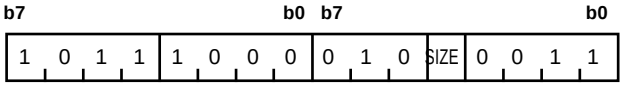


[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

RMPA

(1) RMPA.size



.size	SIZE
.B	0
.W	1

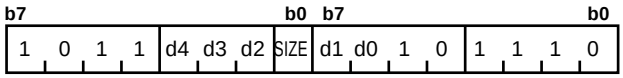
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/7+2m

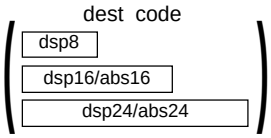
*1 m denotes the number of operations to be performed.

ROLC

(1) ROLC.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

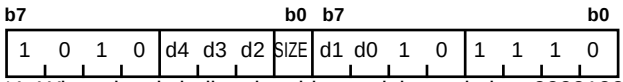
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

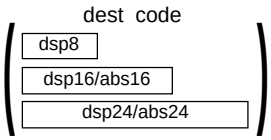
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

RORC

(1) RORC.size dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

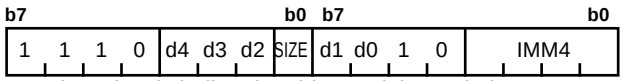
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

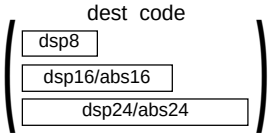
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ROT

(1) ROT.size #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

#IMM	IMM4	dest	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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

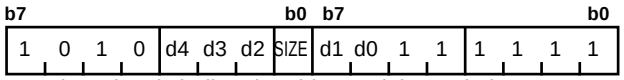
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

ROT

(2) ROT.size R1H, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

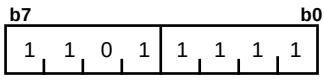
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

RTS

(1) RTS

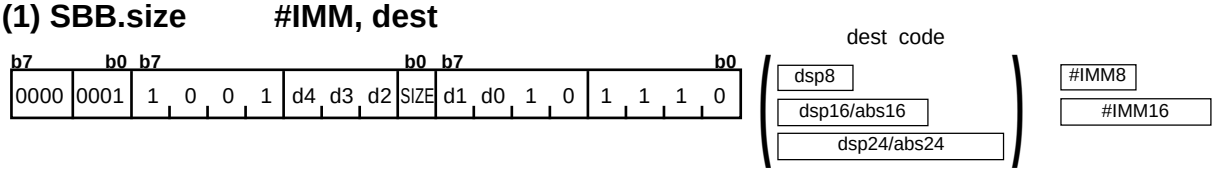


[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/6

SBB

(1) SBB.size #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

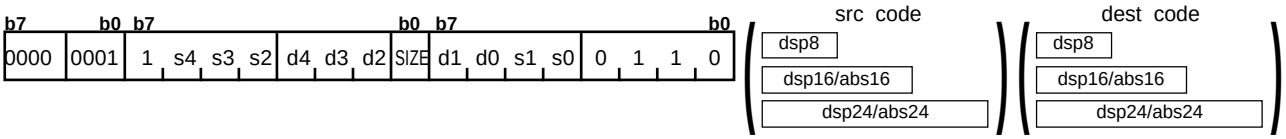
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/1	4/1	4/3	5/3	5/3	6/3	6/3	7/3	6/3	7/3

*1 When (.W) is specified for the size specifier(.size),the number of bytes in the table is increased by 1.

SBB

(2) SBB.size src, dest



.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
An	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:8[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:16[SB/FB]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:24[An]	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4
abs16	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs24	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

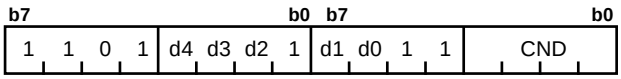
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

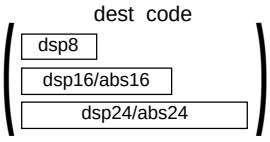
*1 When branched to label the number of cycles in the table is increased by 2.

SC*Cnd*

(1) SC*Cnd* dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



Cnd	CND	*Cnd*	CND
LTU/NC	0 0 0 0	GEU/C	1 0 0 0
LEU	0 0 0 1	GTU	1 0 0 1
NE/NZ	0 0 1 0	EQ/Z	1 0 1 0
PZ	0 0 1 1	N	1 0 1 1
NO	0 1 0 0	O	1 1 0 0
GT	0 1 0 1	LE	1 1 0 1
GE	0 1 1 0	LT	1 1 1 0

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R2/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R3/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	---/A0/---	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	---/A1/---	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

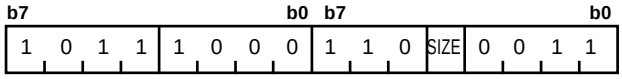
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/1	2/1	2/1	3/1	3/1	4/1	4/1	5/1	4/1	5/1

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SCMPU

(1) SCMPU.size



.size	SIZE
.B	0
.W	1

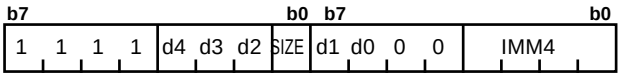
[Number of Bytes/Number of Cycles]

Size specifier	Bytes/Cycles		Remark
	Contents match and the instruction is terminated	Contents do not match and the instruction is terminated	
.B	2/6+3m	2/6+3m	The last 0 (null) is the 8 high-order bits of word
.W	2/6+1.5m	2/9+1.5m	
.W	2/8+1.5m	2/10+1.5m	The last 0(null) is the 8 low-order bits of word

*1 m denotes the number of transfers to be performed.

SHA

(1) SHA.size #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

.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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

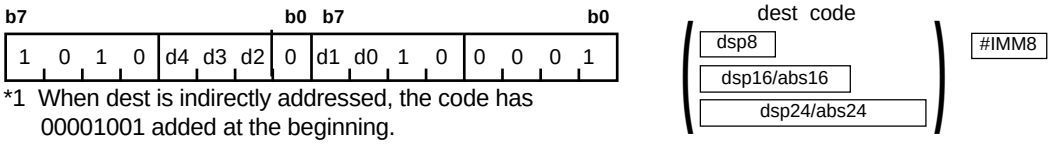
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHA

(2) SHA.L #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

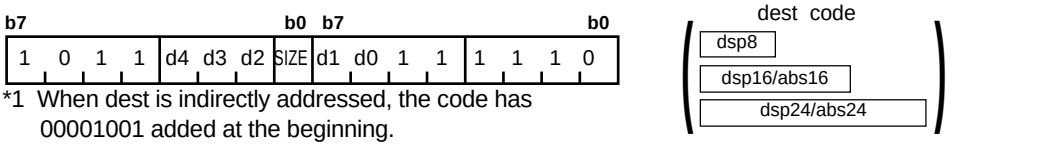
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/6	3/6	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHA

(3) SHA.size R1H, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

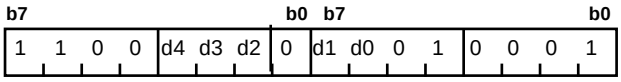
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHA

(4) SHA.L R1H, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

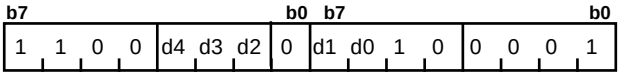
[Number of Bytes/Number of Cycles]

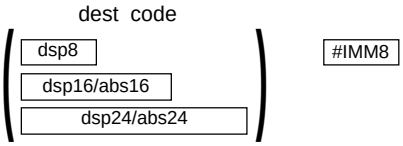
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/6	2/6	2/8	3/8	3/8	4/8	4/8	5/8	4/8	5/8

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHANC

(1) SHANC.L #IMM, dest





*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	----------	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	----------	- - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

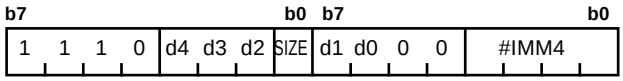
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/4	3/4	3/7	4/7	4/7	5/7	5/7	6/7	5/7	6/7

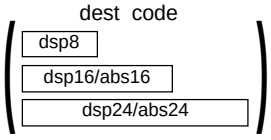
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHL

(1) SHL.size #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

#IMM	IMM4	dest	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		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

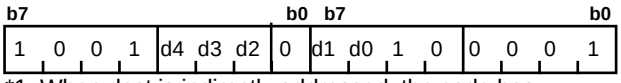
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

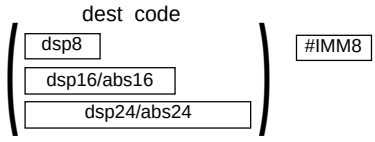
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHL

(2) SHL.L #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

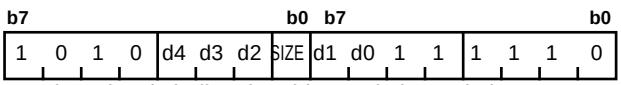
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/6	3/6	3/8	4/8	4/8	5/8	5/8	6/8	5/8	6/8

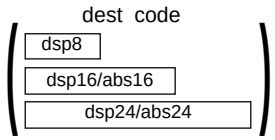
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHL

(3) SHL.size R1H, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

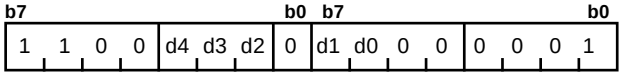
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/3	2/3	2/5	3/5	3/5	4/5	4/5	5/5	4/5	5/5

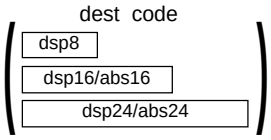
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHL

(4) SHL.L R1H, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

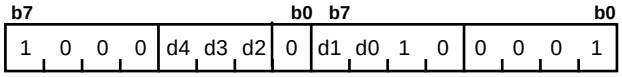
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/6	2/6	2/8	3/8	3/8	4/8	4/8	5/8	4/8	5/8

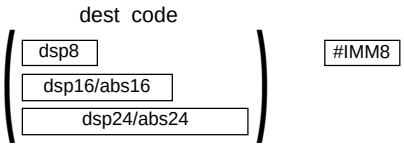
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SHLNC

(1) SHLNC.L #IMM, dest



*1 When dest is indirectly addressed, the code has 00001001 added at the beginning.



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	----------	- - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	----------	- - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

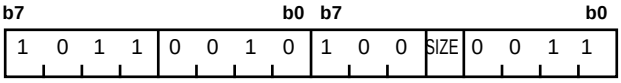
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/4	3/4	3/7	4/7	4/7	5/7	5/7	6/7	5/7	6/7

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SIN

(1) SIN.size



.size	SIZE
.B	0
.W	1

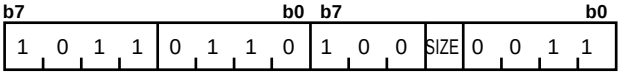
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+2m

*1 m denotes the number of transfers to be performed.

SMOVB

(1) SMOVB.size



.size	SIZE
.B	0
.W	1

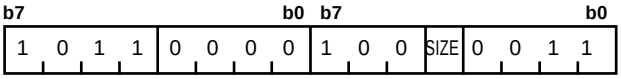
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+2m

*1 m denotes the number of transfers to be performed.

SMOVF

(1) SMOVF.size



.size	SIZE
.B	0
.W	1

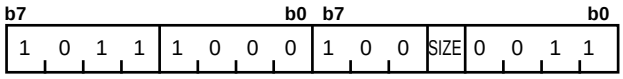
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+2m

*1 m denotes the number of transfers to be performed.

SMOVU

(1) SMOVU.size



.size	SIZE
.B	0
.W	1

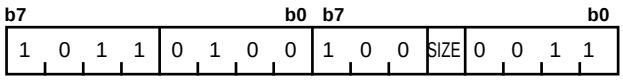
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+2m

*1 m denotes the number of transfers to be performed.

SOUT

(1) SOUT.size



.size	SIZE
.B	0
.W	1

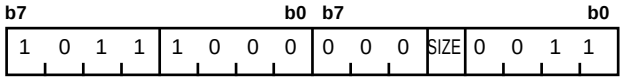
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/1+2m

*1 m denotes the number of transfers to be performed.

SSTR

(1) SSTR.size



.size	SIZE
.B	0
.W	1

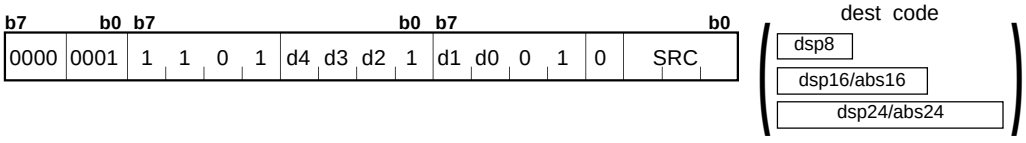
[Number of Bytes/Number of Cycles]

Bytes/Cycles	2/2+m

*1 m denotes the number of transfers to be performed.

STC

(1) STC src, dest



src	SRC
-	000
-	001
DMA0	010
DMA1	011
DRA0	100
DRA1	101
DSA0	110
DSA1	111

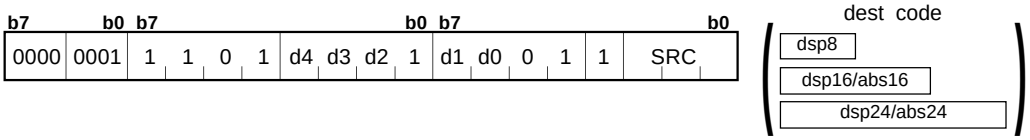
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/3	3/3	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

STC

(2) STC src, dest



src	SRC
DCT0	000
DCT1	001
FLG	010
SVF	011
DRC0	100
DRC1	101
DMD0	110
DMD1	111

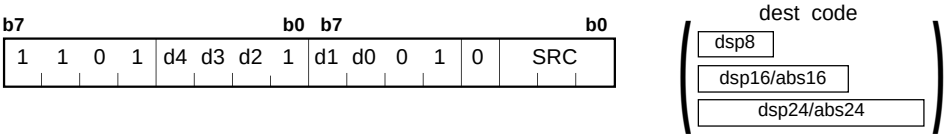
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/2	4/2	4/2	5/2	5/2	6/2	5/2	6/2

STC

(3) STC src, dest



src	SRC
INTB	000
SP	001
SB	010
FB	011
SVP	100
VCT	101
-	110
ISP	111

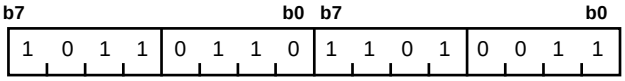
dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	2/3	2/3	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3

STCTX

(1) STCTX abs16, abs24



abs16 abs24

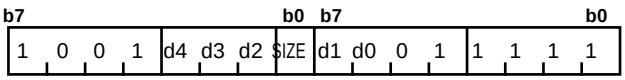
[Number of Bytes/Number of Cycles]

Bytes/Cycles	7/10+2m

*1 m denotes the number of transfers to be performed.

STNZ

(1) STNZ.size #IMM, dest



dest code

dsp8
dsp16/abs16
dsp24/abs24

#IMM8
#IMM16

*1 When dest is indirectly addressed,the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

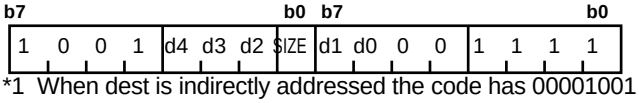
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/2	4/2	4/2	5/2	5/2	6/2	5/2	6/2

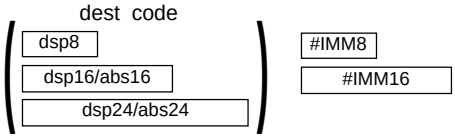
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

STZ

(1) STZ.size #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

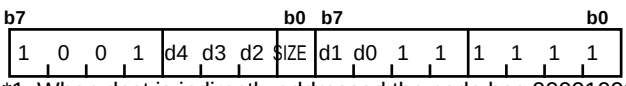
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/2	4/2	4/2	5/2	5/2	6/2	5/2	6/2

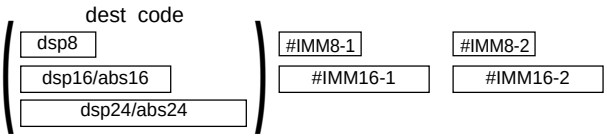
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When Z flag is 0,the number of cycles in the table is increased by 1.
*4 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

STZX

(1) STZX.size #IMM1, #IMM2, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

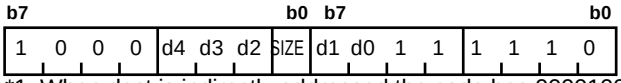
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	4/3	4/3	4/3	5/3	5/3	6/3	6/3	7/3	6/3	7/3

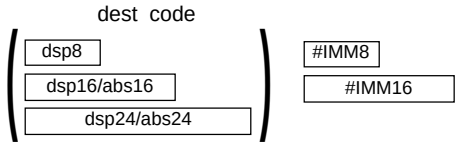
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 2.

SUB

(1) SUB.size:G #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

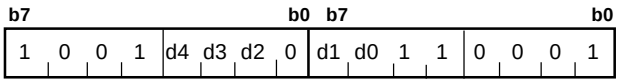
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

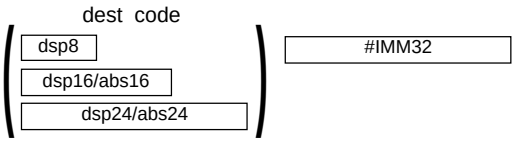
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

SUB

(2) SUB.L:G #IMM, dest



*1 When dest is indirectly addressed,the code has 00001001 added at the beginning.



dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

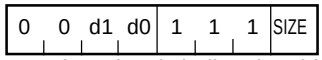
[Number of Bytes/Number of Cycles]

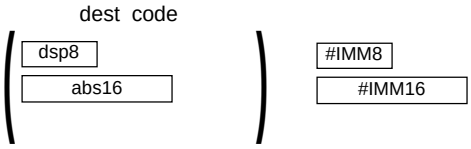
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	6/2	6/2	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SUB

(3) SUB.size:S #IMM, dest





*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

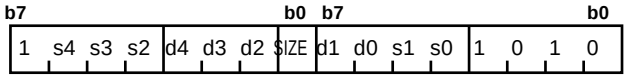
[Number of Bytes/Number of Cycles]

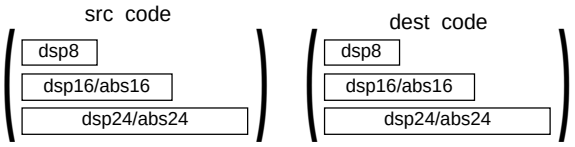
dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.
*3 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

SUB

(4) SUB.size:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

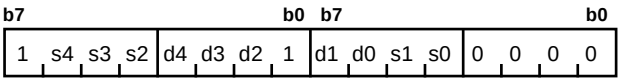
[Number of Bytes/Number of Cycles]

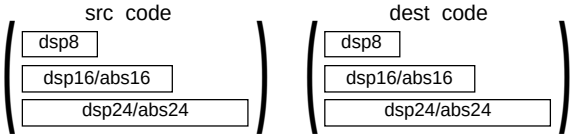
src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

SUB

(5) SUB.L:G src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

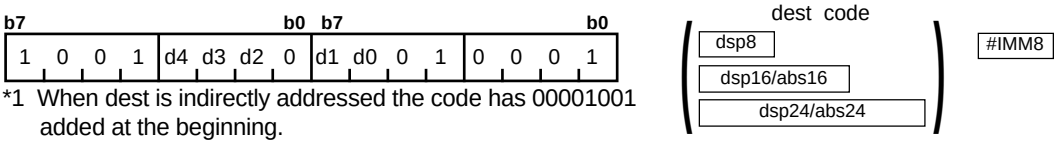
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
An	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
[An]	2/4	2/4	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

SUBX

(1) SUBX #IMM, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

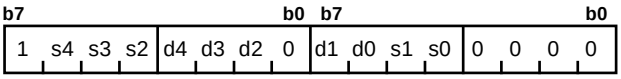
[Number of Bytes/Number of Cycles]

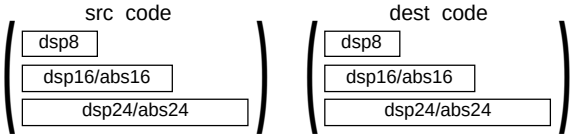
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/2	3/2	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4

*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

SUBX

(2) SUBX src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

src		s4 s3 s2 s1 s0	src		s4 s3 s2 s1s0
Rn	R0L/---/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/---/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/---/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/---/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

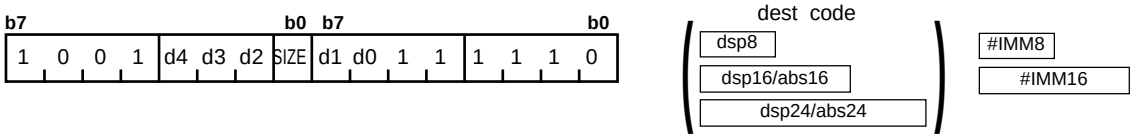
[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
An	2/2	2/2	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
[An]	2/4	2/4	2/6	3/6	3/6	4/6	4/6	5/6	4/6	5/6
dsp:8[An]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:8[SB/FB]	3/4	3/4	3/6	4/6	4/6	5/6	5/6	6/6	5/6	6/6
dsp:16[An]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:16[SB/FB]	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
dsp:24[An]	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6
abs16	4/4	4/4	4/6	5/6	5/6	6/6	6/6	7/6	6/6	7/6
abs24	5/4	5/4	5/6	6/6	6/6	7/6	7/6	8/6	7/6	8/6

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

TST

(1) TST.size:G #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

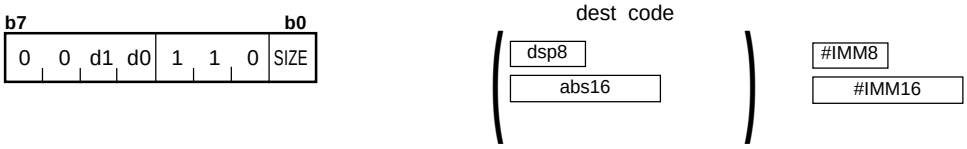
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

TST

(2) TST.size:S #IMM, dest



.size	SIZE
.B	0
.W	1

dest		d1 d0
Rn	R0L/R0	0 0
dsp:8[SB/FB]	dsp:8[SB]	1 0
	dsp:8[FB]	1 1
abs16	abs16	0 1

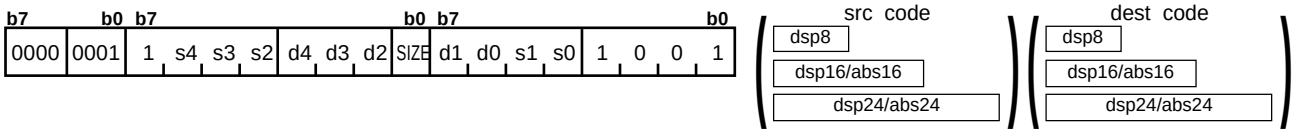
[Number of Bytes/Number of Cycles]

dest	Rn	dsp:8[SB/FB]	abs16
Bytes/Cycles	2/1	3/3	4/3

*1 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

TST

(3) TST.size:G src, dest



.size	SIZE
.B	0
.W	1

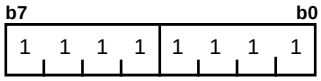
src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
An	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3
[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:8[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:16[SB/FB]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
dsp:24[An]	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4
abs16	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs24	6/3	6/3	6/4	7/4	7/4	8/4	8/4	9/4	8/4	9/4

UND

(1) UND

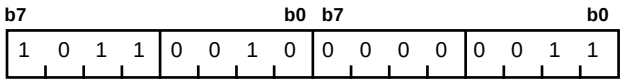


[Number of Bytes/Number of Cycles]

Bytes/Cycles	1/13

WAIT

(1) WAIT

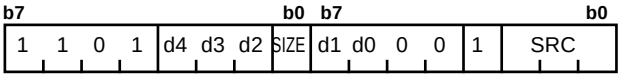


[Number of Bytes]

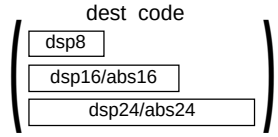
Bytes	2

XCHG

(1) XCHG.size src, dest



*1 When dest is indirectly addressed the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

src	SRC
R0L/R0/---	0 0 0
R1L/R1/---	0 0 1
R0H/R2/-	1 0 0
R1H/R3/-	1 0 1
A0	0 1 0
A1	0 1 1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

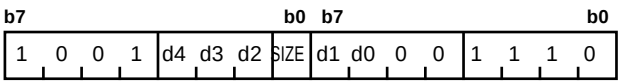
[Number of Bytes/Number of Cycles]

dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/cycles	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4

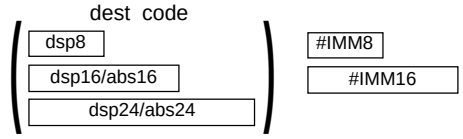
*2 When dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively.

XOR

(1) XOR.size #IMM, dest



*1 When dest is indirectly addressed,the code has 00001001 added at the beginning.



.size	SIZE
.B	0
.W	1

dest		d4 d3 d2 d1 d0	dest		d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

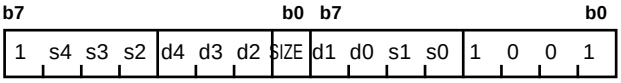
[Number of Bytes/Number of Cycles]

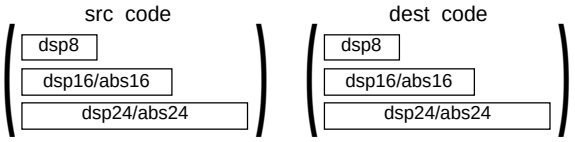
dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Bytes/Cycles	3/1	3/1	3/3	4/3	4/3	5/3	5/3	6/3	5/3	6/3

*2 When (.W) is specified for the size specifier(.size) the number of bytes in the table is increased by 1.

XOR

(2) XOR.size src, dest





*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

[Number of Bytes/Number of Cycles]

src \ dest	Rn	An	[An]	dsp:8[An]	dsp:8[SB/FB]	dsp:16[An]	dsp:16[SB/FB]	dsp:24[An]	abs16	abs24
Rn	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
An	2/1	2/1	2/3	3/3	3/3	4/3	4/3	5/3	4/3	5/3
[An]	2/3	2/3	2/4	3/4	3/4	4/4	4/4	5/4	4/4	5/4
dsp:8[An]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:8[SB/FB]	3/3	3/3	3/4	4/4	4/4	5/4	5/4	6/4	5/4	6/4
dsp:16[An]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:16[SB/FB]	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
dsp:24[An]	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4
abs16	4/3	4/3	4/4	5/4	5/4	6/4	6/4	7/4	6/4	7/4
abs24	5/3	5/3	5/4	6/4	6/4	7/4	7/4	8/4	7/4	8/4

*2 When src or dest is indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 3 respectively. Also, when src and dest both are indirectly addressed, the number of bytes and cycles in the table are increased by 1 and 6, respectively.

Chapter 5

Interrupt

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 and 5.1.2 list the source of interrupts (nonmaskable) and the fixed vector tables.

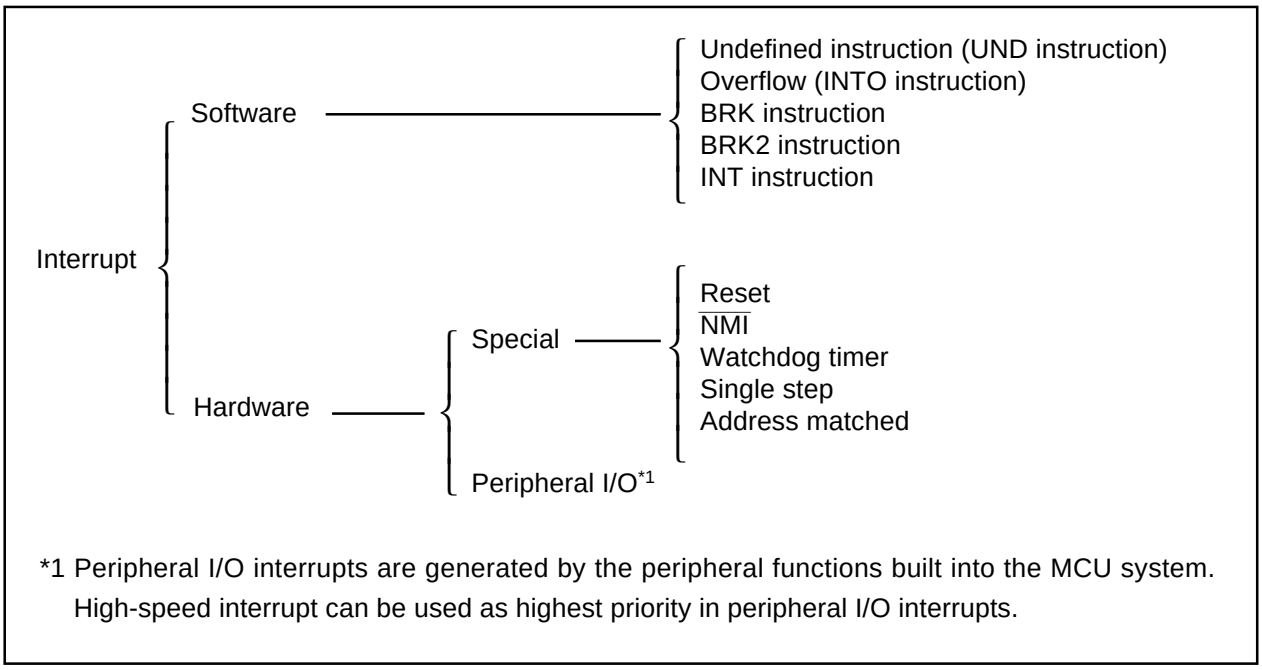


Figure 5.1.1. Classification of interrupts

Table 5.1.1 Interrupt Source (Nonmaskable) and Fixed Vector Table

Interrupt source	Vector table addresses Address (L) to address (H)	Remarks
Undefined instruction	FFFFDC16 to FFFFDF16	Interrupt generated by the UND instruction.
Overflow	FFFFE016 to FFFFE316	Interrupt generated by the INTO instruction.
BRK instruction	FFFFE416 to FFFFE716	Executed beginning from address indicated by vector in variable vector table if content of address FFFFE716 is FF16.
Address match	FFFFE816 to FFFFEB16	Can be controlled by an interrupt enable bit.
Watchdog timer	FFFFF016 to FFFFF316	
$\overline{\text{NMI}}$	FFFFF816 to FFFFFB16	External interrupt generated by driving $\overline{\text{NMI}}$ pin low.
Reset	FFFFFC16 to FFFFFF16	

Table 5.1.2 Interrupt Exclusively for Emulator (Nonmaskable) and Vector Table

Interrupt source	Vector table addresses Address (L) to address (H)	Remarks
BRK2 instruction	Interrupt vector table register exclusively for emulator 00002016 to 00002316	This interrupt is used exclusively for debugger purposes.
Single step		

- 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, ADDX, CMP, CMPX, DIV, DIVU, DIVX, NEG, RMPA, SBB, SCMPU, SHA, SUB, SUBX

(3) BRK interrupt

This interrupt occurs when the BRK instruction is executed.

(4) BRK2 interrupt

This interrupt occurs when the BRK2 instruction is executed. This interrupt is used exclusively for debugger purposes. You normally do not need to use this interrupt.

(5) 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 54 and 57 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.

However, in peripheral I/O interrupts, the U flag is saved when an interrupt occurs and the U flag is cleared to 0 to choose ISP.

Therefore movement of U flag is different by peripheral I/O interrupt or INT instruction in software interrupt number 32 to 54 and 57.

5.1.3 Hardware Interrupts

There are Two types in hardware Interrupts; special interrupts and Peripheral 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.
- **Watchdog timer interrupt**
  This interrupt is caused by the watchdog timer.
- **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.
- **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.

(2) Peripheral I/O interrupts

These interrupts are generated by the on-chip peripheral functions in the MCU system. The types of on-chip peripheral functions vary with each M32C model, so do the types of interrupt causes. The interrupt vector table uses the same software interrupt numbers 0–54 and 57 that are used by the INT instruction. Peripheral I/O interrupts are maskable interrupts. For details about peripheral I/O interrupts, refer to the M32C User's Manual.

For peripheral I/O interrupts, 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.

(3) High-speed interrupts

High-speed interrupts are interrupts in which the response is executed at high-speed. High-speed interrupt can be used as highest priority in peripheral I/O interrupts.

Execute a FREIT instruction to return from the high-speed interrupt routine.

For details about high-speed interrupt, refer to the M32C 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 nonmaskable 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 M32C 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.

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 or FREIT instruction, the changed status takes effect beginning with that REIT or FREIT instruction.
- If the flag is changed by an FCLR, FSET, POPC, or LDC instruction, the changed status takes effect beginning with the next instruction.

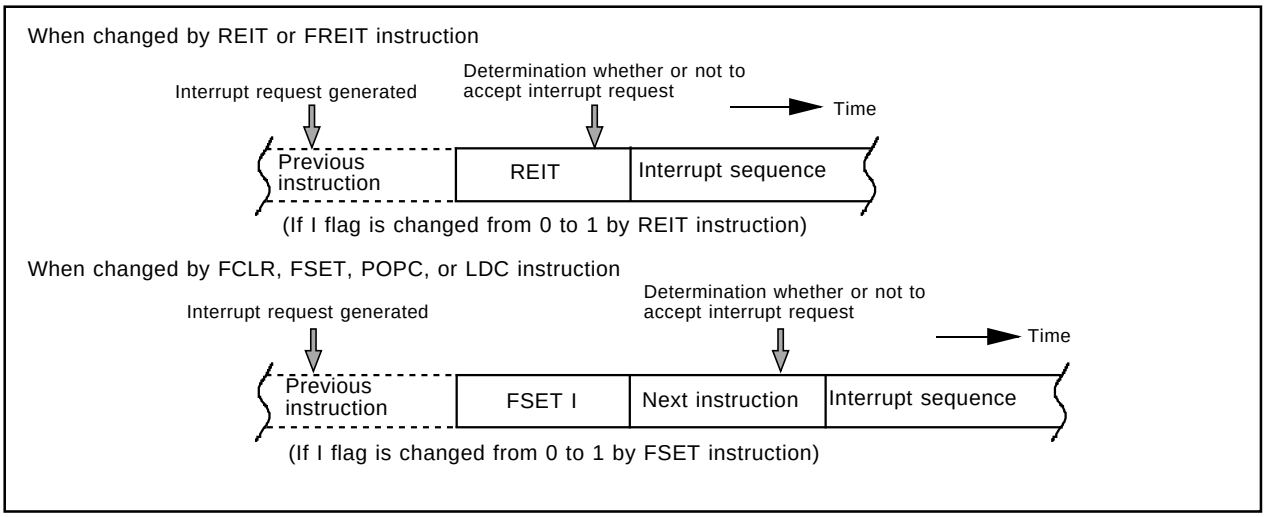


Figure 5.2.1 Timing at which changes of I flag are reflected in interrupt handling

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 or FREIT instruction, the changed level takes effect beginning with the REIT or FREIT 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 next instruction.
- 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 two clock periods after the last clock of the instruction used.

5.2.4 Rewrite the interrupt control register

When a instruction to rewrite the interrupt control register is executed but the interrupt is disabled, the interrupt request bit is not set sometimes even if the interrupt request for that register has been generated. This will depend on the instruction. If this creates problems, use the below instructions to change the register.

Instructions : AND, OR, BCLR, BSET

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 SCMPU, SIN, SMOVB, SMOVF, SMOVU, SSTR, SOUT 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 000000_{16} (address 000002_{16} when high-speed interrupt).

(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. Saves in the flag save register (SVF) in high-speed interrupt.

(5) Saves the content of the program counter (PC) in the stack area. Saves in the PC save register (SVP) in high-speed interrupt.

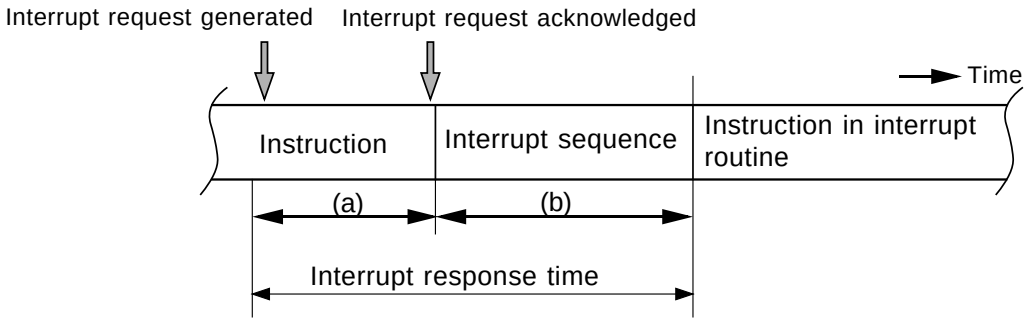
(6) Sets the interrupt priority level of the accepted instruction in the IPL.

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.



(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 29* cycles.
Time (b) is shown in table 5.3.1.

* It is when the divider is immediate or register. When the divider is memory, the following value is added.
  - General instruction addressing : 2 + X
  - Indirect instruction addressing : 5 + X + 2Y

  X is number of wait of the divider area. Y is number of wait of the indirect address stored area. When X and Y are in odd address or in 8 bits bus area, double the value of X and Y. When above addressing modes are modified by the INDEX instructions, add 1 cycle.

Table 5.3.1 Interrupt Sequence Execution Time

Interrupt	Interrupt vector address	16 bits data bus	8 bits data bus
Peripheral I/O	Even address	14 cycles	16 cycles
	Odd address*2	16 cycles	16 cycles
INT instruction	Even address	12 cycles	14 cycles
	Odd address*2	14 cycles	14 cycles
$\overline{\text{NMI}}$ Watchdog timer Undefined instruction Address match	Even address*1	13 cycles	15 cycles
Overflow	Even address*1	14 cycles	16 cycles
BRK instruction	Even address	17 cycles	19 cycles
(Variable vector table)	Odd address*2	19 cycles	19 cycles
Single step BRK2 instruction BRK instruction (Fixed vector table)	Even address*1	19 cycles	21 cycles
High-speed interrupt*3	Vector table is internal register	5 cycles	

*1 The vector table is fixed to even address.
*2 Allocate interrupt vector addresses in even addresses as must as possible.
*3 The high-speed interrupt is independent of these conditions.

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 FLG register is saved to the stack area. Next, the 16 high-order bits and 16 low-order bits of the program counter expanded to 32-bit 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.

In a high-speed interrupt sequence, the contents of the flag register (FLG) is saved to the flag save register (SVF) and program counter (PC) is saved to PC save register (SVP).

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.

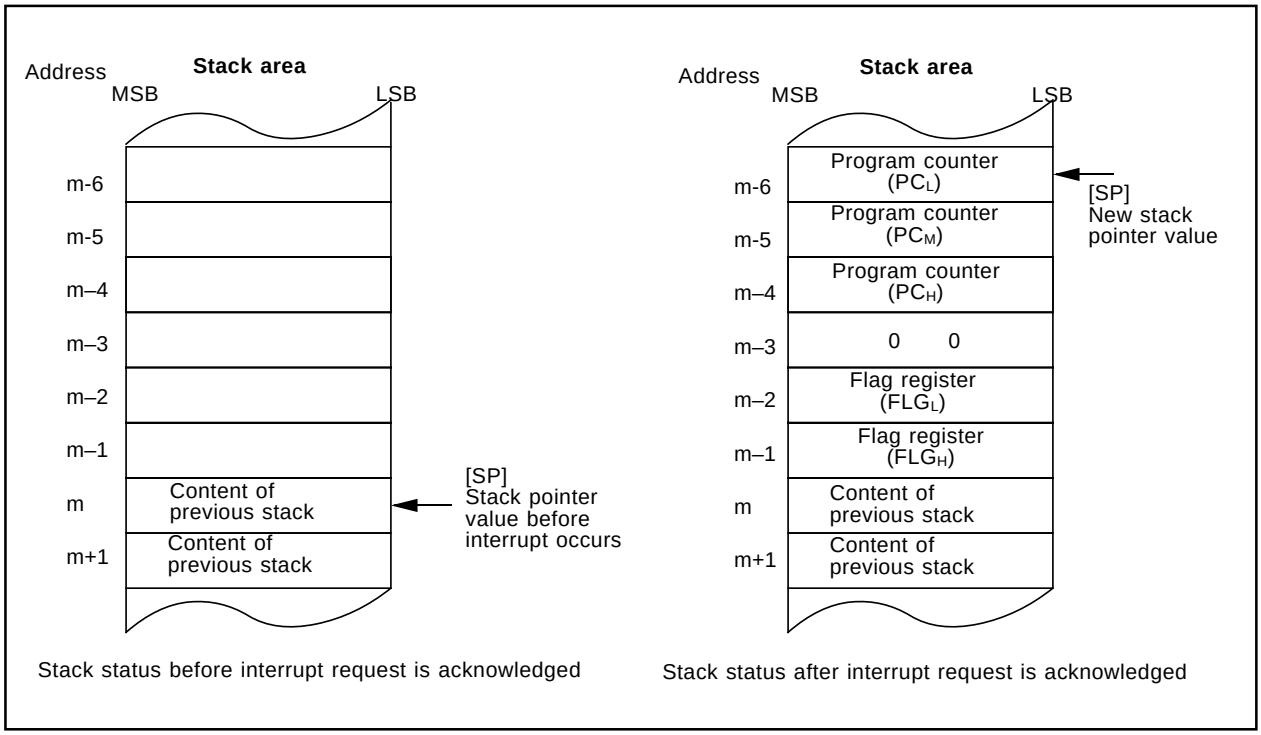


Figure 5.3.2 Stack status before and after an interrupt request is acknowledged

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. In high-speed interrupt, 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 save registers 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 or FREIT 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 interrupt that a request came to most in the first place is accepted at first, and then, 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 M32C model. Please refer to your M32C User's Manual.

Reset > $\overline{\text{NMI}}$ > Watchdog > 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 and FREIT 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)

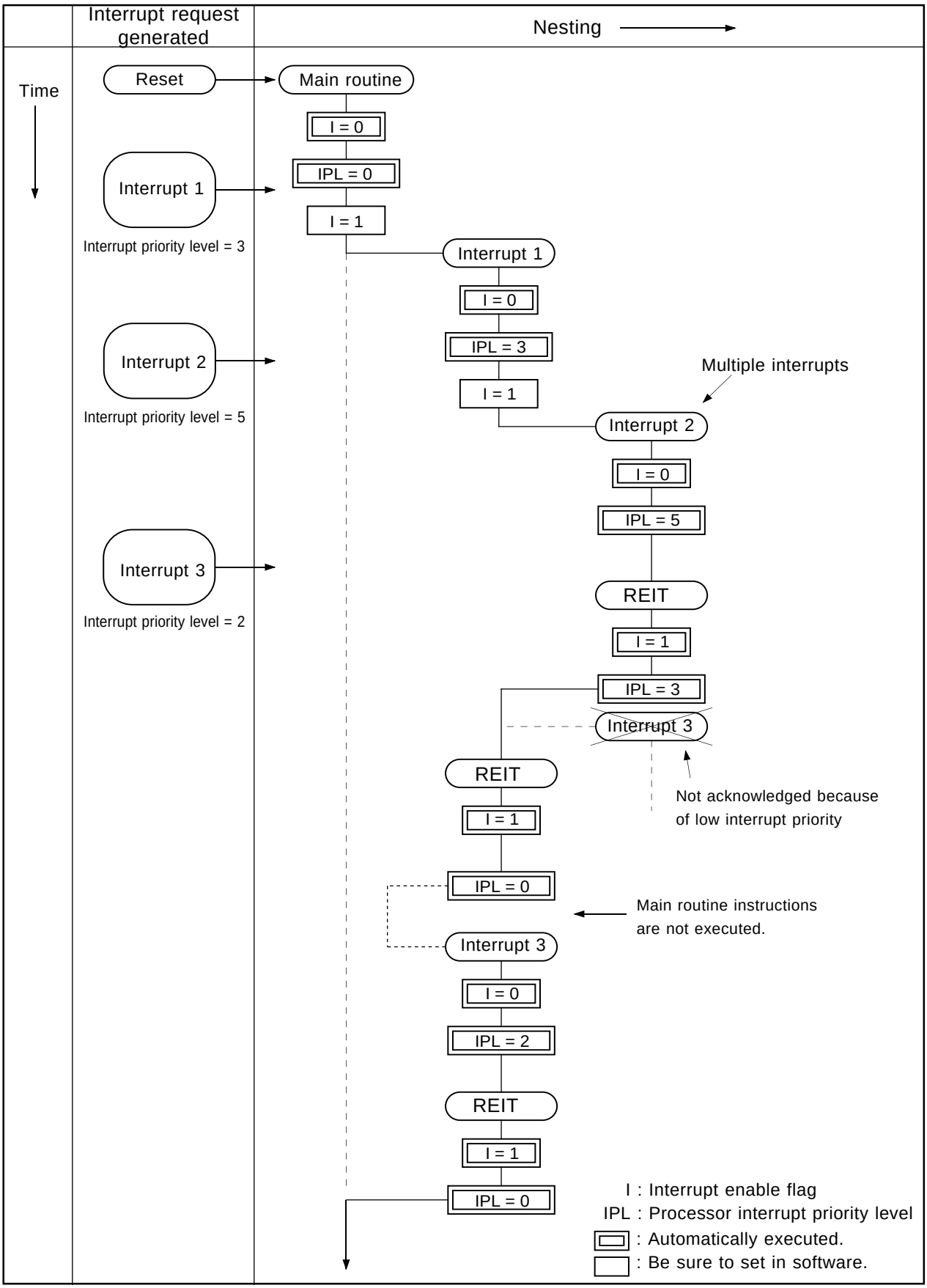


Figure 5.6.1. Multiple interrupts

5.7 Precautions for Interrupts

(1) Reading addresses 000000_{16} and 000002_{16}

- When maskable interrupt is occurred, CPU read the interrupt information (the interrupt number and interrupt request level) in the interrupt sequence from address 000000_{16}. When high-speed interrupt is occurred, CPU read from address 000002_{16}.
  The interrupt request bit of the certain interrupt will then be set to “0”.
  However, reading addresses 000000_{16} and 000002_{16} by software does not set request bit to “0”.

(2) Setting the stack pointer

- The value of the stack pointer immediately after reset is initialized to 000000_{16}. Accepting an interrupt before setting a value in the stack pointer may become a factor of runaway. Be sure to set a value in the stack pointer before accepting an interrupt. When using the $\overline{\text{NMI}}$ interrupt, initialize the stack pointer at the beginning of a program. Any interrupt including the $\overline{\text{NMI}}$ interrupt is generated immediately after executing the first instruction after reset. Set an even number to the stack pointer. When an even number is set, execution efficiency is increased.

(3) Rewrite the interrupt control register

- When a instruction to rewrite the interrupt control register is executed but the interrupt is disabled, the interrupt request bit is not set sometimes even if the interrupt request for that register has been generated. This will depend on the instruction. If this creates problems, use the below instructions to change the register.
  Instructions : AND, OR, BCLR, BSET

5.8 Exit from Stop Mode and Wait Mode

When using an peripheral I/O interrupt to exit stop mode or wait mode, the relevant interrupt must have been enabled and set to a priority level above the level set by the interrupt priority set bits for exiting a stop/wait state. Set the interrupt priority set bits for exiting a stop/wait state to the same level as the processor interrupt level (IPL) of flag register (FLG).
RESET and $\overline{\text{NMI}}$ interrupt are independent of the interrupt priority set bits for exiting a stop/wait state, and stop/wait state is exited.

Chapter 6

Calculation Number of Cycles

6.1 Instruction queue buffer

6.1 Instruction queue buffer

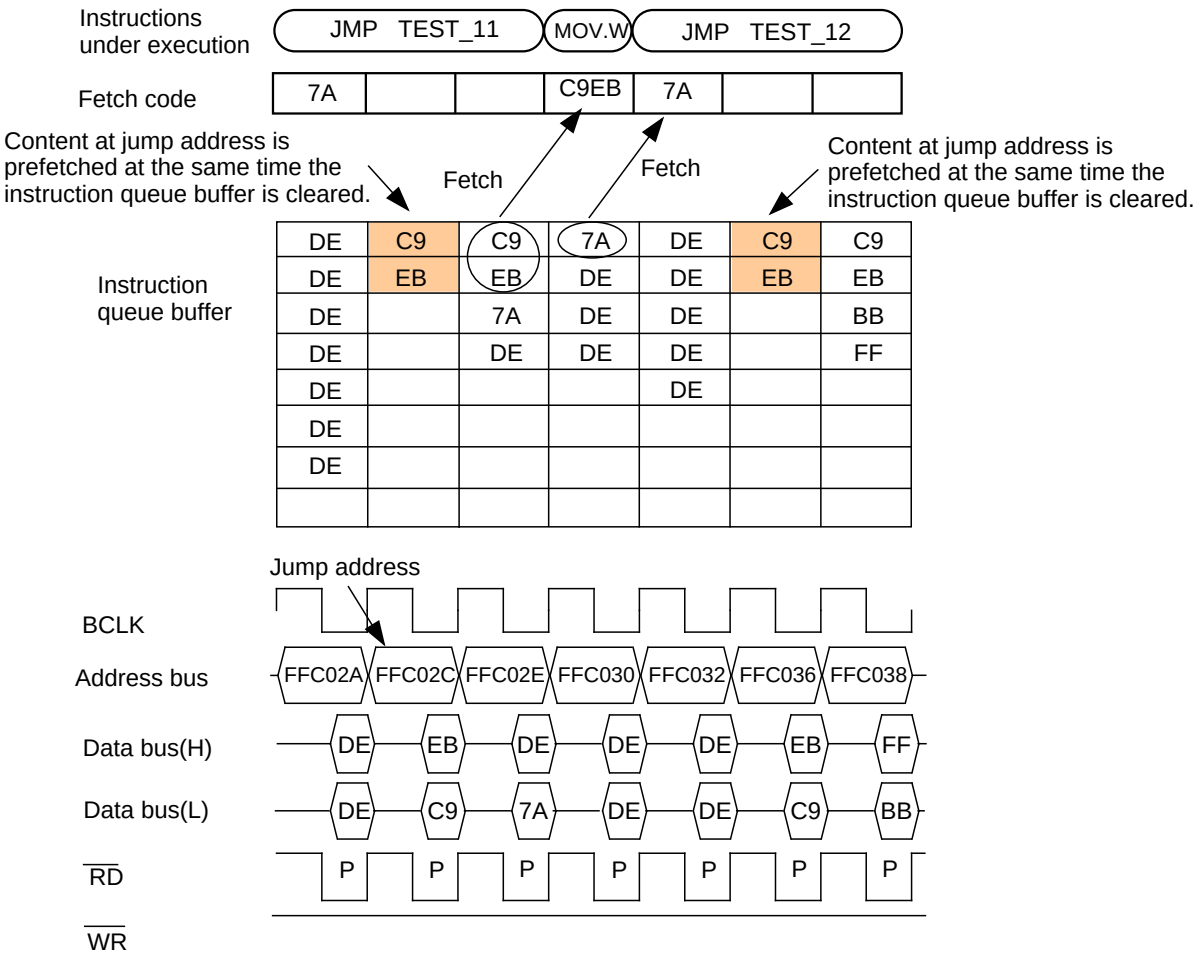
The M32C/80 Series have 8-stage (8-byte) instruction queue buffers. If the instruction queue buffer has a free space when the CPU can use the bus, instruction codes are taken into the instruction queue buffer. This is referred to as “prefetch”. The CPU reads (fetches) these instruction codes from the 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 seven 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.

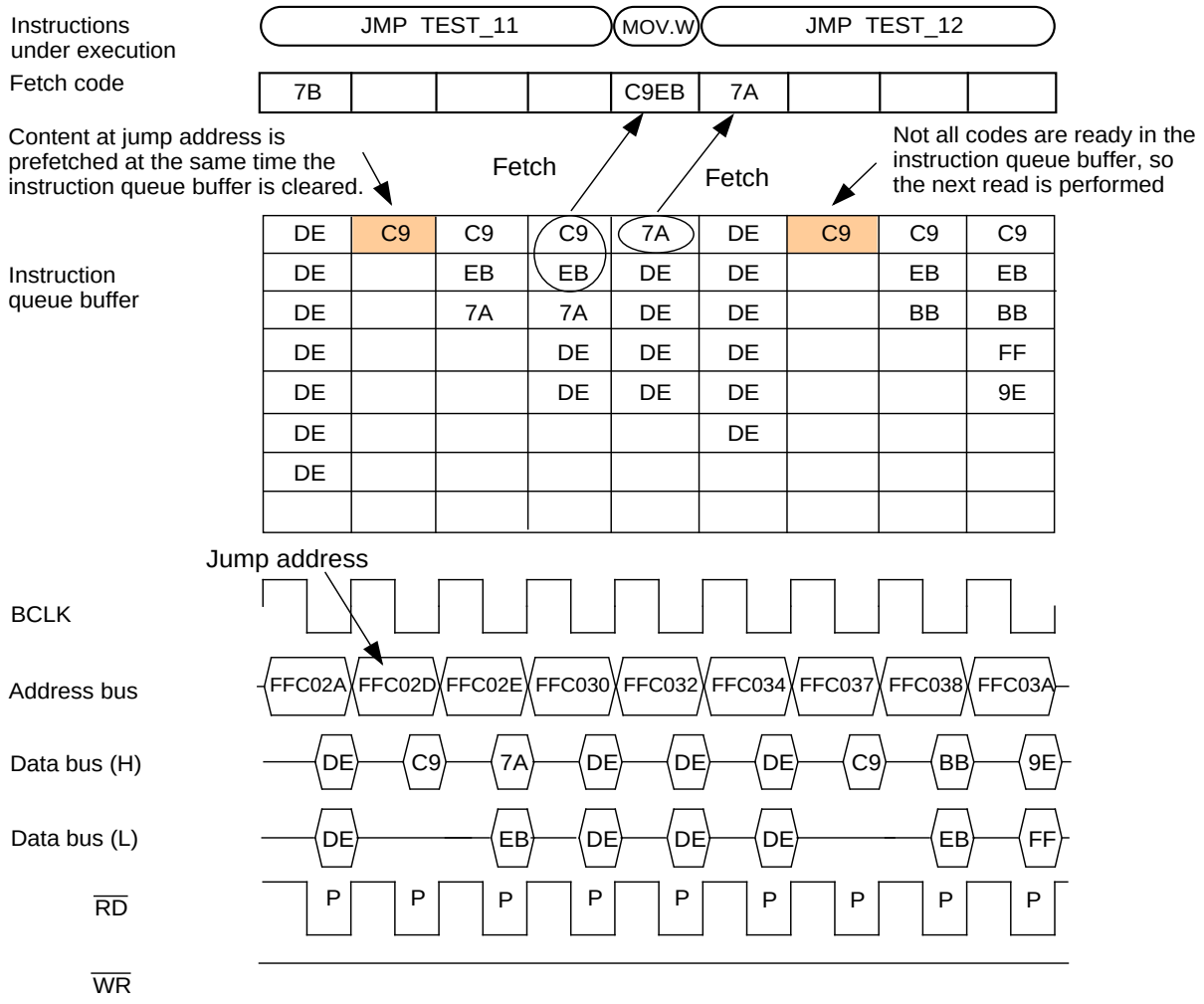
Figures 6.1.1 to 6.1.8 show examples of instruction queue buffer operation and CPU execution cycles.



Sample program

```
Address   Code     Instruction
FFC024      TEST_10:
FFC024    7A       JMP    TEST_11
FFC025    DE       NOP
FFC026    DE       NOP
FFC027    DE       NOP
FFC028    DE       NOP
FFC029    DE       NOP
FFC02A    DE       NOP
FFC02B    DE       NOP
FFC02C      TEST_11:
FFC02C    C9EB     MOV.W:G   R0,R1
FFC02E    7A       JMP    TEST_12
FFC02F    DE       NOP
FFC030    DE       NOP
FFC031    DE       NOP
FFC032    DE       NOP
FFC033    DE       NOP
FFC034    DE       NOP
FFC035    DE       NOP
FFC036      TEST_12:
```

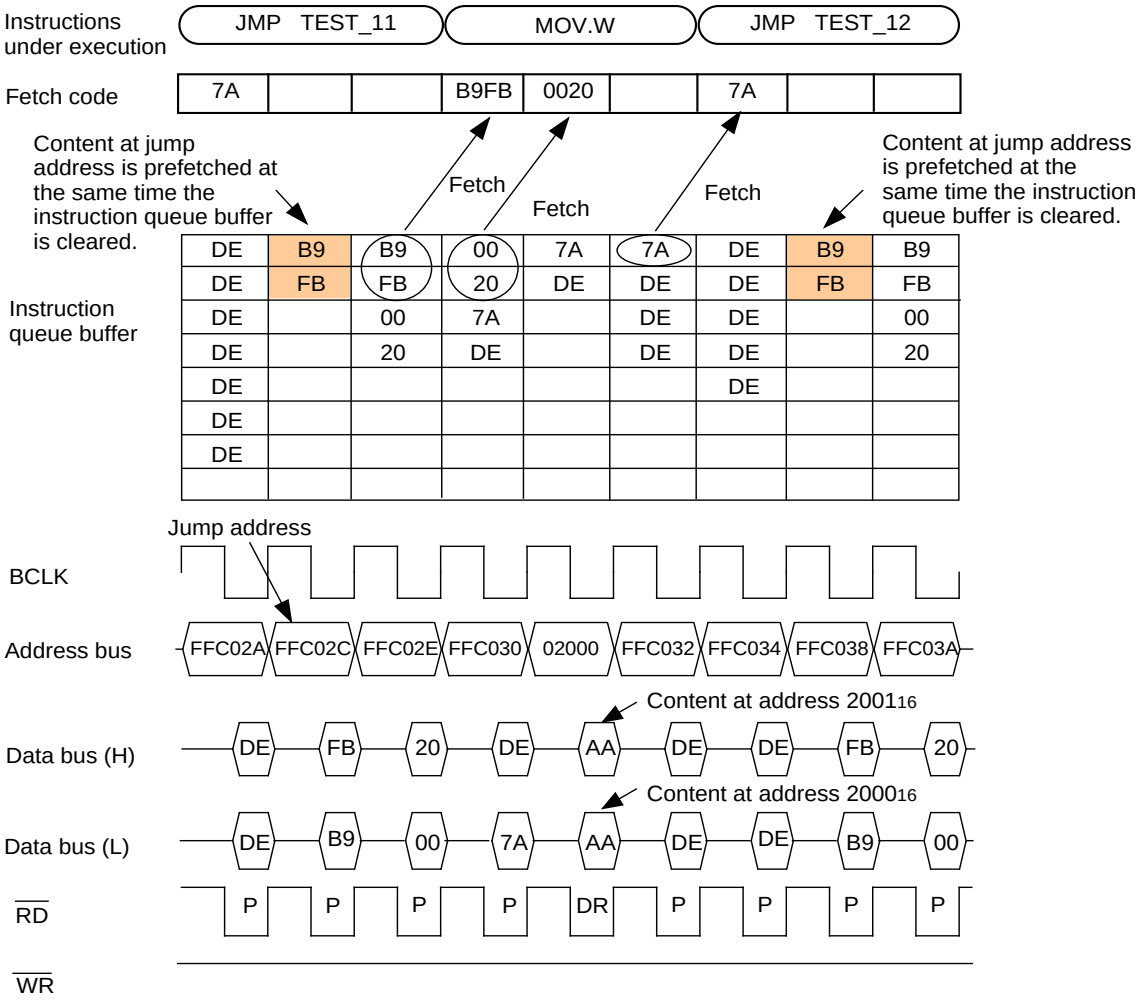
Figure 6.1.1. When executing a register transfer instruction starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)



Sample program

```
Address   Code      Instruction
FFC024      TEST_10:
FFC024    7B        JMP         TEST_11
FFC025    DE        NOP
FFC026    DE        NOP
FFC027    DE        NOP
FFC028    DE        NOP
FFC029    DE        NOP
FFC02A    DE        NOP
FFC02B    DE        NOP
FFC02C    DE        NOP
FFC02D      TEST_11:
FFC02D    C9EB      MOV.W:G     R0,R1
FFC02F    7A        JMP         TEST_12
FFC030    DE        NOP
FFC031    DE        NOP
FFC032    DE        NOP
FFC033    DE        NOP
FFC034    DE        NOP
FFC035    DE        NOP
FFC036    DE        NOP
FFC037      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)

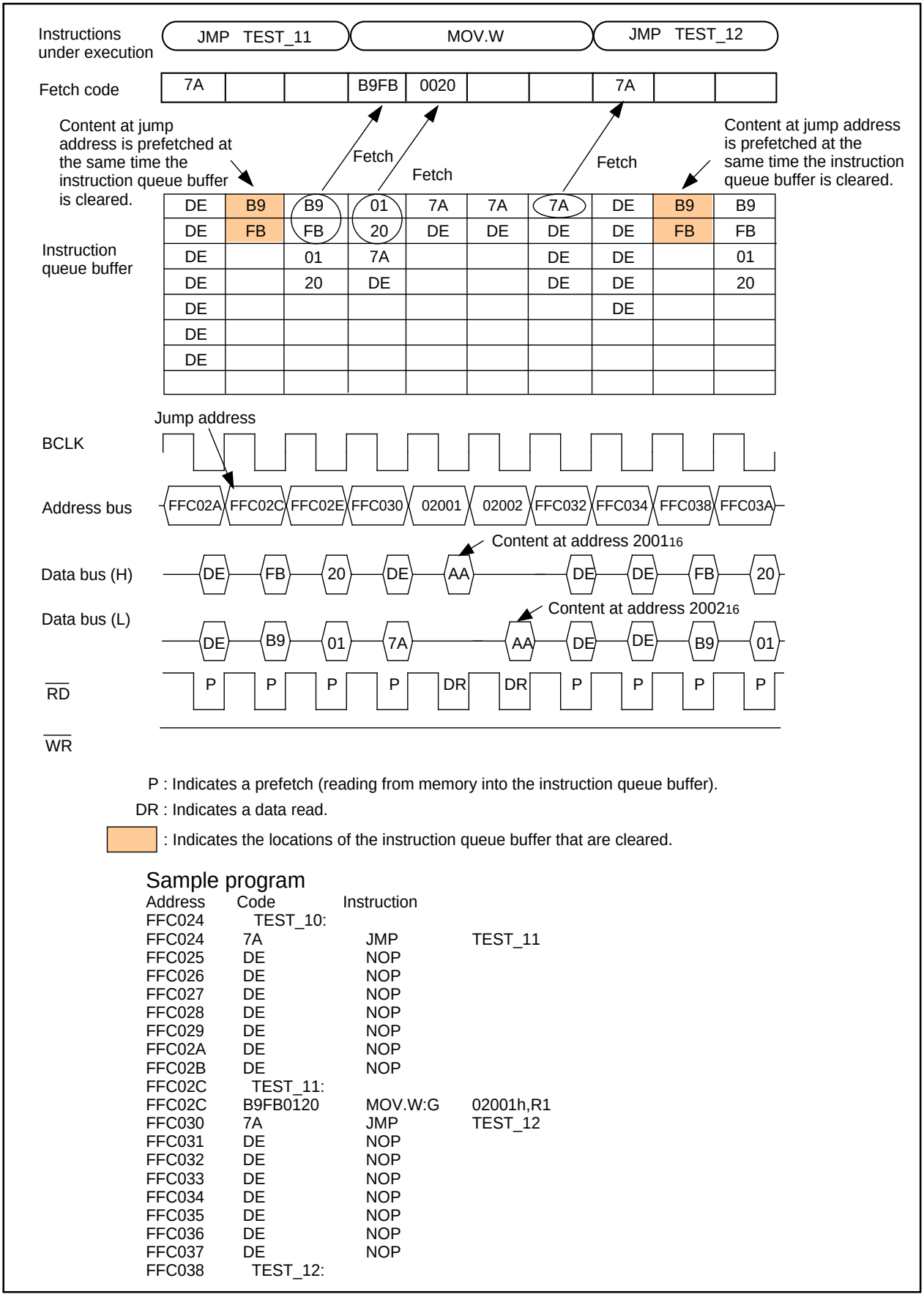


P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
: Indicates the locations of the instruction queue buffer that are cleared.

Sample program

```
Address     Code          Instruction
FFC024        TEST_10:
FFC024      7A            JMP          TEST_11
FFC025      DE            NOP
FFC026      DE            NOP
FFC027      DE            NOP
FFC028      DE            NOP
FFC029      DE            NOP
FFC02A      DE            NOP
FFC02B      DE            NOP
FFC02C        TEST_11:
FFC02C      B9FB0020      MOV.W:G      02000h,R1
FFC030      7A            JMP          TEST_12
FFC031      DE            NOP
FFC032      DE            NOP
FFC033      DE            NOP
FFC034      DE            NOP
FFC035      DE            NOP
FFC036      DE            NOP
FFC037      DE            NOP
FFC038        TEST_12:
```

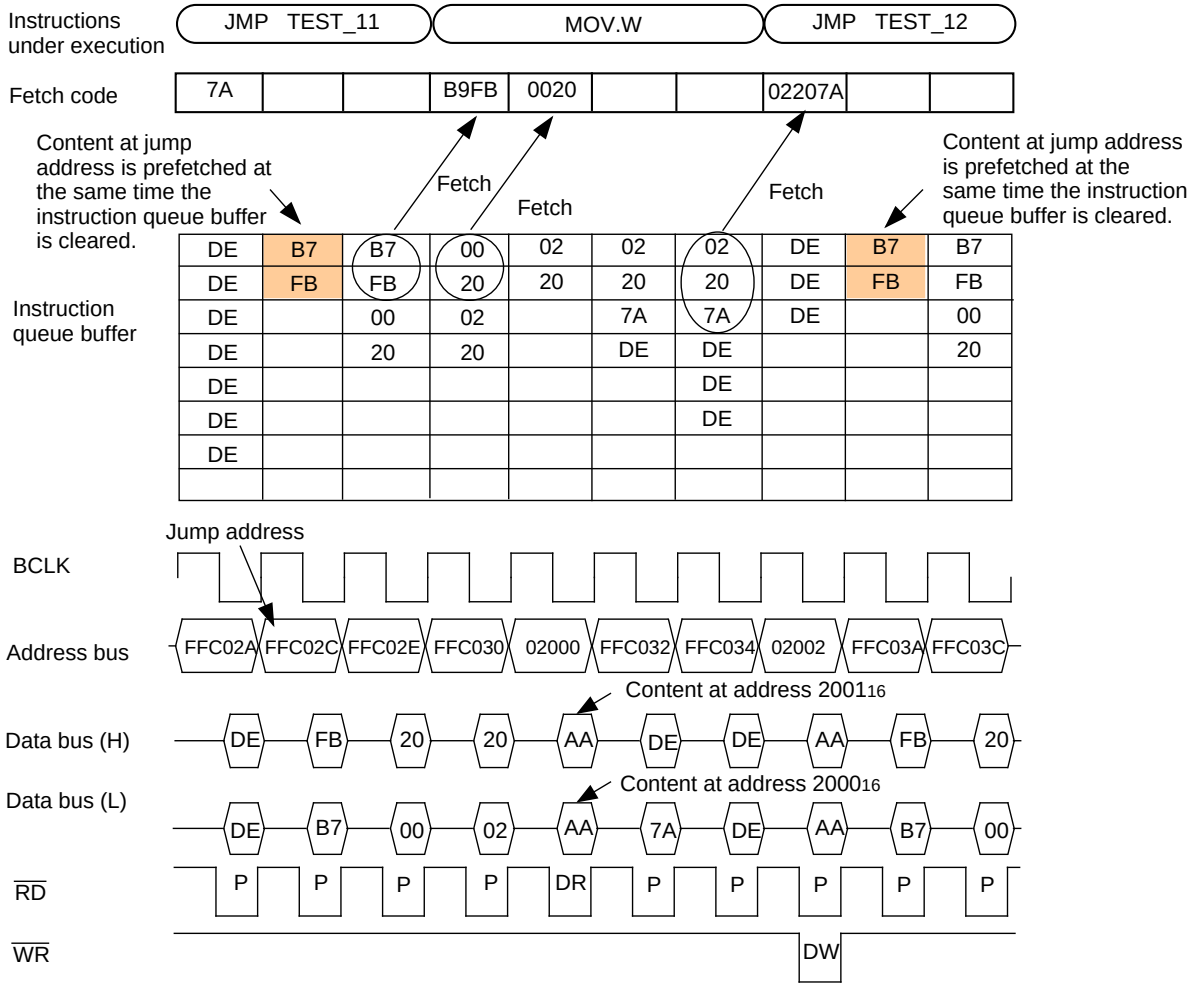
Figure 6.1.3. When executing an instruction to read from even addresses starting from an even address (Program area: 16-bit bus without wait state; Data area: 16-bit bus without wait state)



Sample program

```
Address   Code          Instruction
FFC024      TEST_10:
FFC024    7A            JMP          TEST_11
FFC025    DE            NOP
FFC026    DE            NOP
FFC027    DE            NOP
FFC028    DE            NOP
FFC029    DE            NOP
FFC02A    DE            NOP
FFC02B    DE            NOP
FFC02C      TEST_11:
FFC02C    B9FB0120      MOV.W:G      02001h,R1
FFC030    7A            JMP          TEST_12
FFC031    DE            NOP
FFC032    DE            NOP
FFC033    DE            NOP
FFC034    DE            NOP
FFC035    DE            NOP
FFC036    DE            NOP
FFC037    DE            NOP
FFC038      TEST_12:
```

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)

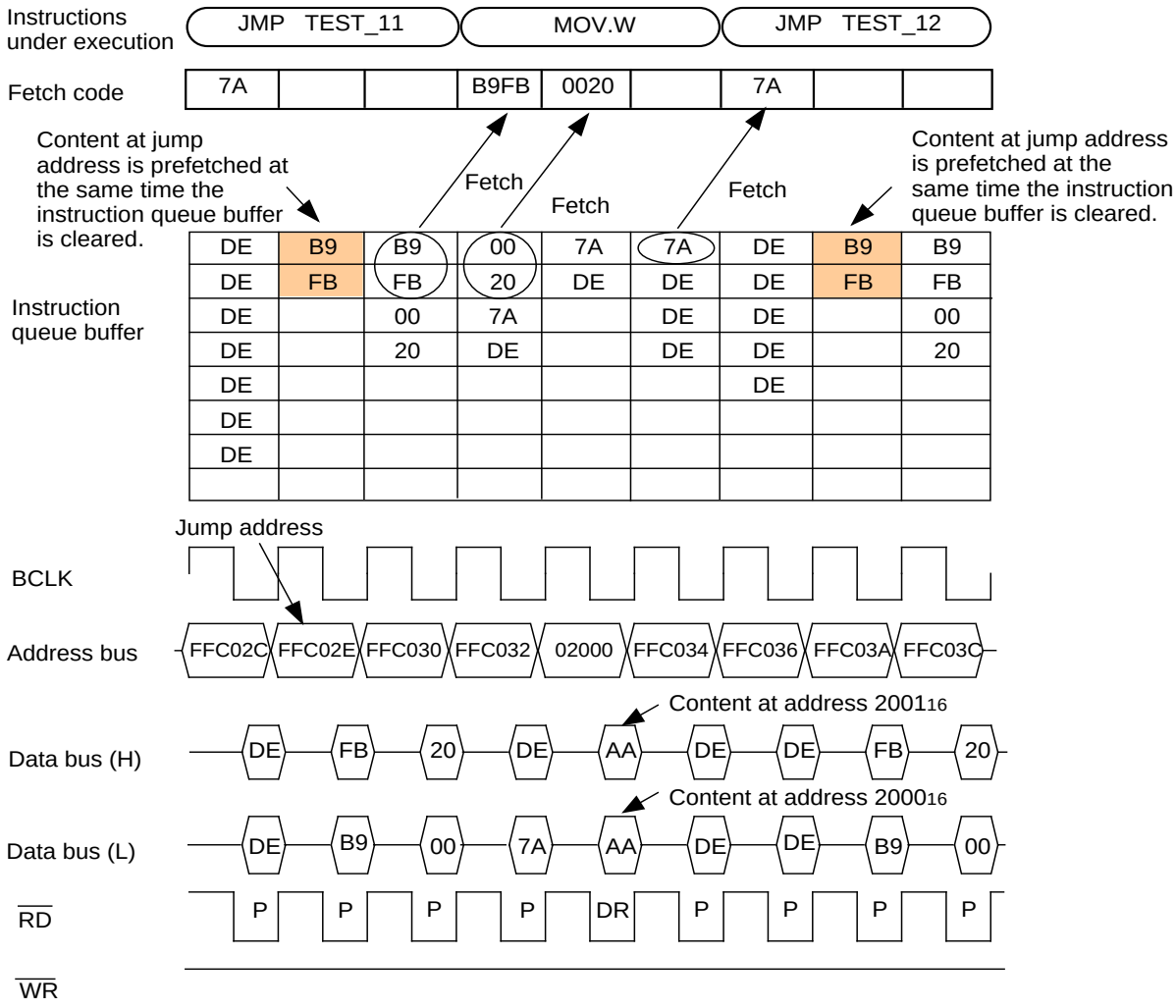


P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
DW : Indicates a data write.
: Indicates the locations of the instruction queue buffer that are cleared.

Sample program

```
Address   Code            Instruction
FFC024      TEST_10:
FFC024    7A              JMP     TEST_11
FFC025    DE              NOP
FFC026    DE              NOP
FFC027    DE              NOP
FFC028    DE              NOP
FFC029    DE              NOP
FFC02A    DE              NOP
FFC02B    DE              NOP
FFC02C      TEST_11:
FFC02C    B7FB00200220    MOV.W   02000h,02002h
FFC032    7A              JMP     TEST_12
FFC033    DE              NOP
FFC034    DE              NOP
FFC035    DE              NOP
FFC036    DE              NOP
FFC037    DE              NOP
FFC038    DE              NOP
FFC039    DE              NOP
FFC03A      TEST_12:
```

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)

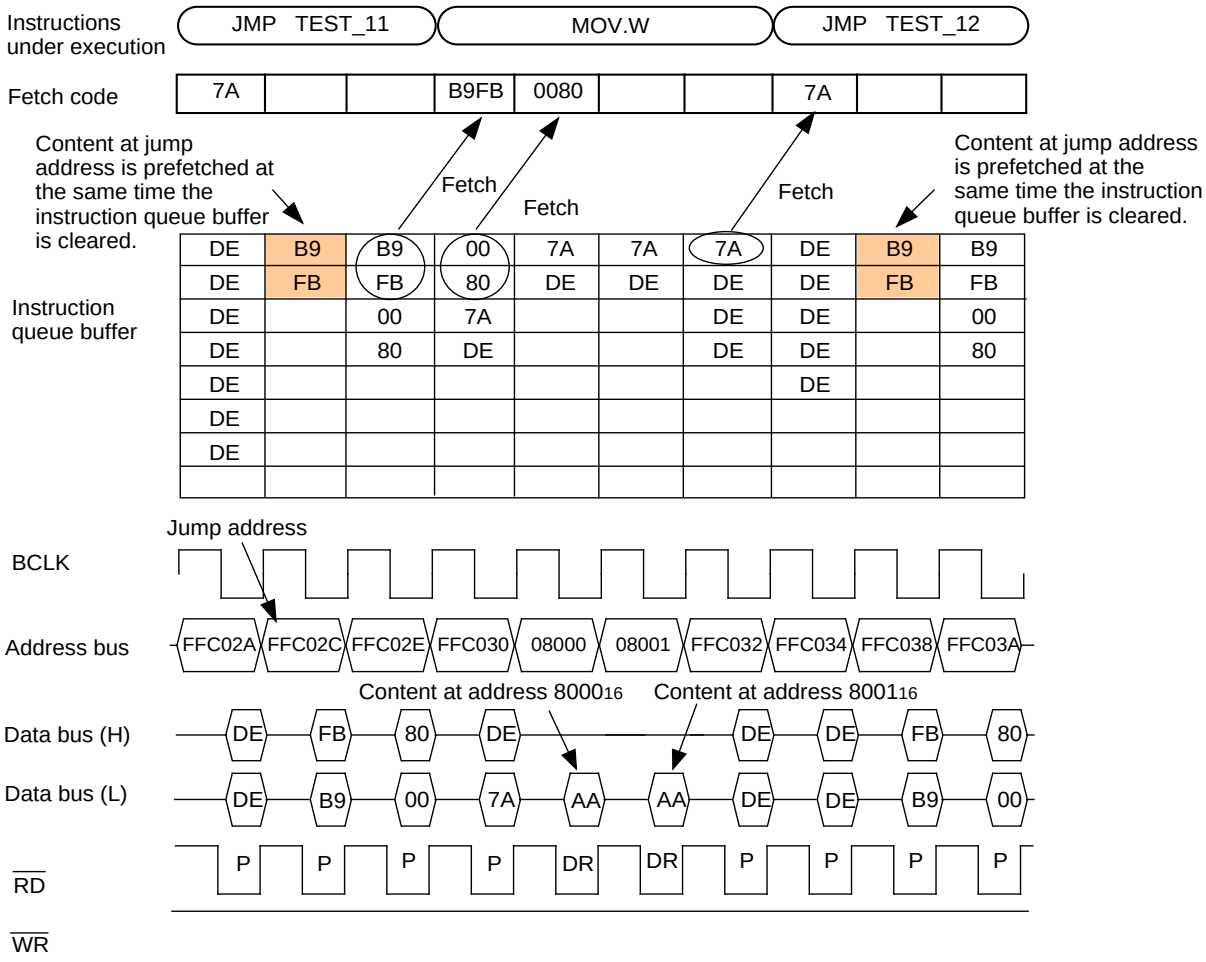


P : Indicates a prefetch (reading from memory into the instruction queue buffer).
DR : Indicates a data read.
: Indicates the locations of the instruction queue buffer that are cleared.

Sample program

```
Address  Code        Instruction
FFC026       TEST_10:
FFC026   7A          JMP        TEST_11
FFC027   DE          NOP
FFC028   DE          NOP
FFC029   DE          NOP
FFC02A   DE          NOP
FFC02B   DE          NOP
FFC02C   DE          NOP
FFC02D   DE          NOP
FFC02E       TEST_11:
FFC02E   B9FB0020    MOV.W:G    02000h,R1
FFC032   7A          JMP        TEST_12
FFC033   DE          NOP
FFC034   DE          NOP
FFC035   DE          NOP
FFC036   DE          NOP
FFC037   DE          NOP
FFC038   DE          NOP
FFC039   DE          NOP
FFC03A       TEST_12:
```

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)



P : Indicates a prefetch (reading from memory into the instruction queue buffer).

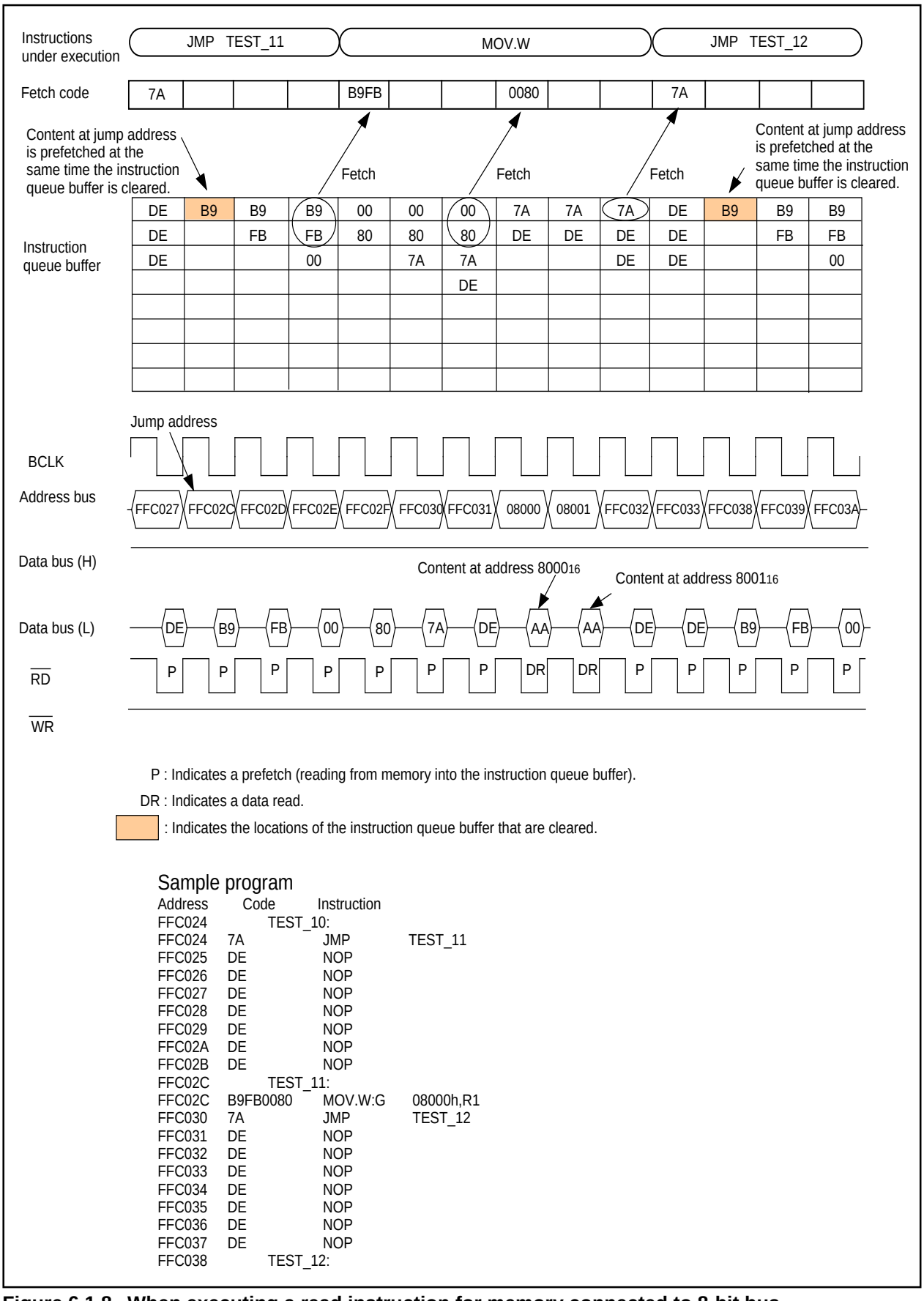
DR : Indicates a data read.

: Indicates the locations of the instruction queue buffer that are cleared.

Sample program

```
Address     Code        Instruction
FFC024          TEST_10:
FFC024  7A              JMP         TEST_11
FFC025  DE              NOP
FFC026  DE              NOP
FFC027  DE              NOP
FFC028  DE              NOP
FFC029  DE              NOP
FFC02A  DE              NOP
FFC02B  DE              NOP
FFC02C          TEST_11:
FFC02C  B9FB0080        MOV.W:G     08000h,R1
FFC030  7A              JMP         TEST_12
FFC031  DE              NOP
FFC032  DE              NOP
FFC033  DE              NOP
FFC034  DE              NOP
FFC035  DE              NOP
FFC036  DE              NOP
FFC037  DE              NOP
FFC038          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)



Sample program

```
Address     Code        Instruction
FFC024              TEST_10:
FFC024      7A          JMP         TEST_11
FFC025      DE          NOP
FFC026      DE          NOP
FFC027      DE          NOP
FFC028      DE          NOP
FFC029      DE          NOP
FFC02A      DE          NOP
FFC02B      DE          NOP
FFC02C              TEST_11:
FFC02C      B9FB0080    MOV.W:G     08000h,R1
FFC030      7A          JMP         TEST_12
FFC031      DE          NOP
FFC032      DE          NOP
FFC033      DE          NOP
FFC034      DE          NOP
FFC035      DE          NOP
FFC036      DE          NOP
FFC037      DE          NOP
FFC038              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)

Chapter 7

Precautions

7.1 String/Product-Sum Operation Instruction

7.1 String Instruction, Product-Sum Operation Instruction

The string instructions and the product sum operation instruction listed in 7.1.1. Subject Instructions will be aborted under the conditions listed in 7.1.2 Problem Conditions.

7.1.1 Subject Instructions

String instructions: SCMPU, SIN, SMOVB, SMOVF, SMOVU, SOUT, SSTR
Product sum operation instruction: RMPA

7.1.2 Problem Conditions

When DMAC is not used:

1) (a) The interrupt A is requested when bits ILVL2 to ILVL0 in the interrupt control registers are set to other than 000b (level 0, interrupt disabled). However, the interrupt request is not acknowledged because the I flag is set to 0 (interrupt disabled) or the requested interrupt has smaller priority level than IPL (IPL interrupt priority level < 001b) while the interrupt is requested.

(b) After (a), set the IR bit in the interrupt control register for the interrupt A to 0 (no interrupt request) by program or set the interrupt priority level smaller than the last set level.

(c) After (b), execute the subject instruction, SCMPU, SIN, SMOVB, SMOVF, SMOVU, SOUT, SSTR, or RMPA, immediately after setting the I flag to 1 or IPL to smaller priority than the interrupt priority level, which is set when an interrupt request is generated, to enable the requested interrupt.

When DMAC is used,

2) (a) The interrupt A is requested when bits ILVL2 to ILVL0 are set to other than 000b. However, the interrupt request is not acknowledged because the I flag is set to 0 or the requested interrupt has smaller priority level than IPL (IPL interrupt priority level < 001b) while the interrupt is requested.

(b) After (a), set the IR bit for the interrupt A to 0 by program or set the interrupt priority level smaller than the last set level.

(c) After (b), execute the subject instruction within next three instructions after setting the I flag to 1 or IPL to smaller priority than the interrupt priority level, which is set when an interrupt request is generated, to enable the requested interrupt.

3) (a) Interrupts are generated immediately before or in the middle of executing the subject instruction. The interrupt A request is generated when bits ILVL2 to ILVL0 are set to other than 000b (level 0, interrupt disabled) in the interrupt routine. However, the interrupt request is not acknowledged because I flag is set to 0 or because IPL is equal to or greater than the interrupt priority level even if I flag is set to 1 (multiple interrupts enabled).

(b) After (a), set the IR bit for the interrupt A to 0 by program or set the interrupt priority level smaller than the last set level.

(c) After (b), execute the subject instructions after the interrupts are completed with the REIT instruction or FREIT instruction.

If DMA transfer occurs in conditions 2)-(c) or 3)-(c), the subject instruction is aborted.
The patterns for the above conditions 1) through 3) are illustrated in Figure 7.1.

7.1.3 Operation Check

Use the flow chart in Figure 7.2 to determine whether the countermeasure programs are needed. When the countermeasure is needed, refer to 7.1.4.

7.1.4 Countermeasure Program

To execute the subject instruction, interrupts need to be disabled. If interrupts cannot be disabled, use the countermeasure program in Figure 7.3.

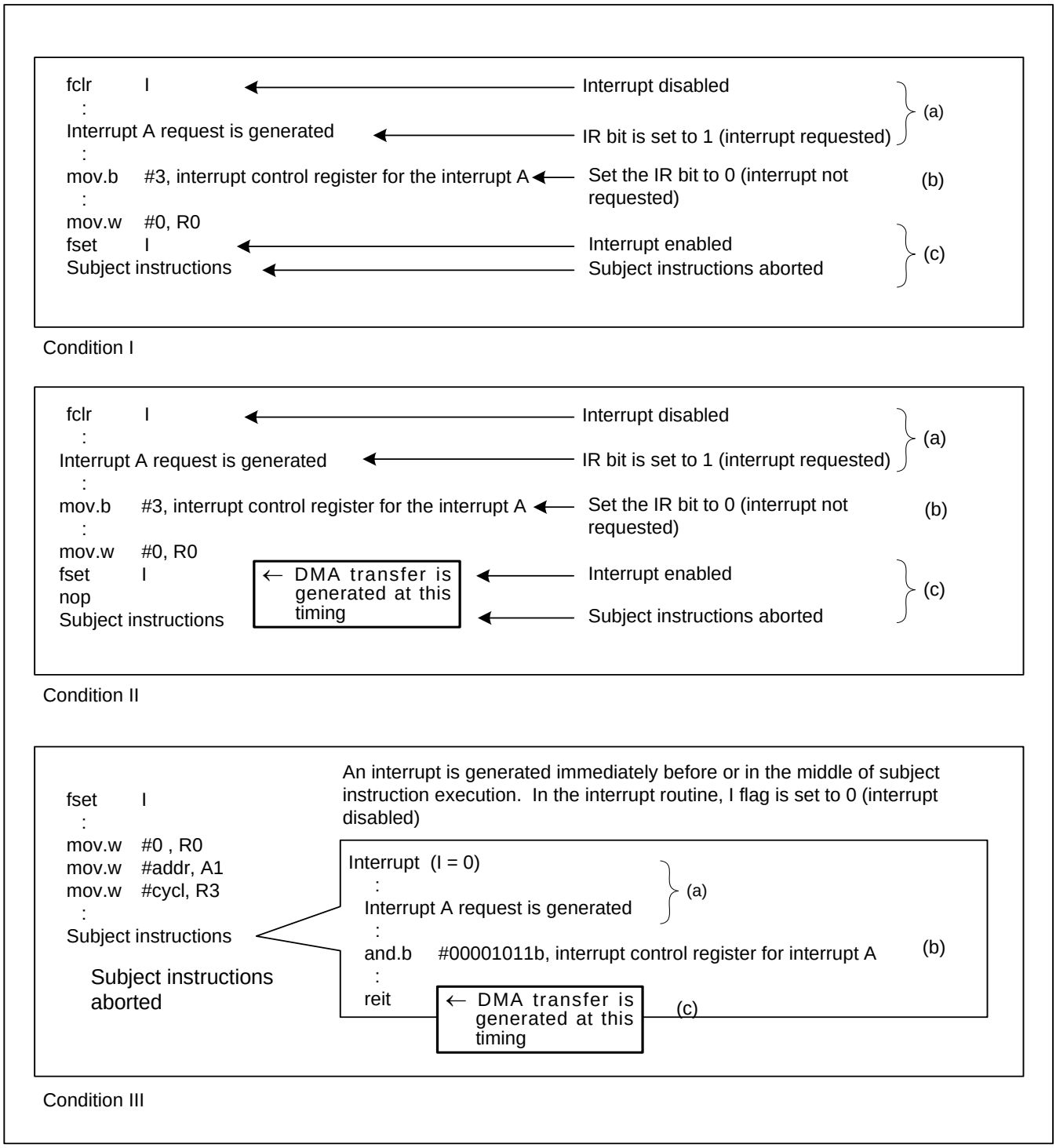


Figure 7.1 Problem Conditions

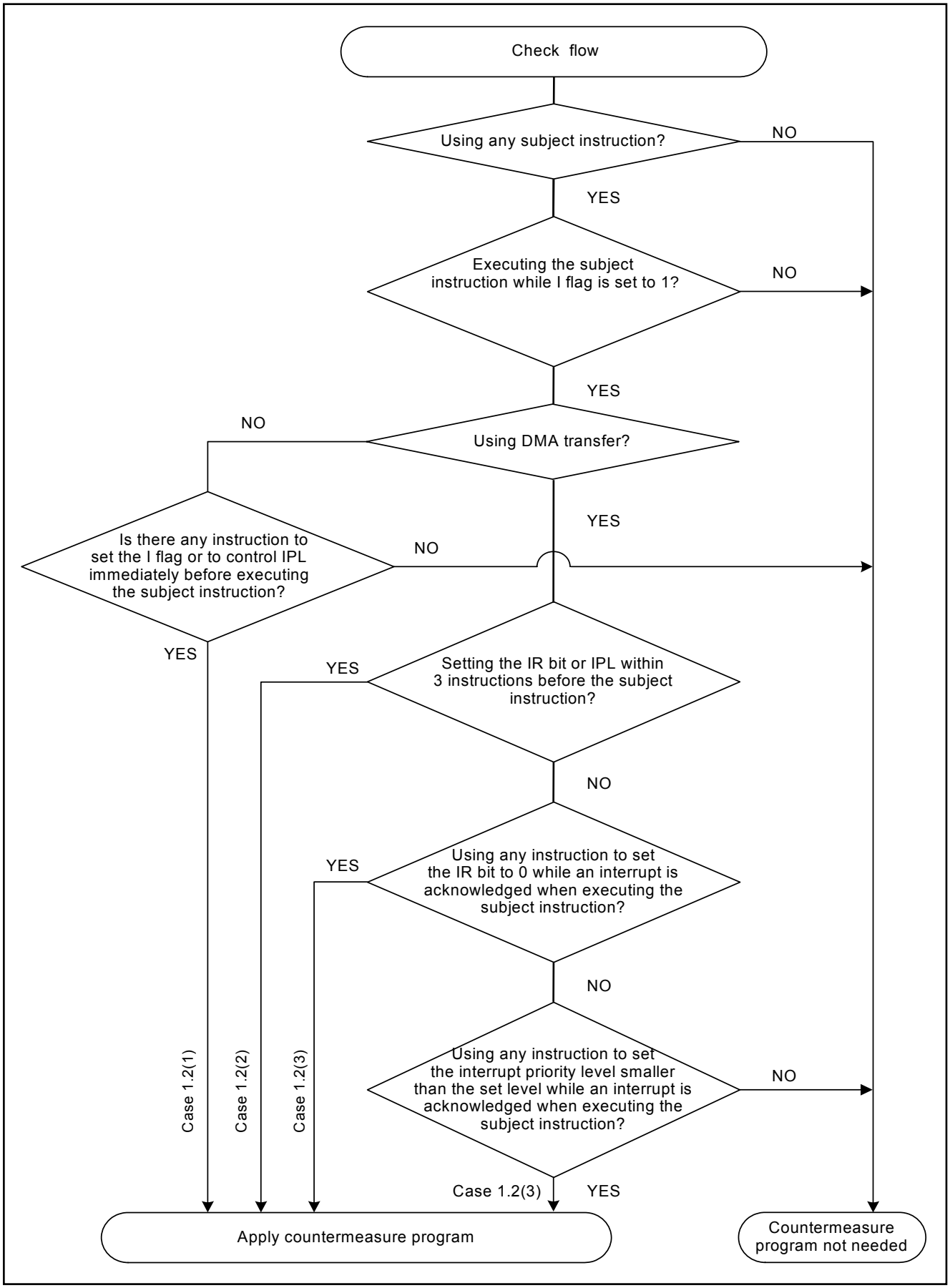


Figure 7.2 Operation Check Flow

•Instructions SIN, SMOVB, SMOVF, SOUT, and SSTR

```
Loop:
        String instruction
        cmp.w       #0, R3
        jnz         Loop
```

•SMOVU Instruction

When the size specifier is ".B,"

```
Loop:
        smovu.b
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jnz         Loop
```

When the size specifier is ".W,"

```
Loop:
        smovu.w
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jz          Next
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jnz         Loop
Next:
```

•RMPA Instruction

```
Loop:
        rmpa.b (rmpa.w)
        pushc       FLG
        jz          Next
        popc        FLG
        jmp         Loop
Next:
        popc        FLG
```

•SCMPU Instruction

When the size specifier is ".B,"

```
Loop:
        scmpu.b
        pushc       FLG
        jnz         Next
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jz          Next
        popc        FLG
        jmp         Loop
Next:
        popc        FLG
```

When the size specifier is ".W,"

```
Loop:
        scmpu.w
        pushc       FLG
        jnz         Next
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jz          Next
        sub.l       #1, A0
        sub.l       #1, A1
        cmp.b       #0, [A0]
        jz          Next
        popc        FLG
        jmp         Loop
Next:
        popc        FLG
```

Figure 7.3 Countermeasure Programs

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 the same 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 page, 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.

CPU

Q

How do I distinguish between the static base register (SB) and the frame base register (FB)?

A

Only positive displacement is allowed in SB Relative Addressing, while FB Relative Addressing can be with positive or negative displacement.
If you write a program in C, use FB as a stack frame base register.
You can use SB and FB as intended in programming in the assembly language.

CPU

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

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 divider.

In general, the following relation among quotient, divider, dividend, and remainder holds.

dividend = divider × 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.

Q

It is 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 MCU 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.

Glossary

Technical terms used in this software manual are explained below. They are good in this manual only.

Term	Meaning	Related word
borrow	To move a digit to the next lower position.	carry
carry	To move 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.	
LSB	Abbreviation for Least Significant Bit The bit occupying the lowest-order position of a data item.	MSB

Term	Meaning	Related word
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 MCU 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
dsp24	24-bit displacement
dsp16	16-bit displacement
dsp8	8-bit displacement
EVA( )	An effective address indicated by what is enclosed in ( )
EXTS( )	Sign extension indicated by what is enclosed in ( )
EXTZ( )	Zero extension indicated by what is enclosed in ( )
(HH)	Higher-order byte of higher-order word of a register or memory (highest byte)
H4:	Four higher-order bits of an 8-bit register or 8-bit memory
(HL)	Lower-order byte of higher-order word of a register or memory
| |	Absolute value
(LH)	Higher-order byte of lower-order word of a register or memory
(LL)	Lower-order byte of lower-order word of a register or memory (lowest byte)
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 ( )
MSB	Most Significant Bit
PC_H	Higher-order byte of the program counter
PC_{ML}	Middle-order byte and lower-order byte of the program counter
FLG_H	Four higher-order bits of the flag register
FLG_L	Eight lower-order bits of the flag register
[ ]	Indirect addressing

Index

Rev.	Date	Description	
		Page	Summary
B	Sep 01, 2001	—	Preliminary edition issued
1.00	May 31, 2006		Document number is changed from "MEJ19B0002-0200Z" to "REJ09B0319-0100"
		2	"Speed performance" is modified.
		109, 111	MUL and MULU instruction -Add "R0, R1, A0, A1 can be selected for *dest*." in [Function] line 7. -Delete "the result is stored ~~in R2R0~~ in 32 bits." in [Function] line 12. -Add "R2R0 is selected for *dest*." in [Function] line 12. -"*1 When you specify (.B) and (.W) for the indirect addressing [src], you can use in all addressing except R0L/R0/R2R0, R0H/R2/-, R1L/R1/R3R1, R1H/R3/-, and #IMM; when (.L), you cannot use [src]." ---> "*1 When the size specifier (.size) is (.B), indirect instruction addressing [src] and [dest] can be used in all addressing except R0L, R0H, R1L, R1H, and #IMM. When the size specifier (.size) is (.W), indirect instruction addressing [src] can be used in all addressing except R0, R1, and #IMM. Indirect instruction addressing [dest] cannot be used in any addressing. When the size specifier (.size) is (.L), no indirect instruction addressing can be used." -Delete "*3 When you specify (.L) for the size specifier (.size), you can choose only R2R0 for *dest*."
		Chap 3	Errors in [Function] are fixed.
		Chap 4	"Number of cycles" is modified.
		257, 258 260, 261	"When (.W) is specified for the size specifier (.size), only Rn and An can be selected for *dest*." is added.
		315	"Note" is modified.
		331	"Chapter 7 Precautions" is added.

M32C/80 Series Software Manual

Publication Date: Rev.1.00 May 31, 2006

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.

M32C/80 Series
Software Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ09B0319-0100