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

32

R32C/100 SERIES

SOFTWARE MANUAL

RENESAS MCU
M16C FAMILY / R32C/100 SERIES

Renesas Electronics
www.renesas.com

Rev.1.00 2009.04

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

The Notation in This Manual

For detailed explications of the R32C/100 Series's operation, the notation based on specific rules is used throughout this manual as shown below.

Category	Notation	Description/Meaning
Symbols	IMM	Immediate value
	IMMEX	Immediate value to be sign-extended
	abs	Absolute value of absolute addressing mode
	dsp	Displacement value of relative addressing mode
	[ ]	Indirect addressing mode
	label	Specified address as Label used in program counter relative addressing mode. Write a label when programming. An address can be written directly, as well.
	bit	Specified bit number. Write a numeric value in the actual program.
	R0LB, R0B, R2R0B etc.	Bank1 register (with "B" appended)
Numbers	00b	Binary number (with "b" appended)
	0000h	Hexadecimal number (with "h" appended). For a numeric number beginning with an alphabetic letter, add 0 before the letter. For example, FFh should be written as 0FFh.
	100	Decimal number (BCD)
Bit length	#IMM:8 etc.	The underlined part indicates an effective bit length for the operand.
	:3	Effective bit length is 3.
	:4	Effective bit length is 4.
	:8	Effective bit length is 8.
	:16	Effective bit length is 16.
	:24	Effective bit length is 24.
	:32	Effective bit length is 32.
Instruction formats	MOV. B:S etc.	The underlined part specifies instruction format.
	:G	Generic format
	:Q	Quick format
	:S	Short format
	:Z	Zero format
Data size for operation	MOV.W etc.	The underlined part specifies data size for operation.
	.B	Byte (8 bits)
	.W	Word (16 bits)
	.L	Long word (32 bits)
	.BW	Byte to Word
	.BL	Byte to Long word
	.WL	Word to Long word

Category	Notation	Description/Meaning
Jump distance	JMP.A etc.	Underlined part specifies effective bit length of jump relative distance.
	.S	3-bit PC forward relative. The effective value is 1 to 8.
	.B	8-bit PC relative. The effective value is -128 to 127.
	.W	16-bit PC relative. The effective value is -32768 to 32767.
	.A	24-bit PC forward relative. The effective value is -8388608 to 8388607.
Operators		C language syntax is applied in principle. The notations used in this manual are shown as below.
	=	Assignment operator. Assign the value on the right side to the left.
	-	Unary minus or subtraction
	+	Addition
	*	Dereferencing operator or multiplication
	/	Division
	%	Modulus
	~	Bitwise NOT
	&	Bitwise AND
	|	Bitwise OR
	^	Bitwise XOR (exclusive or)
	(float)	Cast operator
	;	End of a statement
	{ }	Start and end of compound statements. More than two statements can be described in {}.
	if (expression) statement 1 else statement 2	If the conditional expression evaluates to true, statement 1 is executed, otherwise statement 2 is executed.
	for (statement 1;expression;statement 2) {loop body}	After statement 1 is executed, the loop body is executed followed by statement 2 as long as the conditional expression evaluates to true.
	do statement while (expression);	The statement is executed until the expression evaluates to false. The loop body is executed at least once whether true or not.
	while (expression) statement	The statement is executed until the expression evaluates to false.
	==, !=	Equal, not equal comparisons
	>, <	Greater-than, less-than comparisons
	>=, <=	Greater-than or equal to, less-than or equal to comparisons
	&&, ||	Logical AND, logical OR The operators guarantee left-to-right evaluation.
Keyword	true	Non-C language notation. If the condition specified by *Cnd* is true, 1 is obtained.
Available src/ dest	#IMMEX	#IMMEX:8 or #IMMEX:16 can be specified.
	#IMM	Either #IMM:8, #IMM:16 or #IMM:32 can be specified depending on data size for operation. Either #IMM:3 or #IMM:4 is not included.

TABLE OF CONTENTS

R32C/100 SERIES INSTRUCTION SET

Table C.1 Instruction Set - Alphabetical (1 / 5)

Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
ABS	Absolute	42	182
ADC	Add with Carry	43	182
ADCF	Add Carry Flag	44	183
ADD	Add	45	184
ADDF	Add Floating-point	47	187
ADSF	Add Sign Flag	48	188
AND	And Logical	49	189
BCLR	Clear a Bit	51	191
BITINDEX	Index next Bit Instruction	52	192
BMCnd	Move a Bit Conditionally (conditions listed below)	53	192
BMC	Move a bit if the C flag is 1	53	192
BMEQ	Move a bit if equal	53	192
BMGE	Move a bit if greater as signed integers or equal	53	192
BMGEU	Move a bit if greater as unsigned integers or equal	53	192
BMGT	Move a bit if greater as signed integers	53	192
BMGTU	Move a bit if greater as unsigned integers	53	192
BMLE	Move a bit if less as signed integers or equal	53	192
BMLEU	Move a bit if less as unsigned integers or equal	53	192
BMLT	Move a bit if less as signed integers	53	192
BMLTU	Move a bit if less as unsigned integers	53	192
BMN	Move a bit if value is negative	53	192
BMNC	Move a bit if the C flag is 0	53	192
BMNE	Move a bit if not equal	53	192
BMNO	Move a bit if the O flag is 0	53	192
BMNZ	Move a bit if the Z flag is 0	53	192
BMO	Move a bit if the O flag is 1	53	192
BMPZ	Move a bit if value is positive or 0	53	192
BMZ	Move a bit if the Z flag is 1	53	192
BNOT	Not a Bit	55	193
BRK	Break	56	193
BRK2	Break 2	57	193
BSET	Set a Bit	58	194
BTST	Test a Bit	59	194
BTSTC	Test a Bit and Clear	60	195

Table C.1 Instruction Set - Alphabetical (2 / 5)

Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
BTSTS	Test a Bit and Set	61	196
CLIP	Clip	62	196
CMP	Compare	63	197
CMPF	Compare Floating-point	65	199
CNVIF	Convert Integer to Floating-point	66	201
DADC	Add Decimal with Carry	67	203
DADD	Add Decimal	68	205
DEC	Decrement	69	206
DIV	Signed Divide	70	207
DIVF	Divide Floating-point	71	209
DIVU	Unsigned Divide	72	211
DIVX	Signed Divide extra	73	213
DSBB	Subtract Decimal with Borrow	74	215
DSUB	Subtract Decimal	75	217
EDIV	Extended Signed Divide with Remainder	76	219
EDIVU	Extended Unsigned Divide with Remainder	77	220
EDIVX	Extended Signed Divide extra with Remainder	78	221
EMUL	Extended Signed Multiply	79	222
EMULU	Extended Unsigned Multiply	80	223
ENTER	Enter and Create Stack Frame	81	223
EXITD	Exit and Deallocate Stack Frame	82	224
EXITI	Exit Interrupt and Deallocate Stack Frame	83	224
EXTS	Sign Extend	84	224
EXTZ	Zero Extend	85	226
FCLR	Set a Flag	87	228
FREIT	Return from Fast Interrupt	88	228
FSET	Set a Flag	89	229
INC	Increment	90	229
INDEXType	Index (types listed below)	91	230
INDEXB	Add to the first and second operands	91	231
INDEX1	Add to the first operand	91	230
INDEX2	Add to the second operand	91	230
INT	Interrupt	92	231
INTO	Interrupt on Overflow	93	231
JCnd	Jump Conditionally (conditions listed below)	94	232
JC	Jump if the C flag is 1	94	232
JEQ	Jump if equal	94	232
JGE	Jump if greater as signed integers or equal	94	232
JGEU	Jump if greater as unsigned integers or equal	94	232

Table C.1 Instruction Set - Alphabetical (3 / 5)

Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
JGT	Jump if greater as signed integers	94	232
JGTU	Jump if greater as unsigned integers	94	232
JLE	Jump if less as signed integers or equal	94	232
JLEU	Jump if less as unsigned integers or equal	94	232
JLT	Jump if less as signed integers	94	232
JLTU	Jump if less as unsigned integers	94	232
JN	Jump if a negative value	94	232
JNC	Jump if the C flag is 0	94	232
JNE	Jump if not equal	94	232
JNO	Jump if the O flag is 0	94	232
JNZ	Jump if the Z flag is 0	94	232
JO	Jump if the O flag is 1	94	232
JPZ	Jump if a positive value or 0	94	232
JZ	Jump if the Z flag is 1	94	232
JMP	Jump Always	95	232
JMPI	Jump Indirectly	96	233
JSR	Jump to Subroutine	97	234
JSRI	Jump to Subroutine Indirectly	98	234
LDC	Load into Control Register	99	235
LDCTX	Load Context	100	236
LDIPL	Load Interrupt Priority Level	102	237
MAX	Select Maximum Value	103	237
MIN	Select Minimum Value	104	239
MOV	Move	105	241
MOVA	Move Effective Address	108	246
MOVDir	Move Nibble Data (directions listed below)	109	247
MOVHH	Move upper src to upper dest	109	247
MOVHL	Move upper src to lower dest	109	247
MOVLH	Move lower src to upper dest	109	247
MOVLL	Move lower src to lower dest	109	247
MUL	Signed Multiply	111	249
MULF	Multiply Floating-point	112	251
MULU	Unsigned Multiply	113	253
MULX	Signed Multiply and Round	114	255
NEG	Negate	116	255
NOP	No Operation	117	256
NOT	Logical Complement	118	256
OR	Or Logical	119	257
POP	Pop Data off the Stack	120	258
POPC	Pop Control Register off the Stack	121	259

Table C.1 Instruction Set - Alphabetical (4 / 5)

Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
POPM	Pop Registers off the Stack	122	259
PUSH	Push Data on the Stack	123	260
PUSHA	Push Effective Address on the Stack	125	262
PUSHC	Push Control Register on the Stack	126	262
PUSHM	Push Registers on the Stack	127	262
REIT	Return from Interrupt	128	263
RMPA	Repeat Multiply and Accumulation	129	263
ROLC	Rotate the bits to the Left with Carry	130	263
RORC	Rotate the bits to the Right with Carry	131	264
ROT	Rotate the bits	132	264
ROUND	Round Floating-point to Integer	133	265
RTS	Return from Subroutine	134	266
SBB	Subtract with Borrow	135	267
SCCnd	Store Condition Conditionally (conditions listed below)	136	268
SCC	Store 1 if the C flag is 1	136	268
SCEQ	Store 1 if equal	136	268
SCGE	Store 1 if greater as signed integers or equal	136	268
SCGEU	Store 1 if greater as unsigned integers or equal	136	268
SCGT	Store 1 if greater as signed integers	136	268
SCGTU	Store1 if greater as unsigned integers	136	268
SCLE	Store 1 if less as signed integers or equal	136	268
SCLEU	Store 1 if less as unsigned integers or equal	136	268
SCLT	Store 1 if less as signed integers	136	268
SCLTU	Store 1 if less as unsigned integers	136	268
SCN	Store 1 if a negative value	136	268
SCNC	Store 1 if the C flag is 0	136	268
SCNE	Store 1 if not equal	136	268
SCNO	Store 1 if the O flag is 0	136	268
SCNZ	Store 1 if the Z flag is 0	136	268
SCO	Store 1 if the O flag is 1	136	268
SCPZ	Store 1 if a positive value or 0	136	268
SCZ	Store 1 if the Z flag is 1	136	268
SCMPU	Compare Strings until not equal	138	269
SHA	Arithmetic Shift	139	269
SHL	Logical Shift	140	270
SIN	Input Strings	142	272
SMOVB	Move Strings Backward	143	272
SMOVF	Move Strings Forward	144	272

Table C.1 Instruction Set - Alphabetical (5 / 5)

Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
SMOVU	Move Strings While Unequal to Zero	145	273
SOUT	Output Strings	146	273
SSTR	Store Strings	147	274
STC	Store from Control Register	148	274
STCTX	Store Context	149	275
STNZ	Store on Not Zero	151	276
STOP	Stop	152	276
STZ	Store on Zero	153	277
STZX	Store according to Zero Flag	154	277
SUB	Subtract	155	278
SUBF	Subtract Floating-point	156	280
SUNTIL	Search Equal String	157	281
SWHILE	Search Unequal String	158	282
TST	Test Logical	159	282
UND	Undefined Instruction Interrupt	160	283
WAIT	Wait	161	283
XCHG	Exchange	162	284
XOR	Exclusive Or Logical	163	284

Table C.2 Instruction Set - Functional (1 / 3)

Function	Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
Move	MOV	Move	105	241
	MOVA	Move Effective Address	108	246
	MOVDir	Move Nibble Data	109	247
	POP	Pop Data off the Stack	120	258
	POPM	Pop Registers off the Stack	122	259
	PUSH	Push Data on the Stack	123	260
	PUSHA	Push Effective Address on the Stack	125	262
	PUSHM	Push Registers on the Stack	127	262
	STNZ	Store on Not Zero	151	276
	STZ	Store on Zero	153	277
	STZX	Store according to Zero Flag	154	277
	XCHG	Exchange	162	284
Bit pro-cessing	BCLR	Clear a Bit	51	191
	BITINDEX	Index next Bit Instruction	52	192
	BMCnd	Move a Bit Conditionally	53	192
	BNOT	Not a Bit	55	193
	BSET	Set a Bit	58	194
	BTST	Test a Bit	59	194
	BTSTC	Test a Bit and Clear	60	195
	BTSTS	Test a Bit and Set	61	196
Shift	ROLC	Rotate the bits to the Left with Carry	130	263
	RORC	Rotate the bits to the Right with Carry	131	264
	ROT	Rotate the bits	132	264
	SHA	Arithmetic Shift	139	269
	SHL	Logical Shift	140	270
Arithmetic operation	ABS	Absolute	42	182
	ADC	Add with Carry	43	182
	ADCF	Add Carry Flag	44	183
	ADD	Add	45	184
	ADSF	Add Sign Flag	48	188
	CLIP	Clip	62	196
	CMP	Compare	63	197
	DEC	Decrement	69	206
	DIV	Signed Divide	70	207
	DIVU	Unsigned Divide	72	211

Table C.2 Instruction Set - Functional (2 / 3)

Function	Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
Arithmetic operation	DIVX	Signed Divide extra	73	213
	EDIV	Extended Signed Divide with Remainder	76	219
	EDIVU	Extended Unsigned Divide with Remainder	77	220
	EDIVX	Extended Signed Divide extra with Remainder	78	221
	EMUL	Extended Signed Multiply	79	222
	EMULU	Extended Unsigned Multiply	80	223
	EXTS	Sign Extend	84	224
	EXTZ	Zero Extend	85	226
	INC	Increment	90	229
	MAX	Select Maximum Value	103	237
	MIN	Select Minimum Value	104	239
	MUL	Signed Multiply	111	249
	MULU	Unsigned Multiply	113	253
	MULX	Signed Multiply and Round	114	255
	NEG	Negate	116	255
	SBB	Subtract with Borrow	135	267
	SUB	Subtract	155	278
Decimal operation	DADC	Add Decimal with Carry	67	203
	DADD	Add Decimal	68	205
	DSBB	Subtract Decimal with Borrow	74	215
	DSUB	Subtract Decimal	75	217
Froating point operation	ADDF	Add Floating-point	47	187
	CMPF	Compare Floating-point	65	199
	CNVIF	Convert Integer to Floating-point	66	201
	DIVF	Divide Floating-point	71	209
	MULF	Multiply Floating-point	112	251
	ROUND	Round Floating-point to Integer	133	265
	SUBF	Subtract Floating-point	156	280
Sum of products operation	RMPA	Repeat Multiply and Accumulation	129	263
Logical operation	AND	And Logical	49	189
	NOT	Logical Complement	118	256
	OR	Or Logical	119	257
	TST	Test Logical	159	282
	XOR	Exclusive Or Logical	163	284

Table C.2 Instruction Set - Functional (3 / 3)

Function	Mnemonic	Instruction Full Name	Page for Function	Page for Instruction Codes/ Number of Cycles
Jump	JCnd	Jump Conditionally	94	232
	JMP	Jump Always	95	232
	JMPI	Jump Indirectly	96	233
	JSR	Jump to Subroutine	97	234
	JSRI	Jump to Subroutine Indirectly	98	234
	RTS	Return from Subroutine	134	266
String	SCMPU	Compare Strings until not equal	138	269
	SIN	Input Strings	142	272
	SMOVB	Move Strings Backward	143	272
	SMOVF	Move Strings Forward	144	272
	SMOVU	Move Strings While Unequal to Zero	145	273
	SOUT	Output Strings	146	273
	SSTR	Store Strings	147	274
	SUNTIL	Search Equal String	157	281
	SWHILE	Search Unequal String	158	282
Control register operation	FCLR	Clear a Flag	87	228
	FSET	Set a Flag	89	229
	LDC	Load into Control Register	99	235
	POPC	Pop Control Register off the Stack	121	259
	PUSHC	Push Control Register on the Stack	126	262
	STC	Store from Control Register	148	274
Interrupt	BRK	Break	56	193
	BRK2	Break 2	57	193
	FREIT	Return from Fast Interrupt	88	228
	INT	Interrupt	92	231
	INTO	Interrupt on Overflow	93	231
	LDIPL	Load Interrupt Priority Level	102	237
	REIT	Return from Interrupt	128	263
	UND	Undefined Instruction Interrupt	160	283
High-level language support	ENTER	Enter and Create Stack Frame	81	223
	EXITD	Exit and Deallocate Stack Frame	82	224
	EXITI	Exit Interrupt and Deallocate Stack Frame	83	224
OS support	LDCTX	Load Context	100	236
	STCTX	Store Context	149	275
Others	INDEXType	Index	91	230
	NOP	No Operation	117	256
	SCCnd	Store Condition Conditionally	136	268
	STOP	Stop	152	276
	WAIT	Wait	161	283

1. Overview

1.1 R32C/100 Series Overview

The R32C/100 Series MCU is a single-chip microcomputer which is developed for embedded applications. Having C language supporting instructions and frequently used instructions in one-byte opcodes, the R32C/100 Series MCU enables efficient, memory-saving programs to be developed in both C language and assembly language. Moreover, the R32C/100 Series MCU contains 108 instructions suitable for its versatile addressing modes. In addition to arithmetic/logic operations on a bit or 4-bit data, this powerful instruction set allows register-to-register, register-to-memory, and memory-to-memory operations to be performed.
The R32C/100 Series MCU achieves fast arithmetic processing by using single-clock-cycle instructions. It also enables fast multiplication by using the internal multiplier and floating-point multiplier.

(1) Features

(a) Register set

Data Registers	Four 32-bit data registers (Each register can be used as two 16-bit registers. Two of the four registers can be also used as four 8-bit registers each.)
Address Registers	Four 32-bit address registers
Base Registers	Two 32-bit base registers

(b) Instruction set

Instructions suitable for C language (stack frame operation)	: ENTER, EXITD, etc.
Memory-to-memory instructions	: MOV, ADD, SUB, etc.
Powerful bit processing instructions	: BCLR, BTST, BSET, etc.
4-bit (nibble) data transfer instructions	: MOVLL, MOVHL, etc.
Frequently used 1-byte instructions	: MOV, ADD, SUB, JMP, etc.
Fast, single-clock-cycle instructions	: MOV, ADD, SUB, etc.
Fixed-point multiplication instruction	: MULX
Floating-point instructions	: ADDF, SUBF, MULF, DIVF, etc.

(c) 4G-byte linear address space

Relative jump instructions according to jump distance

(d) Fast instruction execution

Single-clock-cycle instructions: 36 out of 108 instructions are executed in a single-clock-cycle.

(2) Performance (operating at 100 MHz)

Register-to-register transfer	10 ns
Register-to-memory transfer	20 ns
Register-to-register addition/subtraction	10 ns
8-bit × 8-bit register-to-register multiplication	20 ns
16-bit × 16-bit register-to-register multiplication	20 ns
32-bit × 32-bit register-to-register multiplication	20 ns
16 / 16-bit register-to-register division	150 ns
32 / 32-bit register-to-register division	220 ns

1.2 Address Space

Figure 1.1 shows the address map.
Two special function register (SFR) spaces, SFR1 and SFR2, are mapped to addresses 00000000h to 000003FFh, and 00040000h to 0004FFFFh, respectively.
Memory is mapped to addresses 00000400h to 0003FFFFh, 00060000h to 0007FFFFh, and FFE00000h to FFFFFFFFh.
The RAM space extends from address 00000400h to higher addresses, and the ROM space extends from FFFFFFFFh to lower addresses. The fixed vector space is fixed from addresses FFFFFFDCh to FFFFFFFFh.

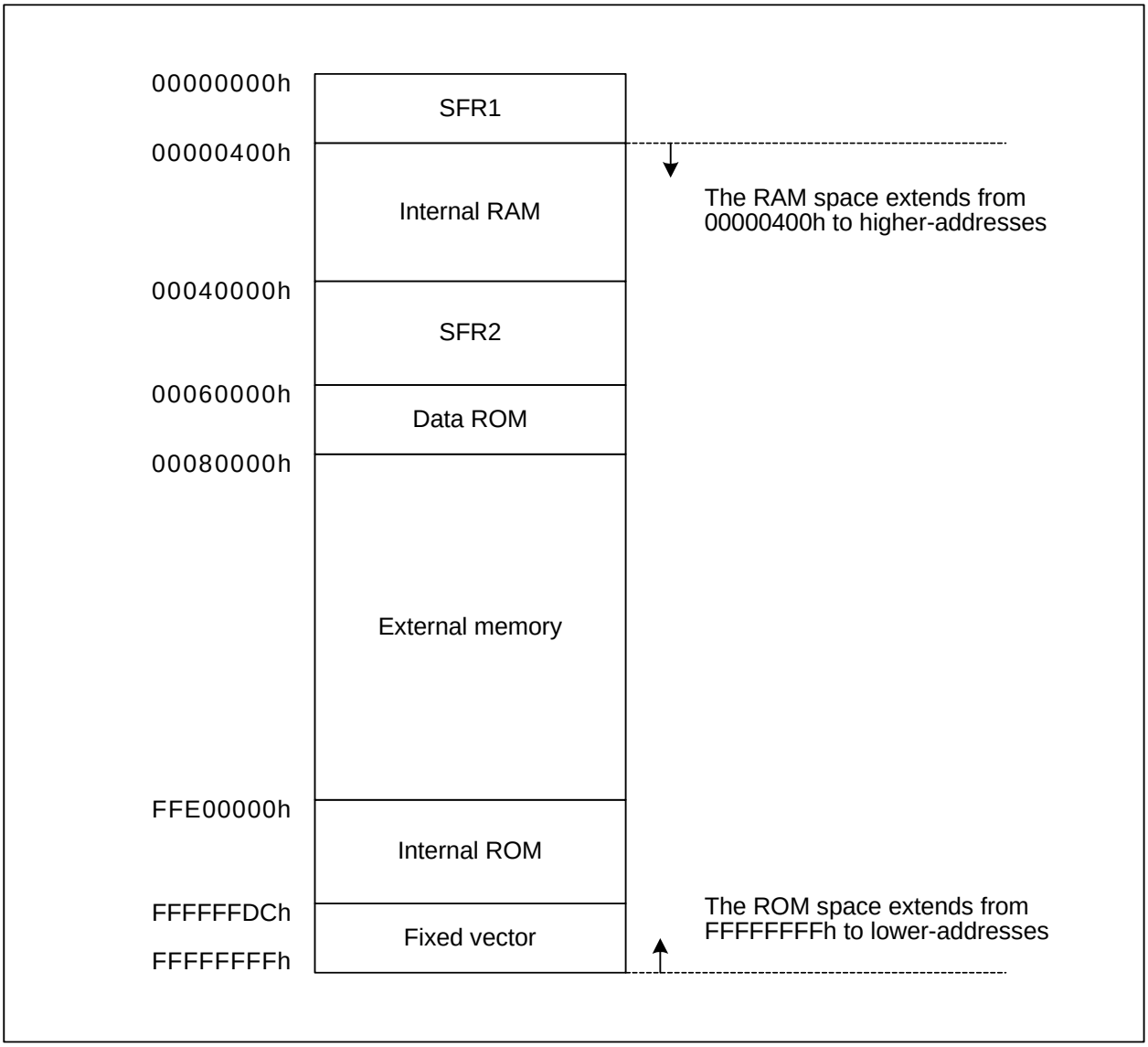


Figure 1.1 Address Map

1.3 Register Set

The central processing unit (CPU) contains registers as shown in Figure 1.2. There are two register banks each consisting of R2R0, R3R1, R6R4, R7R5, A0, A1, A2, A3, SB, and FB registers. Use the register bank select flag (B flag in the flag register) to switch from one register bank to the other.

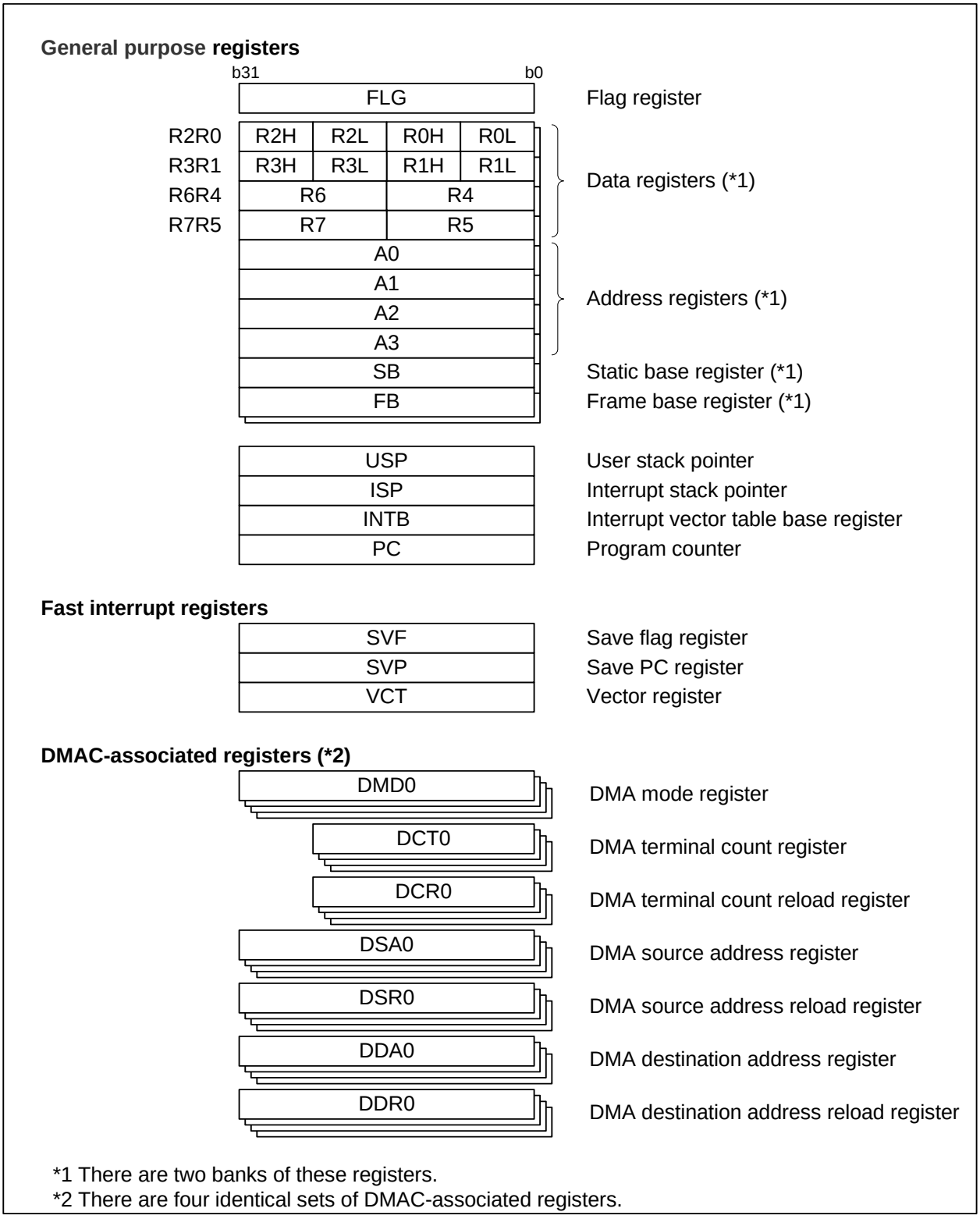


Figure 1.2 CPU Register Configuration

1.3.1 General Purpose Registers

There are nine kinds of general purpose registers.

(1) Data Registers (R2R0, R3R1, R6R4, R7R5, R0, R0H, R0L, R1, R1H, R1L, R2, R2H, R2L, R3, R3H, R3L, R4, R5, R6, and R7)

These 32-bit registers (R2R0, R3R1, R6R4, and R7R5) are primarily used for data transfers, and arithmetic and logic operations.
Each of the registers can be divided into upper and lower 16-bit registers, e.g. R2R0 can be divided into R2 and R0, R3R1 can be divided into R3 and R1, etc.
Moreover, data registers R2R0 and R3R1 can be divided into four 8-bit data registers: upper (R2H, and R3H), upper-middle (R2L, and R3L), lower-middle (R0H, and R1H) and lower (R0L, and R1L).

(2) Address Registers (A0, A1, A2, and A3)

These 32-bit registers have functions similar to the data registers. They are also used for address register indirect addressing and address register relative addressing.

(3) Static Base Register (SB)

This 32-bit register is used for SB relative addressing.

(4) Frame Base Register (FB)

This 32-bit register is used for FB relative addressing.

(5) Program Counter (PC)

This 32-bit counter indicates the address of the instruction to be executed next.

(6) Interrupt Vector Table Base Register (INTB)

This 32-bit register indicates the start address of the relocatable vector table.

(7) User Stack Pointer (USP) and Interrupt Stack Pointer (ISP)

Two types of 32-bit stack pointers are provided: a user stack pointer (USP) and an interrupt stack pointer (ISP).
Use the stack pointer select flag (U flag) to select either USP or ISP. The U flag is bit 7 of the flag register (FLG).
For a faster interrupt sequence, set the USP or ISP to a multiple of 4.

(8) Flag Register (FLG)

Out of the 32 bits that form this register, 16 bits are reserved and the other 16 bits are used as flags. Refer to **1.3.4, “Flag Register (FLG)”** for details of each flag function.

1.3.2 Fast Interrupt Registers

The following three registers are provided to achieve faster interrupt response. They are accessed in the interrupt sequence instead of the stack area, which results in reducing the execution time of the interrupt sequence.

(1) Save Flag Register (SVF)

This 16-bit register is used to save the flag register (FLG) when a fast interrupt is generated.

(2) Save PC Register (SVP)

This 32-bit register is used to save the program counter (PC) when a fast interrupt is generated.

(3) Vector Register (VCT)

This 32-bit register is used to indicate the jump address when a fast interrupt is generated.

1.3.3 DMAC-associated Registers

There are seven types of DMAC-associated registers.

(1) DMA Mode Registers (DMD0, DMD1, DMD2, and DMD3)

These 32-bit registers are used to set DMA transfer mode, bit rate etc.

(2) DMA Terminal Count Registers (DCT0, DCT1, DCT2, and DCT3)

These 32-bit registers are used to set DMA transfer counting.

(3) DMA Terminal Count Reload Registers (DCR0, DCR1, DCR2, and DCR3)

These 24-bit registers are used to set reload values for DMA terminal count registers.

(4) DMA Source Address Registers (DSA0, DSA1, DSA2, and DSA3)

These 32-bit registers are used to set DMA source addresses.

(5) DMA Source Address Reload Registers (DSR0, DSR1, DSR2, and DSR3)

These 32-bit registers are used to set reload values for the DMA source address registers.

(6) DMA Destination Address Registers (DDA0, DDA1, DDA2, and DDA3)

These 32-bit registers are used to hold DMA destination addresses.

(7) DMA Destination Address Reload Registers (DDR0, DDR1, DDR2, and DDR3)

These 32-bit registers are used to set reload values for the DMA destination address registers.

1.3.4 Flag Register (FLG)

Figure 1.3 shows the structure of the flag register (FLG).

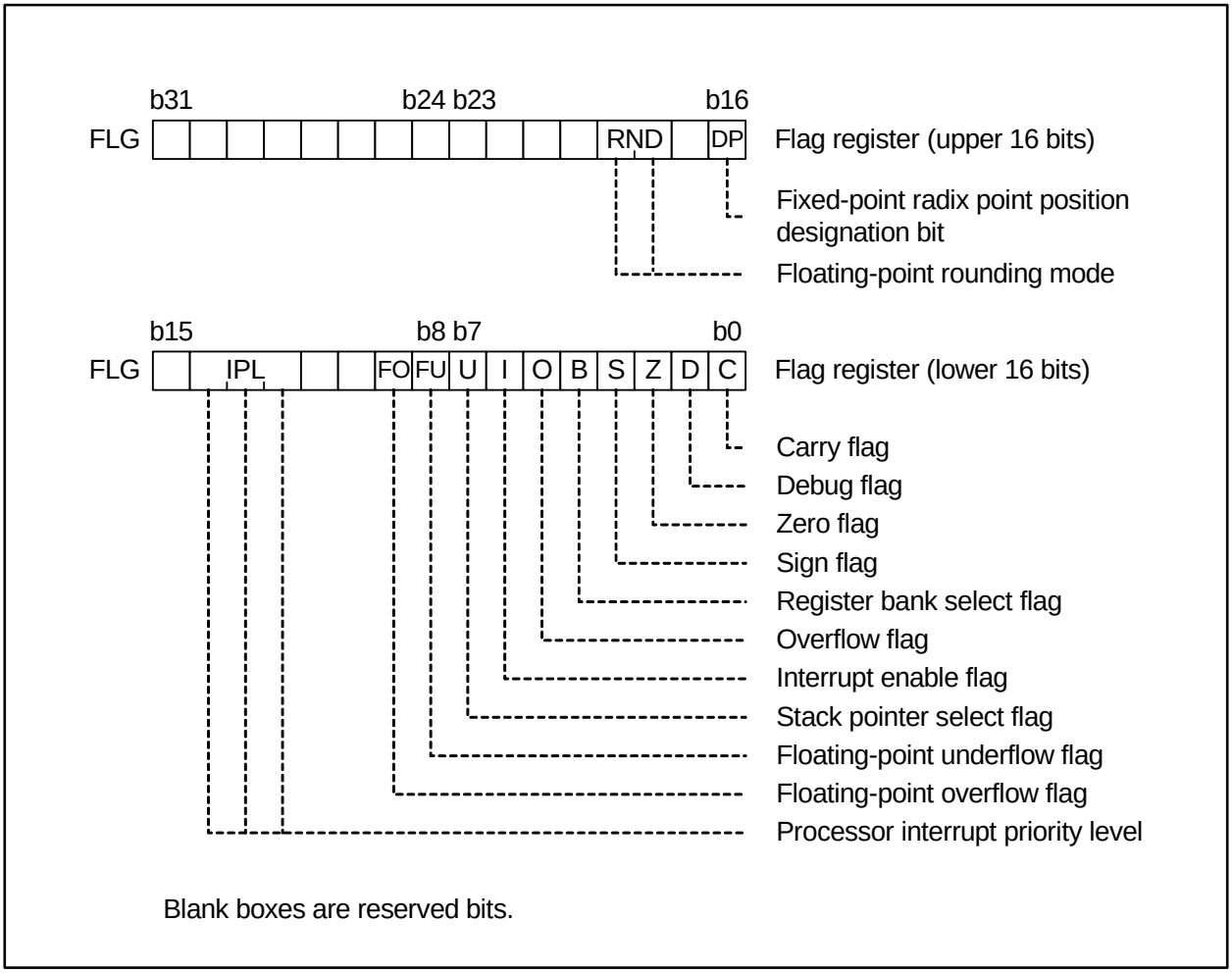


Figure 1.3 Structure of Flag Register (FLG)

The function of each flag is shown as below.

(1) Bit 0: Carry Flag (C Flag)

This flag indicates that a carry, borrow, or shifted-out bit etc. has generated in the arithmetic logic unit (ALU)

(2) Bit 1: Debug Flag (D Flag)

This flag enables a single-step interrupt. By setting this flag to 1, a single step interrupt is generated after executing an instruction.
It becomes 0 when the interrupt is accepted.

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

This flag becomes 1 when the result of an operation is 0; otherwise, it is 0.

(4) Bit 3: Sign Flag (S Flag)

This flag becomes 1 when the result of an operation is a negative value; otherwise, it is 0.

(5) Bit 4: Register Bank Select Flag (B Flag)

This flag selects a register bank. To select the register bank 0, set this bit to 0. To select the register bank 1, set this bit to 1.

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

This flag becomes 1 when an overflow occurs in an operation; otherwise, it is 0.

(7) Bit 6: Interrupt Enable Flag (I Flag)

This flag enables maskable interrupts.
To disable maskable interrupts, set this flag to 0. To enable them, set this flag to 1. When an interrupt is accepted, the flag becomes 0.

(8) Bit 7: Stack Pointer Select Flag (U Flag)

To select the interrupt stack pointer (ISP), set this flag to 0. To select the user stack pointer (USP), set this flag to 1.
It becomes 0 when a hardware interrupt is accepted or when an INT instruction of a software interrupt numbered 0 to 127 is executed.

(9) Bit 8: Floating-point Underflow Flag (FU Flag)

This flag becomes 1 when an underflow occurs in a floating-point operation; otherwise, it is 0.
It also becomes 1 when the operand has invalid numbers (subnormal numbers).

(10) Bit 9: Floating-point Overflow Flag (FO Flag)

This flag becomes 1 when an overflow occurs in a floating-point operation; otherwise, it is 0.
It also becomes 1 when the operand has invalid numbers (subnormal numbers).

(11) Bit 10 and Bit 11: Reserved Bits

(12) Bit 12 to Bit 14: Processor Interrupt Priority Level (IPL)

The 3-bit processor interrupt priority level selects a processor interrupt priority level from level 0 to 7.
An interrupt is acceptable when the interrupt request level is higher than the IPL.
When the processor interrupt priority level (IPL) is set to level 7 (111b), all interrupts are disabled.

(13) Bit 15: Reserved Bit

(14) Bit 16: Fixed-point Radix Point Position Designation Bit (DP bit)

This bit designates the radix point. It also specifies which portion of the fixed-point multiplication result to take. This bit is used in the MULX instruction.

(15) Bit 17: Reserved Bit

(16) Bit 18 and Bit 19: Floating-point Rounding Mode (RND)

The 2-bit floating-point rounding mode (RND) selects a rounding mode for floating-point calculation results.

RND	Rounding Mode	Description
00b	Round to nearest	Round to the nearest value. When the number falls midway, it is rounded to the nearest value with an even least significant bit (b0 = 0).
01b	N/A	Reserved
10b	Round toward - infinity	Rounds toward negative infinity.
11b	Round toward 0	Rounds toward zero.

(17) Bit 20 to Bit 31: Reserved Bits

1.3.5 Register Bank

The R32C/100 Series MCU has two register banks, each consists of data registers (R2R0, R3R1, R6R4, and R7R5), address registers (A0, A1, A2, and A3), frame base register (FB), and static base register (SB). Use the register bank select flag (B flag) of the flag register (FLG) to switch from one register bank to the other.
Figure 1.4 shows the configuration of the register banks.

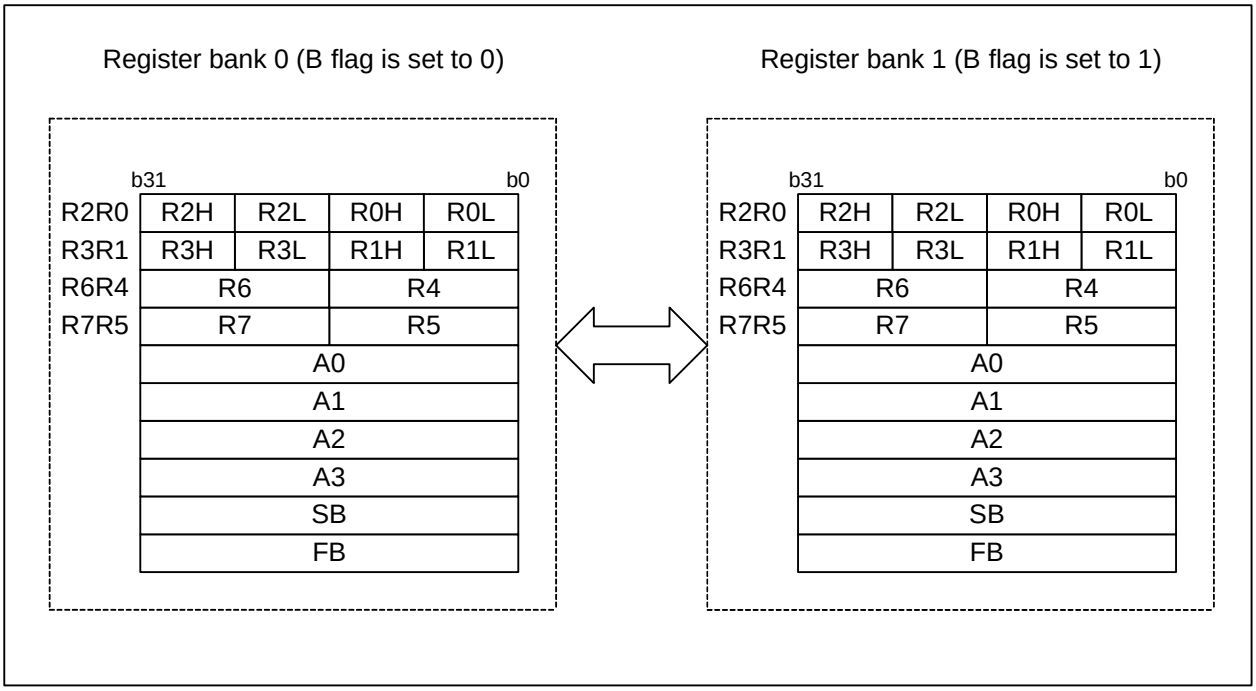


Figure 1.4 Configuration of Register Banks

The R32C/100 Series MCU supports an addressing mode that can access register bank 1 even if the B flag is 0.

1.3.6 Internal Register Values

After the reset is released, each register's value is as follows:

- Data registers (R2R0, R3R1, R6R4, and R7R5) : 00000000h
- Address registers (A0, A1, A2, and A3) : 00000000h
- Static base register (SB) : 00000000h
- Frame base register (FB) : 00000000h
- Interrupt vector table base register (INTB) : 00000000h
- User stack pointer (USP) : 00000000h
- Interrupt stack pointer (ISP) : 00000000h
- Flag register (FLG) : 00000000h
- DMA mode registers (DMD0, DMD1, DMD2, and DMD3) : 00000000h
- DMA terminal count registers (DCT0, DCT1, DCT2, and DCT3) : Undefined
- DMA terminal count reload registers (DCR0, DCR1, DCR2, and DCR3) : Undefined
- DMA source address registers (DSA0, DSA1, DSA2, and DSA3) : Undefined
- DMA source address reload registers (DSR0, DSR1, DSR2, and DSR3) : Undefined
- DMA destination address registers (DDA0, DDA1, DDA2, and DDA3) : Undefined
- DMA destination address reload registers (DDR0, DDR1, DDR2, and DDR3) : Undefined
- Save flag register (SVF) : Undefined
- Save PC register (SVP) : Undefined
- Vector register (VCT) : Undefined

1.4 Data Types

There are six types of data: integer, decimal, fixed-point number, floating-point number, bit, and string.

1.4.1 Integer

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

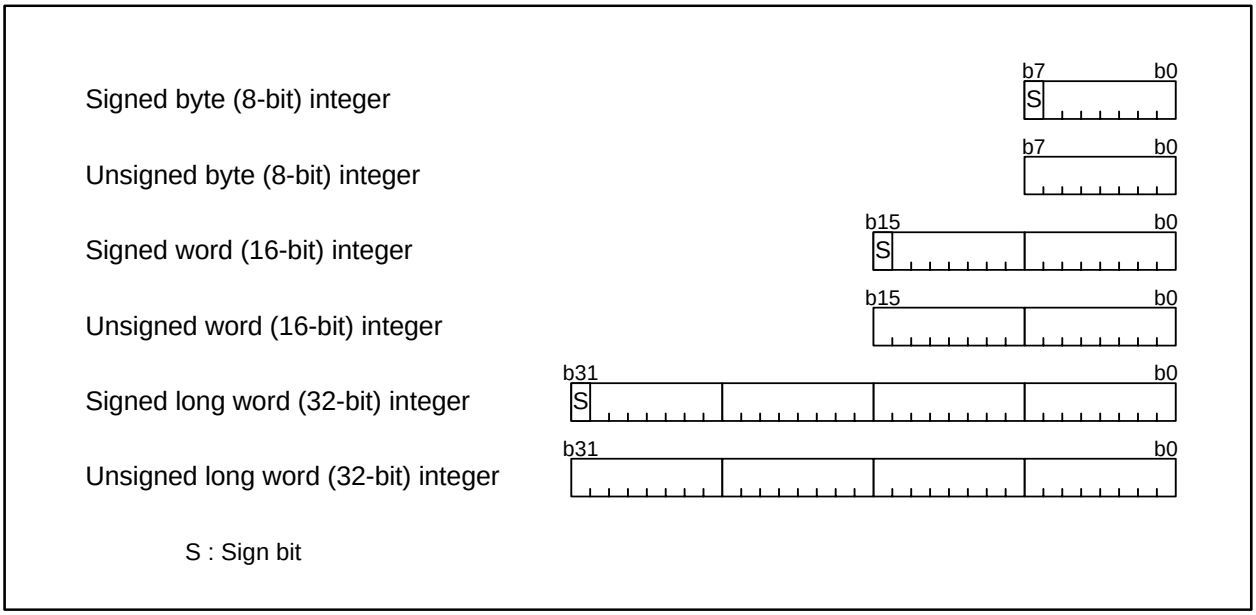


Figure 1.5 Integer

1.4.2 Decimal

Binary-coded decimal (BCD) is used. Four types of instructions can be performed on this data type: DADC, DADD, DSBB, and DSUB.

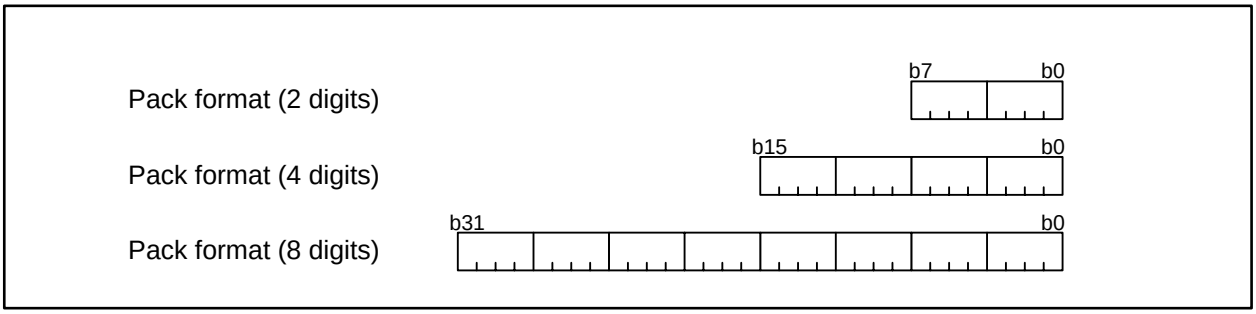


Figure 1.6 Decimal

1.4.3 Fixed-point Number

Usually, multiplication of fixed-point numbers is performed by combining integer multiply and bit shift instructions. The R32C/100 Series MCU provides a unique, exclusive instruction for fixed-point multiplication: MULX instruction.
The fixed-point numbers supported by MULX are as shown in Figure 1.7. There is no radix point for 8-bit fixed-point data when DP = 1.

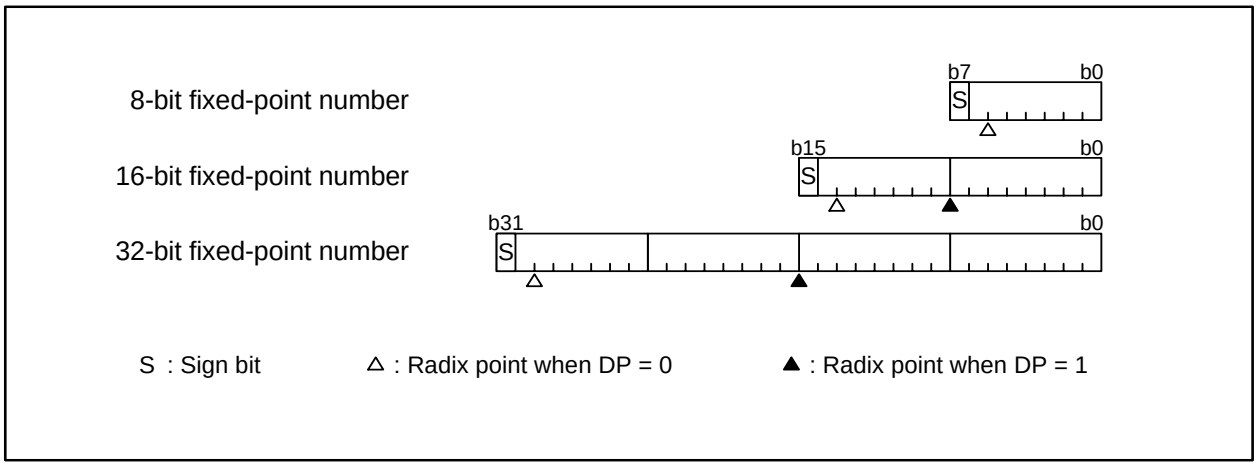


Figure 1.7 Fixed-point Number

1.4.4 Floating-point Number

The R32C/100 Series MCU supports IEEE 754 single precision floating-point format.
Seven types of logical instructions can operate on this data type: ADDF, CMPF, CNVIF, DIVF, MULF, ROUND and SUBF.

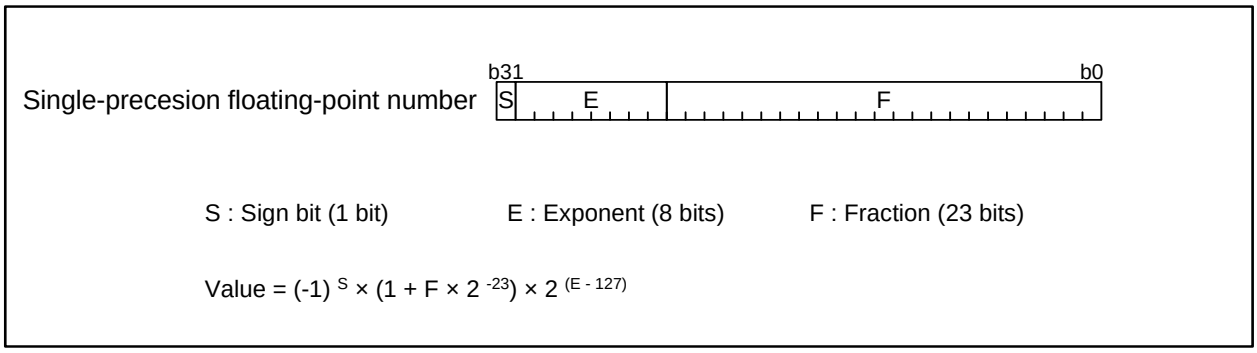


Figure 1.8 Floating-point Number

Floating-point number supports the values below:
- 0 < E < 255 (normal numbers)
- E = 0 and F = 0 (signed zero)

It does not support the values below:
- E = 0 and F > 0 (subnormal numbers)
- E = 255 and F = 0 (infinity)
- E = 255 and F > 0 (NaN: not-a-number)

1.4.5 Bit

Seven types of instructions can operate on this data type: BCLR, BSET, BNOT, BTST, BM*Cnd*, BTSTS and BTSTC.
Each register bit of the 8-bit registers (R0L, R0H, R1L, R1H, R2L, R2H, R3L, and R3H) can be specified by a register name and a bit number from 0 to 7.
Each memory bit can be also specified by a target address and a bit number from 0 to 7. Six types of addressing mode are available: absolute addressing (BTST:S only), sign-extended absolute, address register indirect addressing, address register relative addressing, SB relative addressing, and FB relative addressing.

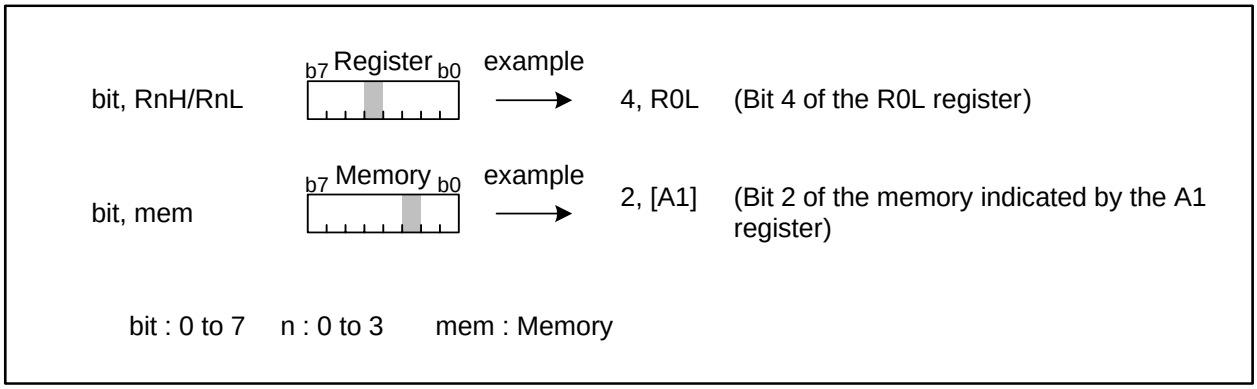


Figure 1.9 Bit

1.4.6 String

A string is a type of data that consists of a sequence of bytes (8-bit), words (16-bit) or long words (32-bit) data.
Nine types of instructions can operate on this data type: SMOVB, SMOVF, SMOVU, SCMPU, SIN, SOUT, SSTR, SUNTIL, and SWHILE.

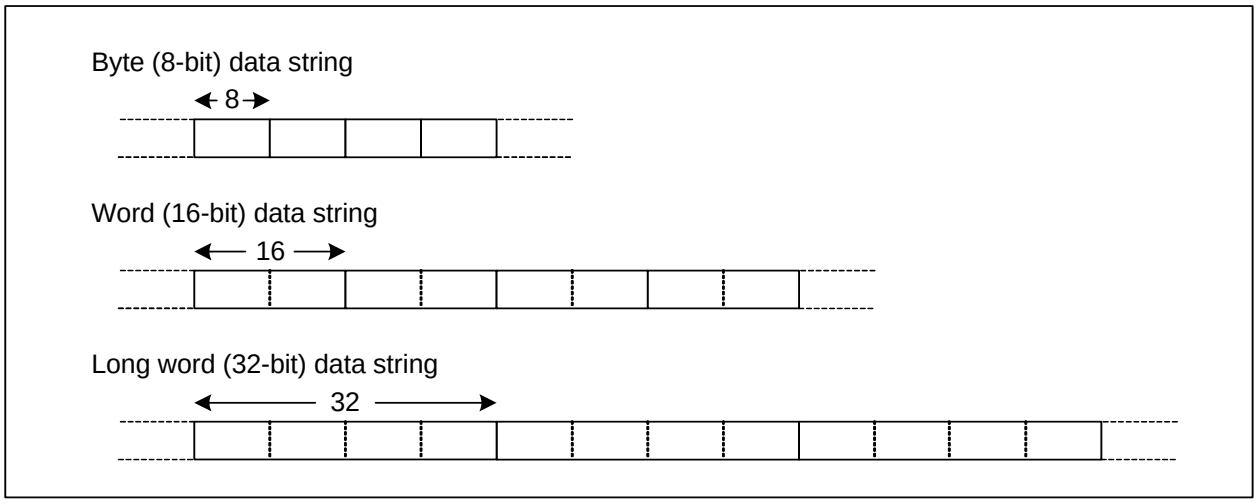


Figure 1.10 String

1.5 Data Configuration

1.5.1 Data Configuration in Register

Figure 1.11 shows the correlation between a register's data size and bit numbers.

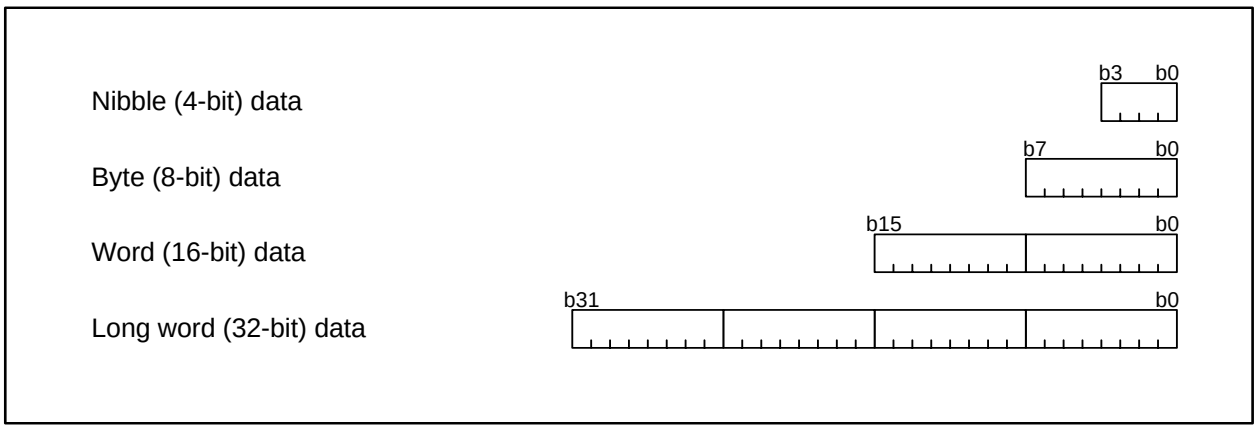


Figure 1.11 Data Configuration in Register

1.5.2 Data Configuration in Memory

Figure 1.12 shows data organization in memory.

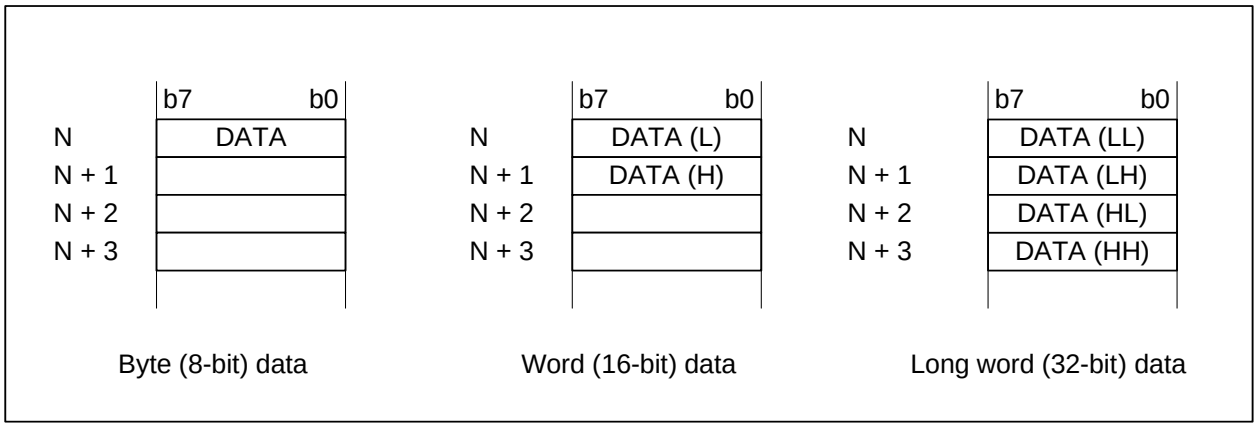


Figure 1.12 Data Configuration in Memory

1.6 Instruction Formats

The R32C/100 Series MCU instructions vary in size from 1 to 14 bytes as shown in Figure 1.13. The opcode determines operation, operand type and number, and instruction length.

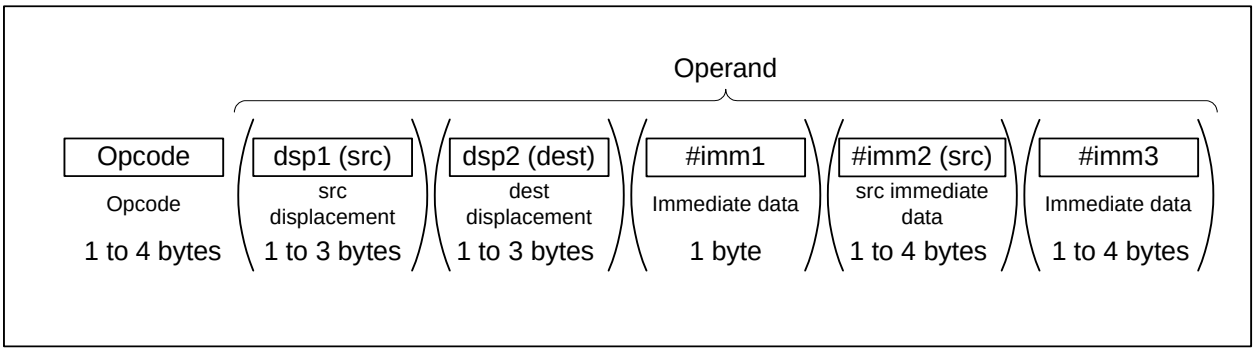


Figure 1.13 Instruction Format

In the R32C/100 Series MCU, instruction codes are defined in a way that frequently used instructions can have fewer bytes. Moreover, some instructions can use more byte-saving formats by limiting their available addressing modes.
There are four types of instruction formats: generic, quick, short, and zero.
The following describes the features of each format.

(1) Generic Format (:G)

An opcode in this format consists of 2 or 3 bytes, and contains information on operation, and each addressing mode of source (hereinafter referred to as src) and destination (hereinafter referred to as dest).
An instruction code consists of an opcode (2 or 3 bytes), a src code (0 to 4 bytes), and a dest code (0 to 3 bytes).

(2) Quick Format (:Q)

An opcode in this format consists of 1 or 2 bytes, and contains information on operation, immediate data, and addressing mode of dest. Note that the immediate data in this opcode is limited to a value that can be expressed by 3 or 4 bits.
An instruction code consists of an opcode (1 or 2 bytes) containing immediate data and a dest code (0 to 3 bytes).

(3) Short Format (:S)

An opcode in this format consists of 1 or 2 bytes, and contains information on operation, and addressing modes of src and dest. Note that available addressing modes are limited.
An instruction code consists of an opcode (1 or 2 bytes), a src code (0 to 4 bytes), and a dest code (0 to 2 bytes).

(4) Zero Format (:Z)

An opcode in this format consists of one byte, and contains information on operation (plus immediate data) and dest addressing modes. Note that the immediate data is fixed to 0, and that the available addressing modes are limited.
An instruction code consists of an opcode (1 byte) and a dest code (0 to 2 bytes).

1.6.1 Opcode

An opcode is a 1- to 3-byte bit string. When an operand is in either indirect instruction addressing mode or bank 1 register direct mode, the opcode is, with an additional 8 bits at the top, a 2- to 4-byte bit string.

1.6.2 Operand

An operand consists of the following fields: dsp1, dsp2, #imm1, #imm2 and #imm3. Each field exists/no exists depending on instructions or addressing mode.
The src code is either in the dsp1 field or #imm2 field, whereas the dest code is in the dsp2 field. The #imm1 field is used to extend an opcode. However, for instructions CLIP and STZX which have two srs codes, src1 is stored in the #imm2 field and src2 is stored in the #imm3 field.

(1) dsp1 field

The dsp1 field consists of 1 to 3 bytes. This field exists only when the addressing mode of src is absolute addressing or relative addressing (except in stack pointer relative addressing).

(2) dsp2 field

The dsp2 field consists of 1 to 3 bytes. This field exists only when the addressing mode of dest is relative addressing (except program counter relative addressing).

(3) #imm1 field

The #imm1 field consists of 1 byte. This field exists only for some instructions.

BM*Cnd* instruction	8-bit immediate data is located in the #imm1 field, and the lower 4 bits specify the condition code.
LDC/STC instruction	The DMAC associated register and VCT register are specified by the 8-bit immediate data located in the #imm1 field. This field does not exist when accessing the following registers: SB, FB, FLG, SP, ISP, SVF, SVP, or INTB.
MOV dsp:8[SP]	The 8-bit immediate data in the #imm1 field represents displacement dsp:8 for the stack pointer.
PUSHM/POPM instruction	The 8-bit immediate data is put in the #imm1 field. The bits which contain 1 are specified as registers to be saved/restored.
ROT/SHA/SHL instruction	The 8-bit immediate data in the #imm1 field specifies the number of bits the data is shifted.

(4) #imm2 field

The #imm2 field consists of 1, 2 or 4 bytes. This field exists only when the addressing mode of src is immediate addressing or sign-extended immediate addressing.

(5) #imm3 field

The #imm3 field consists of 1, 2 or 4 bytes. This field exists only for some instructions.

CLIP or STZX instruction	Two immediate data are specified for src. The first data is in the #imm2 field and the second one is in the #imm3 field.

1.7 Interrupt Vector Table

The interrupt vector table consists of a fixed vector table and a relocatable vector table.

1.7.1 Fixed Vector Table

The fixed vector table is one part of the interrupt vector table, and is allocated in addresses from FFFFFFDCh to FFFFFFFFh. Figure 1.14 shows the fixed vector table.
The interrupt vector table consists of nine 4-byte interrupt vectors. Each vector has the start address of each interrupt handler.

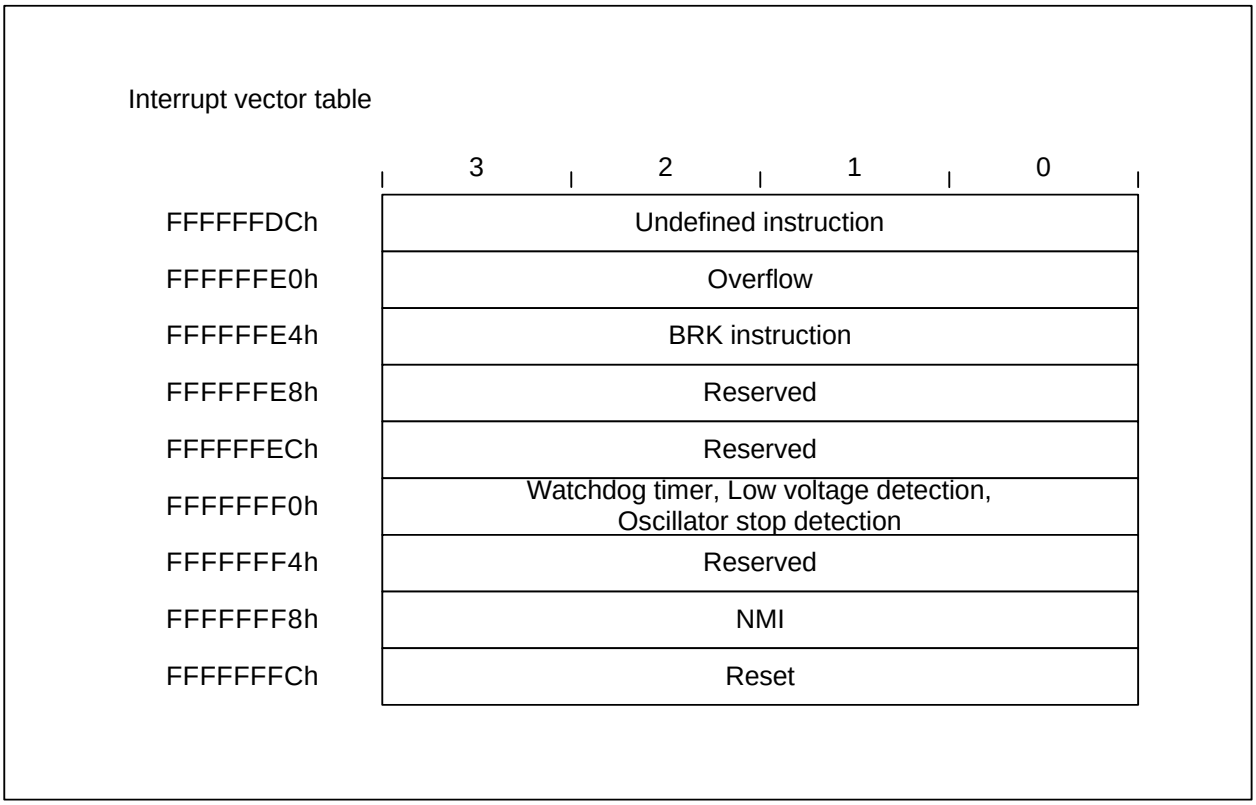


Figure 1.14 Fixed Vector Table

1.7.2 Relocatable Vector Table

The relocatable vector table is another part of the interrupt vector table that is address-variable. Specifically, this vector table is a 1 kbyte, relocatable interrupt vector table whose start address (IntBase) is indicated by the interrupt table register (INTB). Figure 1.15 shows the relocatable vector table.
This vector table consists of 256 4-byte interrupt vectors. Each vector has the start address of interrupt handler. It also has software interrupt numbers from 0 to 255 for the INT instruction.
Peripheral interrupts are also assigned to the relocatable vector table in ascending order from the software interrupt number 0. The number of peripheral interrupts varies with the MCU model. To eliminate overlap with peripheral interrupts, assign the INT instruction interrupts in descending order from software interrupt number 255.
The stack pointer (USP or ISP) to be used for the INT instruction interrupt is determined by the software interrupt number.
For software interrupt numbers 0 to 127, the stack pointer select flag (U flag) is saved when an interrupt request is accepted. Then, the interrupt sequence is executed after the U flag is set to 0 to select the interrupt stack pointer (ISP). The saved U flag is restored upon returning from the interrupt handler.
For the software interrupt numbers 128 to 255, the stack pointer is not switched, so the user stack pointer (USP) is used.
For the peripheral interrupts, the interrupt stack pointer (ISP) is specified when an interrupt request is accepted irrespective of software interrupt numbers.

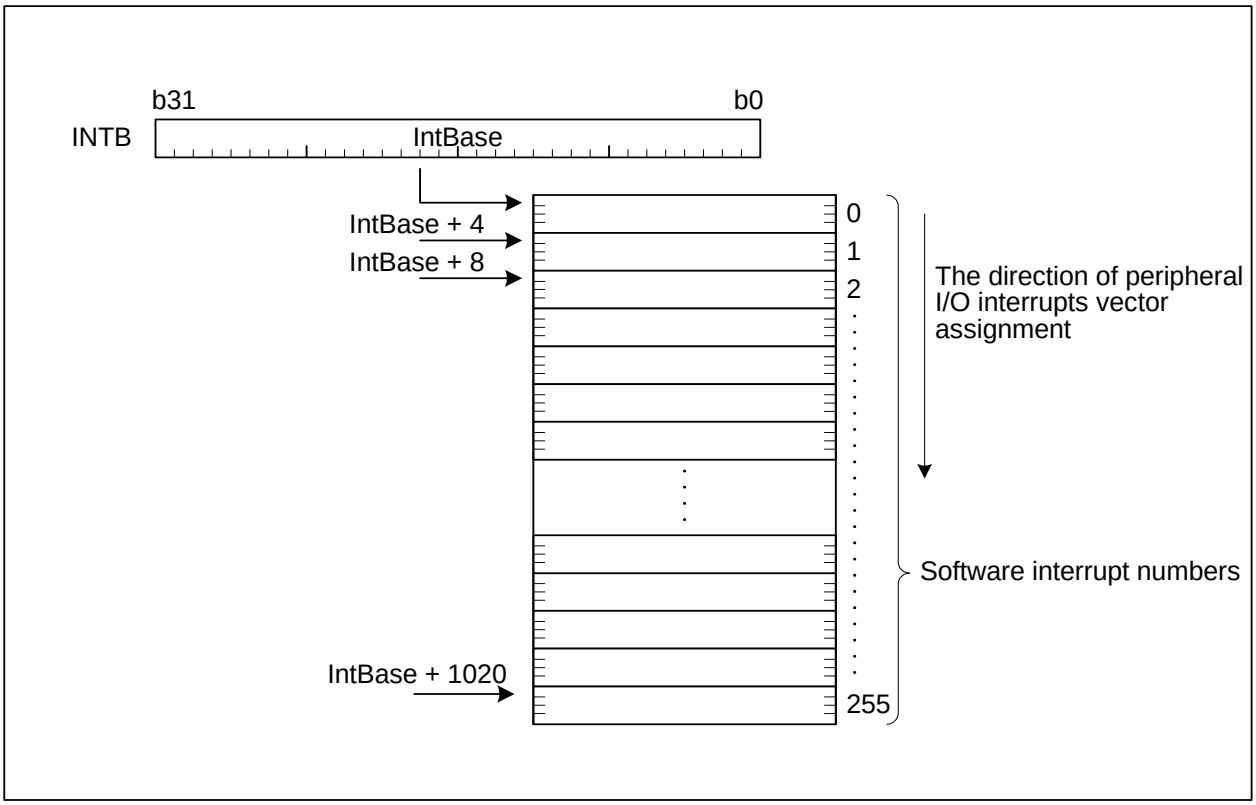


Figure 1.15 Relocatable Vector Table

2. Addressing Mode

2.1 Addressing Mode

This chapter describes symbols and operations for the five addressing modes listed below.

(1) General instruction addressing

This is the most typical addressing mode to access general purpose registers or memory.

- Register Direct
- Immediate
- Sign-Extended Immediate
- Sign-Extended Absolute
- Address Register Indirect
- Address Register Relative
- SB Relative
- FB Relative

(2) Indirect instruction addressing

This addressing mode accesses memory according to data written in the memory. It is available for almost all instructions which support general instruction addressing.

- Sign-Extended Absolute Indirect
- Address Register Indirect Indirect
- Address Register Relative Indirect
- SB Relative Indirect
- FB Relative Indirect

(3) Extended instruction addressing

This addressing mode is extended to reduce code size, and to access registers or memory that general instruction addressing cannot.

- Bank1 Register Direct
- Short Immediate
- Stack Pointer Relative

(4) Special instruction addressing

This addressing mode accesses control registers, flags and memory. Only some instructions such as LDC, FSET, and JMP support this addressing.

- Control Register Direct
- FLG Direct
- Absolute
- Program Counter Relative

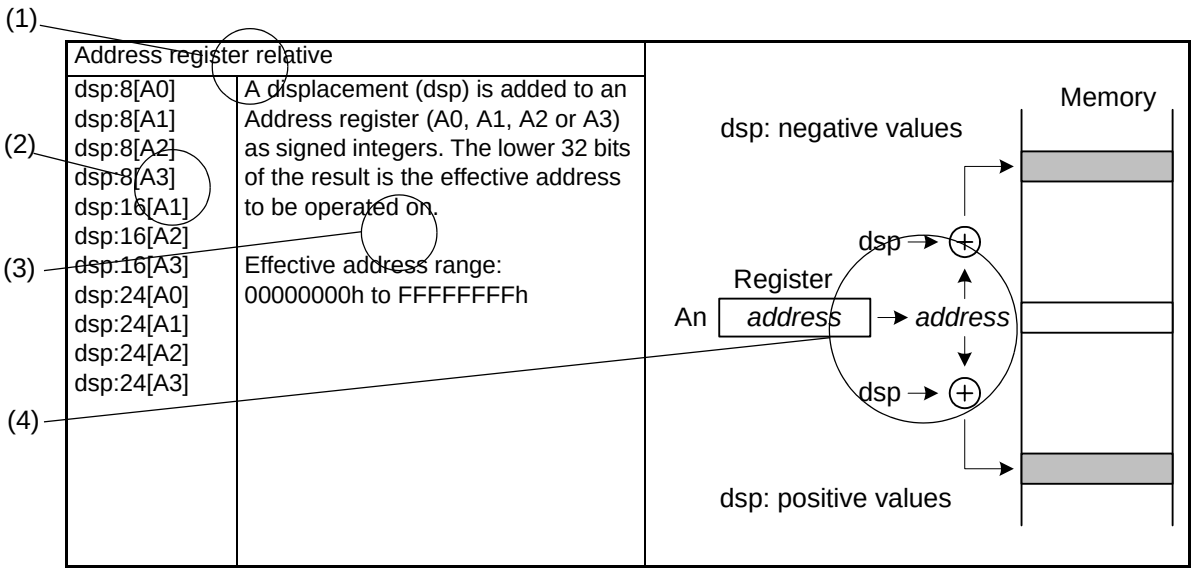
(5) Bit instruction addressing

This addressing mode accesses a bitwise area in general purpose registers or memory.

- Register Direct
- Absolute
- Sign-Extended Absolute
- Address Register Indirect
- Address Register Relative
- SB Relative
- FB Relative

2.2 Guide to This Chapter

The following sample shows how to read this chapter.



(1) Name

Indicates the addressing name.

(2) Symbol

The number of effective bits are shown in the form of a colon followed by a number (e.g. :3, :4, :8, :16, :24, and :32). Note that this element is present only for clarification of the addressing mode and needs not be written. Write a numeric value or a symbol in place of symbols such as dsp.

(3) Explanation

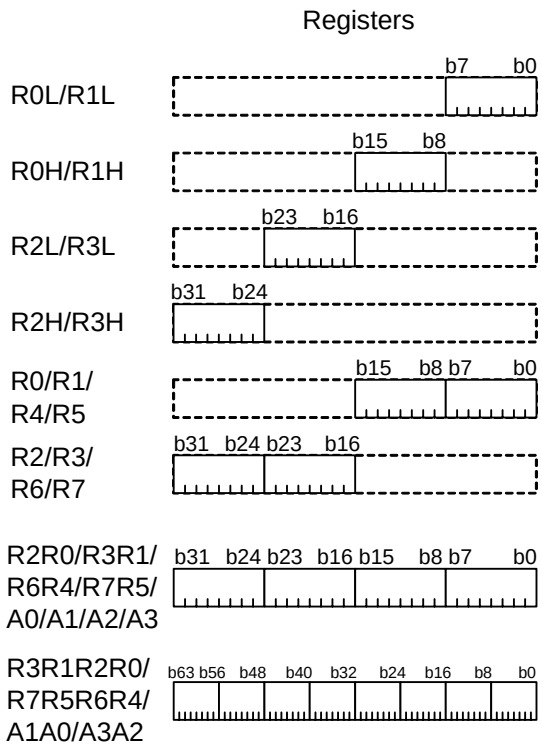
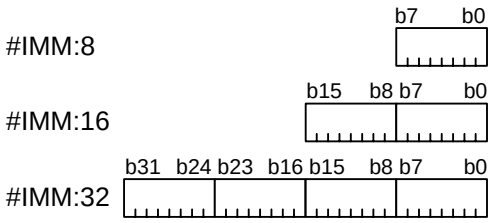
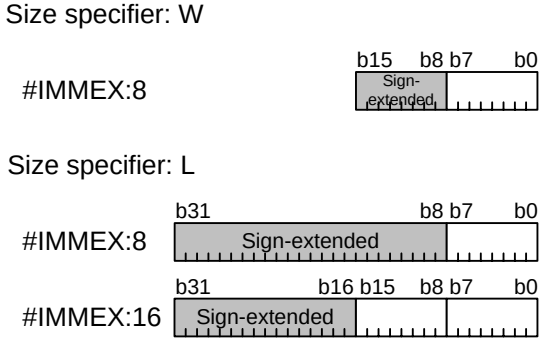
Describes the addressing operation and the effective address range.

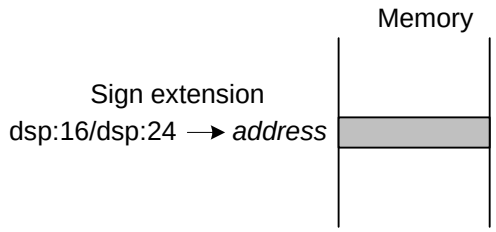
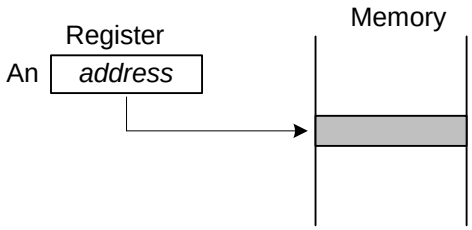
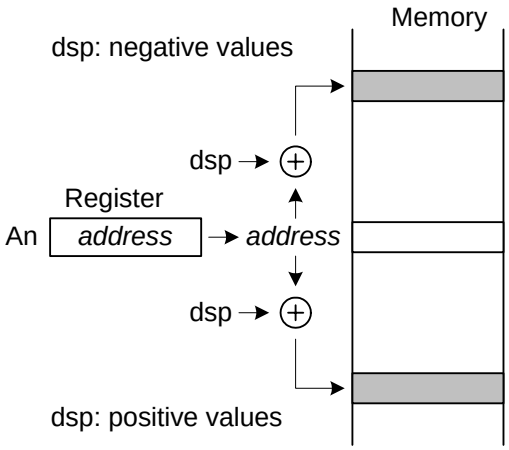
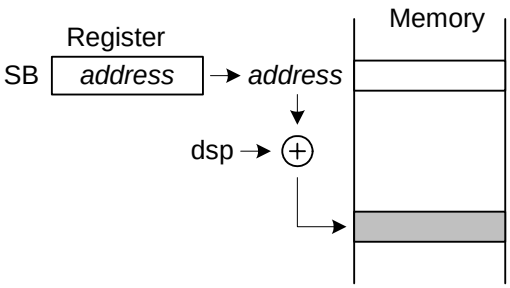
(4) Operation diagram

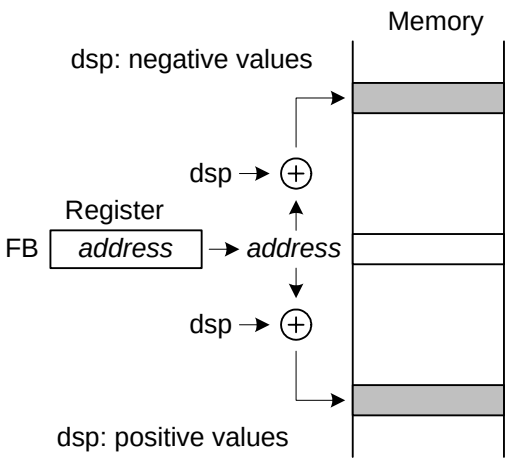
Diagrammatically explains the addressing operation.

2.3 General Instruction Addressing

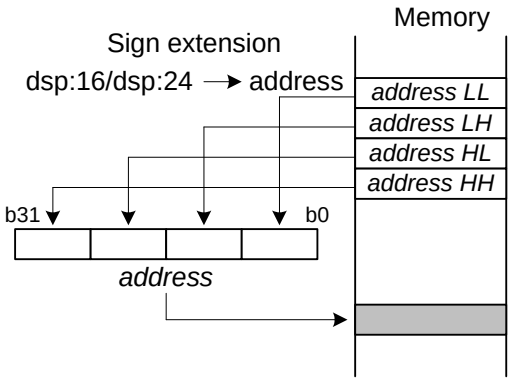
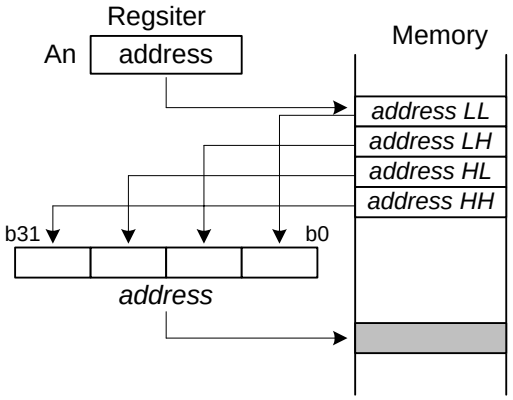
This is the most typical addressing mode to access general purpose registers or memory. It is available for almost all instructions.

Register direct		
R0L R0H R1L R1H R2L R2H R3L R3H R0 R1 R2 R3 R4 R5 R6 R7 R2R0 R3R1 R6R4 R7R5 A0 A1 A2 A3 R3R1R2R0 R7R5R6R4 A1A0 A3A2	The specified register is the object to be operated on.	
Immediate		
#IMM #IMM:8 #IMM:16 #IMM:32	The immediate data indicated by #IMM is the object to be operated on.	
Sign-extended immediate		
#IMMEX #IMMEX:8 #IMMEX:16	The immediate data indicated by #IMMEX which is sign-extended according to size specifier is the object to be operated on.	

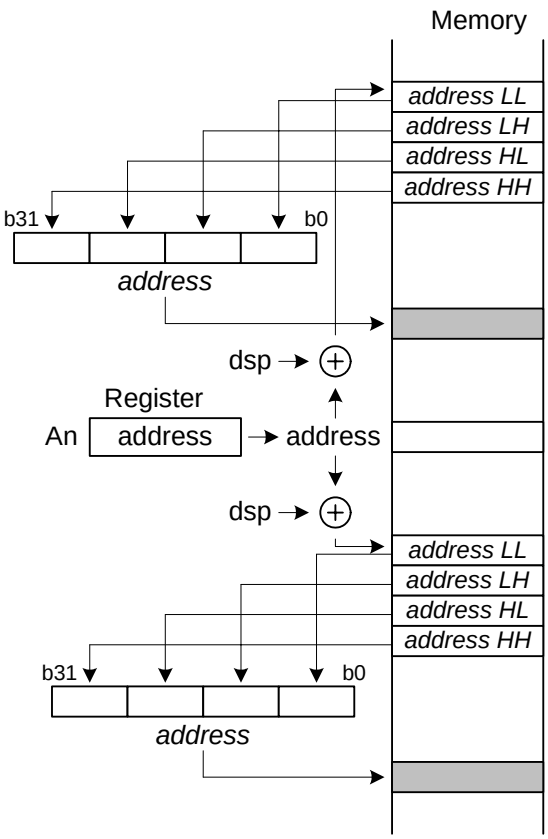
Sign-extended absolute		
dsp:16 dsp:24	A sign-extended displacement (dsp) indicates the effective address to be operated on. Effective address range (dsp: 16): 00000000h to 00007FFFh FFFF8000h to FFFFFFFFh Effective address range (dsp: 24): 00000000h to 007FFFFFh FF800000h to FFFFFFFFh	
Address register indirect		
[A0] [A1] [A2] [A3]	An Address register (A0, A1, A2 or A3) indicates the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	
Address register relative		
dsp:8[A0] dsp:8[A1] dsp:8[A2] dsp:8[A3] dsp:16[A0] dsp:16[A1] dsp:16[A2] dsp:16[A3] dsp:24[A0] dsp:24[A1] dsp:24[A2] dsp:24[A3]	A displacement (dsp) is added to an Address register (A0, A1, A2 or A3) as signed integers. The lower 32 bits of the result is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	
SB relative		
dsp:8[SB] dsp:16[SB] dsp:24[SB]	A displacement (dsp) is added to the Static base register (SB) as unsigned integers. The lower 32 bits of the result is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	

FB relative		
dsp:8[FB] dsp:16[FB]	A displacement (dsp) is added to the Frame base register (FB) as signed integers. The lower 32 bits of the result is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	

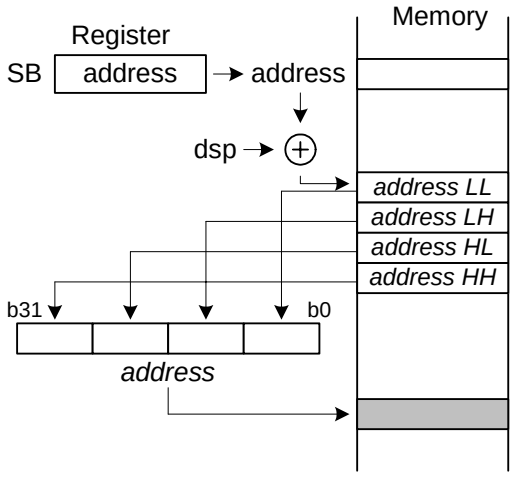
2.4 Indirect Instruction Addressing

Sign-extended absolute indirect		
[dsp:16] [dsp:24]	The 4-byte value which is in an address specified by Sign-extended absolute is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	
Address register indirect indirect		
[[A0]] [[A1]] [[A2]] [[A3]]	The 4-byte value which is in an address specified by Address register indirect addressing is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh	

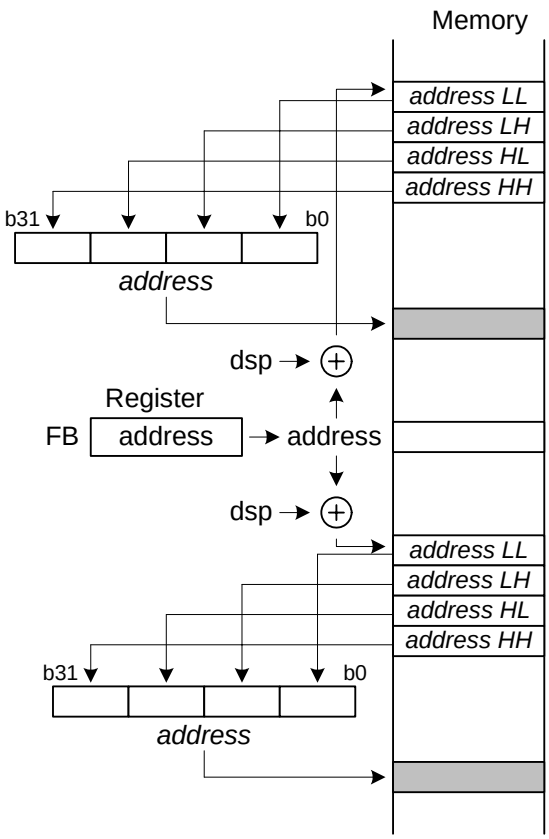
Address register relative indirect	
[dsp:8[A0]] [dsp:8[A1]] [dsp:8[A2]] [dsp:8[A3]] [dsp:16[A0]] [dsp:16[A1]] [dsp:16[A2]] [dsp:16[A3]] [dsp:24[A0]] [dsp:24[A1]] [dsp:24[A2]] [dsp:24[A3]]	The 4-byte value which is in an address specified by Address register relative addressing is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh



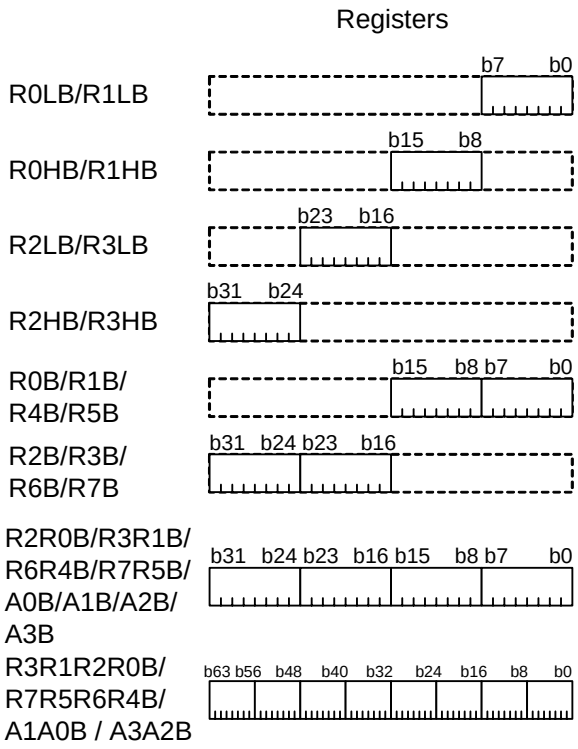
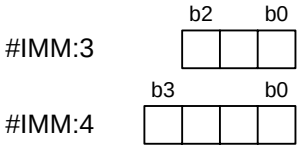
SB relative indirect	
[dsp:8[SB]] [dsp:16[SB]] [dsp:24[SB]]	The 4-byte value which is in an address specified by SB relative addressing is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh

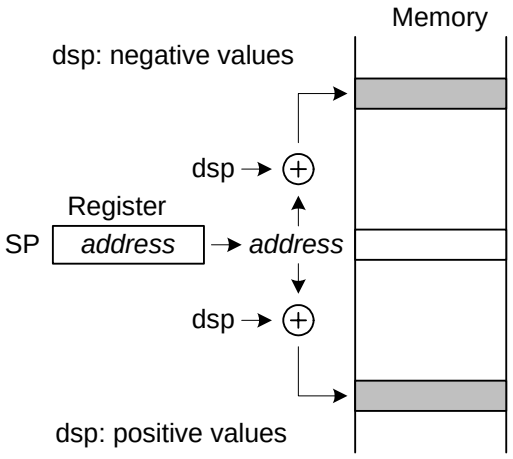


FB relative indirect	
[dsp:8[FB]] [dsp:16[FB]]	The 4-byte value which is in an address specified by FB relative addressing is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh

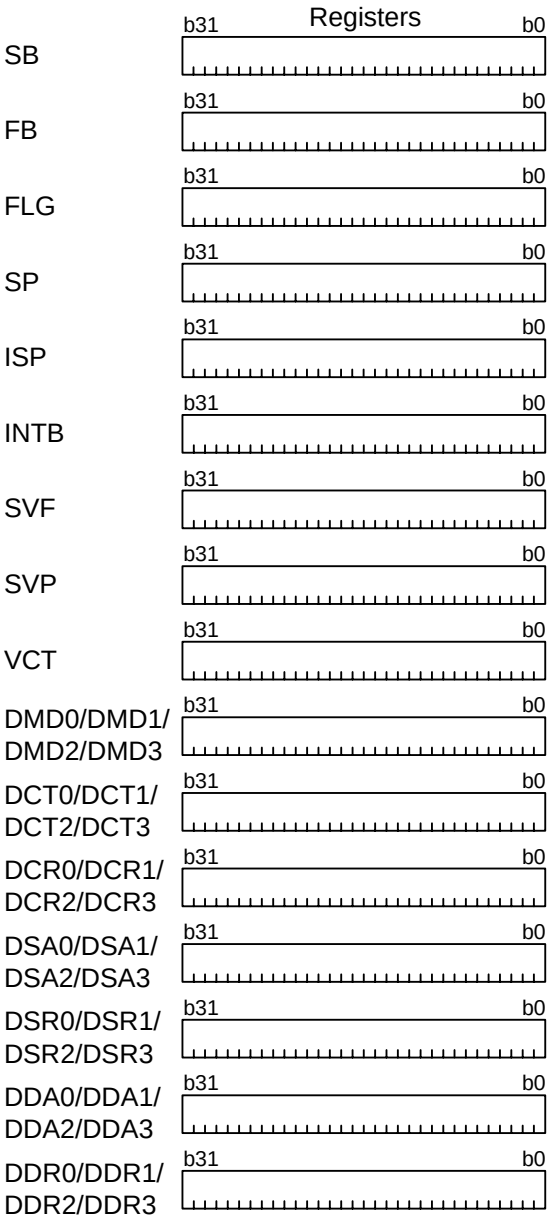
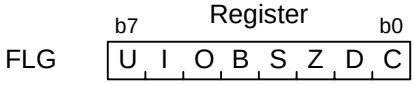


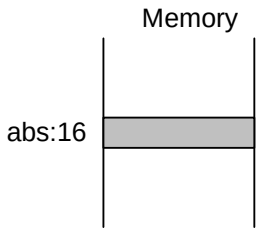
2.5 Extended Instruction Addressing

Bank1 register direct		
R0LB R0HB R1LB R1HB R2LB R2HB R3LB R3HB R0B R1B R2B R3B R4B R5B R6B R7B R2R0B R3R1B R6R4B R7R5B A0B A1B A2B A3B R3R1R2R0B R7R5R6R4B A1A0B A3A2B	The specified Bank1 register is the object to be operated on. Bank1 register direct can be specified irrespective of the B flag value.	
Short immediate		
#0 #IMM:3 #IMM:4	The immediate data indicated by zero or #IMM is the object to be operated on. It is used for zero format or quick format.	

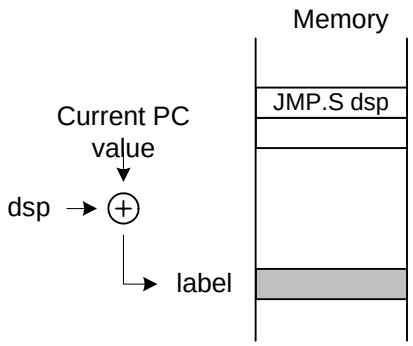
Stack pointer relative		
dsp:8[SP]	A displacement (dsp) is added to the Stack pointer register (SP) as signed integers. The lower 32 bits of the result is the effective address to be operated on. Effective address range: 00000000h to FFFFFFFFh This addressing mode is available in MOV instruction.	

2.6 Special Instruction Addressing

Control register direct		
SB FB FLG SP ISP INTB SVF SVP VCT DMD0 DMD1 DMD2 DMD3 DCT0 DCT1 DCT2 DCT3 DCR0 DCR1 DCR2 DCR3 DSA0 DSA1 DSA2 DSA3 DSR0 DSR1 DSR2 DSR3 DDA0 DDA1 DDA2 DDA3 DDR0 DDR1 DDR2 DDR3	The specified control register is the object to be operated on. When SP is specified, a stack pointer indicated by the U flag is the object to be operated on. This addressing mode is available in ADD:Q, LDC, POPC, PUSHC, and STC instructions.	
FLG direct		
U I O B S Z D C	The specified flag is the object to be operated on. This addressing mode is available in FCLR and FSET instructions.	

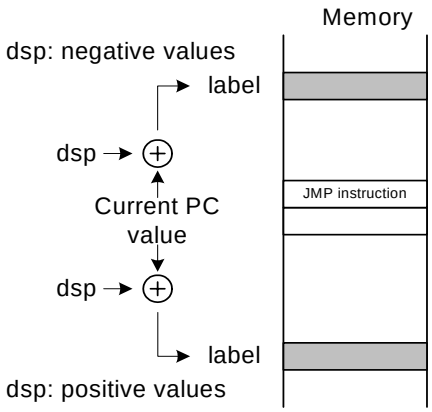
<table>
<tr><td colspan="2">Absolute</td><td rowspan="2"> 
</td></tr>
<tr><td>abs:16</td><td>The value indicated by abs is the effective address to be operated on.

Effective address range:
00000000h to 0000FFFFh

This addressing mode is available in LDCTX and STCTX instructions.</td></tr>
<tr><td colspan="2">Program counter relative</td><td rowspan="2"> 

+1 ≤ dsp ≤ +8
Current PC value = JMP instruction addressing + 1</td></tr>
<tr><td>label
(dsp:3)</td><td>If jump distance specifier (.length) is ".S", the displacement (dsp) is added to the current Program Counter (PC) as unsigned integers. The lower 32 bits of the result is the effective address to be operated on.
A dsp is calculated by the assembler.

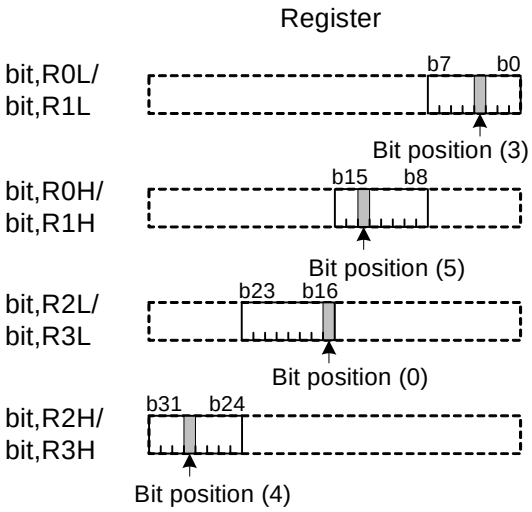
This addressing mode is available in JMP instruction.</td></tr>
<tr><td>label
(dsp:8)
(dsp:16)
(dsp:24)</td><td>If jump distance specifier (.length) is ".B", ".W" or ".A", a displacement (dsp) is added to the current Program Counter (PC) as signed integers. The lower 32 bits of the result is the effective address to be operated on.
A dsp is automatically calculated by the assembler.

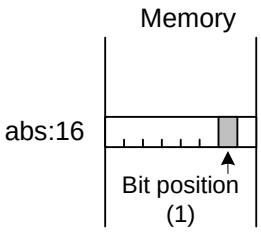
This addressing mode is available in JCnd, JMP and JSR instructions.</td><td> 

In the case when ".B": -128 ≤ dsp ≤ +127
In the case when ".W": -32768 ≤ dsp ≤ +32767
In the case when ".A": -8388608 ≤ dsp ≤ +8388607
Current PC value = JMP instruction address + 1</td></tr>
</table>

2.7 Bit Instruction Addressing

This addressing is available in the following instructions:
BCLR, BSET, BNOT, BTST, BM*Cnd*, BTSTC, and BTSTS

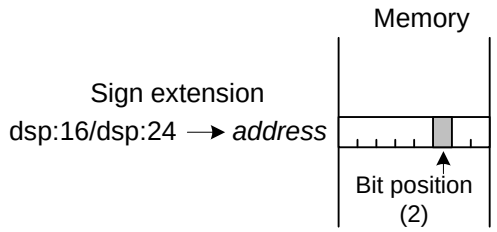
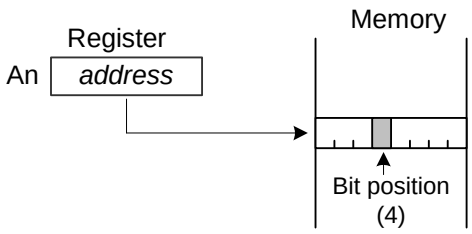
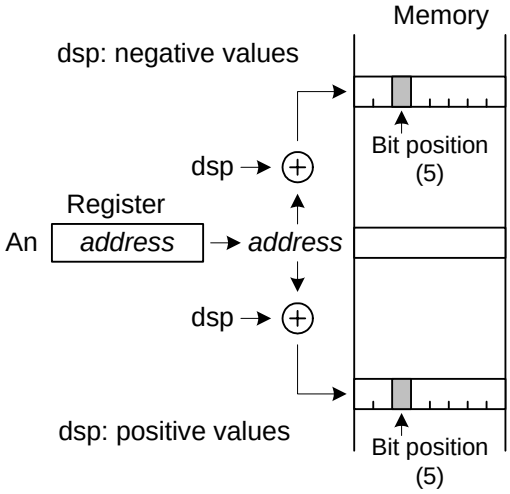
<table>
<tr><td colspan="2">Register direct</td><td rowspan="2"> 
</td></tr>
<tr><td>bit,R0H
bit,R0L
bit,R1H
bit,R1L
bit,R2H
bit,R2L
bit,R3H
bit,R3L</td><td>The register bit specified by the bit position (bit) is the object to be operated on.

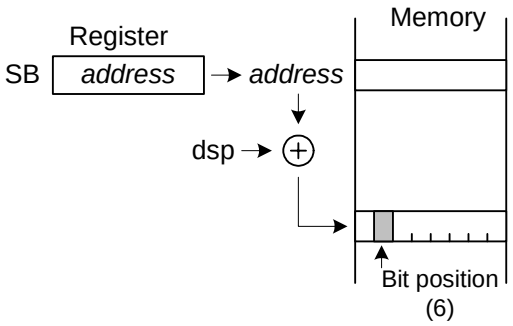
0 to 7 can be specified for the bit position (bit) to be operated on.</td></tr>
<tr><td colspan="2">Absolute</td><td rowspan="2"> 
</td></tr>
<tr><td>bit, abs:16</td><td>The bit specified by the bit position (bit) of the register indicated by abs is the object to be operated on.

Effective address range:
00000000h to 0000FFFFh

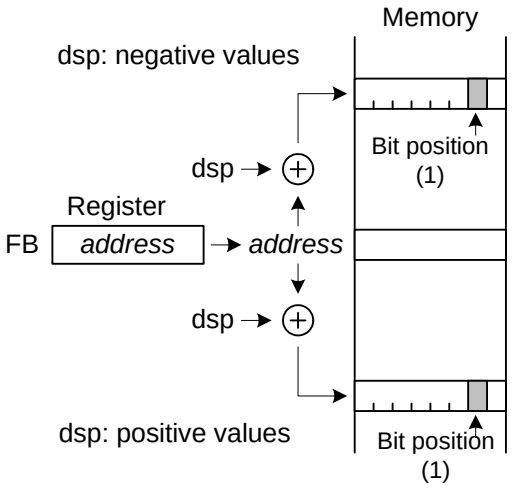
0 to 7 can be specified for the bit position (bit) to be operated on.

This addressing mode is only available in the BTST instruction.</td></tr>
</table>

Sign-extended absolute		
bit,dsp:16 bit,dsp:24	The bit position (bit) indicated by a sign-extended displacement (dsp) is the object to be operated on. Effective address range (dsp: 16): 00000000h to 00007FFFh FFFF8000h to FFFFFFFFh Effective address range (dsp: 24): 00000000h to 007FFFFFh FF800000h to FFFFFFFFh 0 to 7 can be specified for the bit position (bit).	
Address register indirect		
bit,[A0] bit,[A1] bit,[A2] bit,[A3]	The bit position (bit) indicated by an Address register (A0, A1, A2 or A3) is the object to be operated on. The address range to be specified is 00000000h to FFFFFFFFh 0 to 7 can be specified for the bit position (bit).	
Address register relative		
bit,dsp:8[A0] bit,dsp:8[A1] bit,dsp:8[A2] bit,dsp:8[A3] bit,dsp:16[A0] bit,dsp:16[A1] bit,dsp:16[A2] bit,dsp:16[A3] bit,dsp:24[A0] bit,dsp:24[A1] bit,dsp:24[A2] bit,dsp:24[A3]	A displacement (dsp) is added to an Address register (A0, A1, A2 or A3) as signed integers. The bit position (bit) indicated by the lower 32 bits of the result is the object to be operated on. The address range to be specified is 00000000h to FFFFFFFFh 0 to 7 can be specified for the bit position (bit).	

<table>
<tr><td colspan="2">SB relative</td><td rowspan="2"> 
</td></tr>
<tr><td>bit,dsp:8[SB]
bit,dsp:16[SB]
bit,dsp:24[SB]</td><td>A displacement (dsp) is added to the Static base register (SB) as unsigned integers. The bit position (bit) indicated by the lower 32 bits of the result is the object to be operated on.

The address range to be specified is 00000000h to FFFFFFFFh

0 to 7 can be specified for the bit position (bit).</td></tr>
<tr><td colspan="2">FB relative</td><td rowspan="2"> 
</td></tr>
<tr><td>bit,dsp:8[FB]
bit,dsp:16[FB]</td><td>A displacement (dsp) is added to the Frame base register (FB) as signed integers. The bit position (bit) indicated by the lower 32 bits of the result is the object to be operated on.

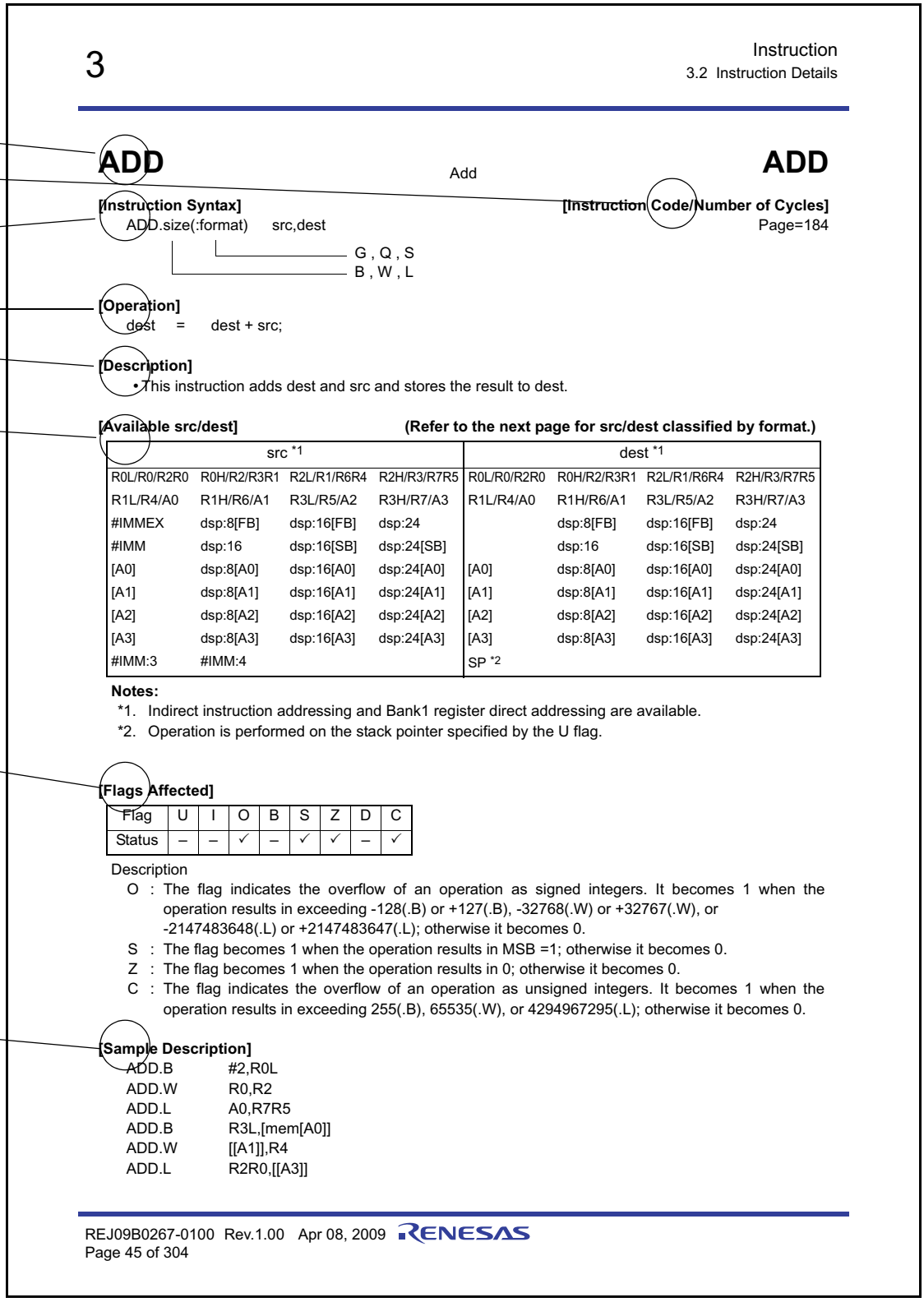
The address range to be specified is 00000000h to FFFFFFFFh

0 to 7 can be specified for the bit position.</td></tr>
</table>

3. Instruction

3.1 Guide to This Chapter

This chapter describes the functionality of each instruction by showing their syntax, operation, function, available src/dest, flags, and example instructions.
The following sample shows how to read this chapter.



3
Instruction
3.2 Instruction Details

(1) **ADD** Add **ADD**

(2) **[Instruction Code/Number of Cycles]** Page=184

(3) **[Instruction Syntax]**
ADD.size(:format) src,dest
G , Q , S
B , W , L

(4) **[Operation]**
dest = dest + src;

(5) **[Description]**
• This instruction adds dest and src and stores the result to dest.

(6) **[Available src/dest]** **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:3	#IMM:4			SP *2			

Notes:
*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. Operation is performed on the stack pointer specified by the U flag.

(7) **[Flags Affected]**

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description
O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB =1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W), or 4294967295(.L); otherwise it becomes 0.

(8) **[Sample Description]**

```
ADD.B    #2,R0L
ADD.W    R0,R2
ADD.L    A0,R7R5
ADD.B    R3L,[mem[A0]]
ADD.W    [[A1]],R4
ADD.L    R2R0,[[A3]]
```

REJ09B0267-0100 Rev.1.00 Apr 08, 2009
Page 45 of 304

(1) Instruction Mnemonic

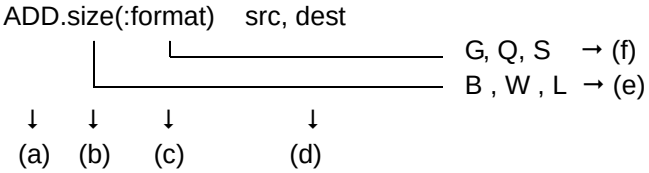
Indicates the mnemonic of the instruction explained.

(2) Instruction code/number of cycles

Indicates the page on which the instruction code/number of cycles are listed.

(3) Instruction Syntax

Indicates the syntax that specifies the operation to be performed and the operand(s) used.



(a) Mnemonic ADD

Specifies the instruction.

(b) Size specifier .size

Specifies the data size of the operand(s).
The size specifiers used in the instruction syntax are listed as below:

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

Some instructions have no size specifier.

(c) Instruction format specifier :format

Specifies the format of the instruction. If omitted, the format is automatically chosen by the assembler.
The instruction format specifiers are listed below:

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

Some instructions have no instruction format specifier.

(d) Operand src, dest

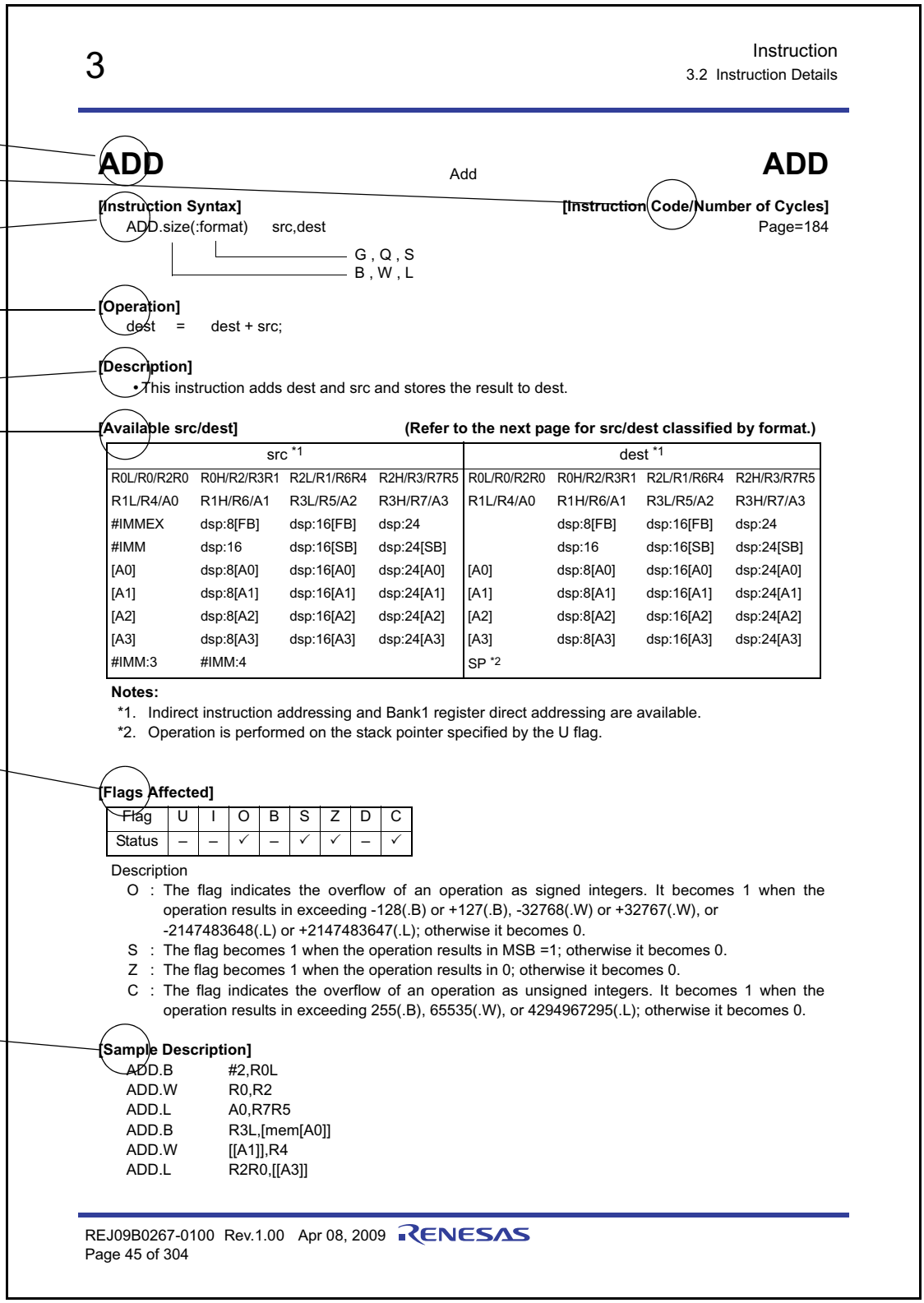
Specifies the operand.

(e) Available size specifiers

Indicates the available data sizes to be specified in (b).

(f) Available instruction format specifiers

Indicates the available instruction formats in (c).



3
Instruction
3.2 Instruction Details

(1) **ADD** Add **ADD**

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

(3) ADD.size(:format) src,dest Page=184

G , Q , S
B , W , L

(4) **[Operation]**

dest = dest + src;

(5) **[Description]**

• This instruction adds dest and src and stores the result to dest.

(6) **[Available src/dest]** **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:3	#IMM:4			SP *2			

Notes:
*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. Operation is performed on the stack pointer specified by the U flag.

(7) **[Flags Affected]**

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description
O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB =1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W), or 4294967295(.L); otherwise it becomes 0.

(8) **[Sample Description]**

```
ADD.B    #2,R0L
ADD.W    R0,R2
ADD.L    A0,R7R5
ADD.B    R3L,[mem[A0]]
ADD.W    [[A1]],R4
ADD.L    R2R0,[[A3]]
```

REJ09B0267-0100 Rev.1.00 Apr 08, 2009
Page 45 of 304

(4) Operation

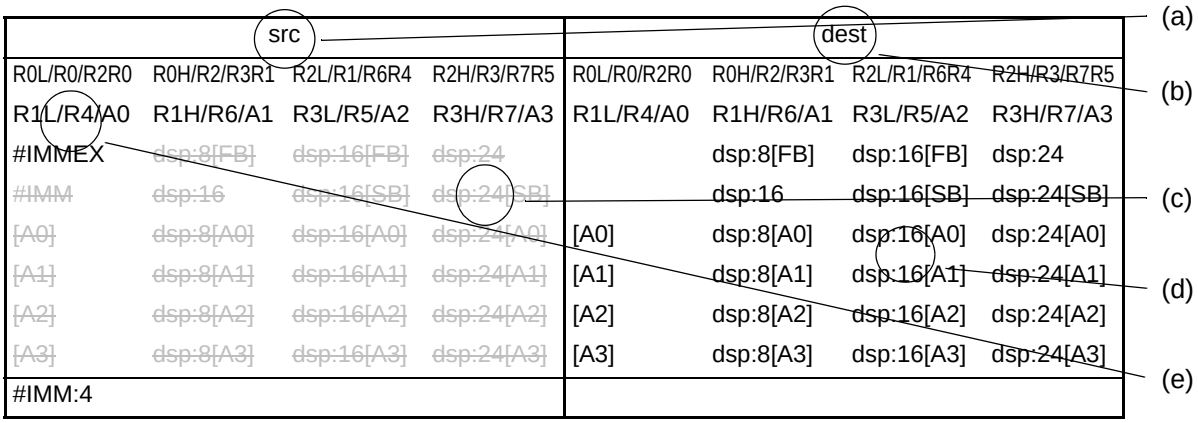
Explains the operation in C language style.

(5) Description

Describes the operation.

(6) Addressing modes for src/dest (label)

Indicates the addressing modes for the operand(s).



src				dest			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:4							

(a) Addressing modes for src (source)

(b) Addressing modes for dest (destination)

(c) Unavailable addressing modes

(d) Available addressing modes

(e) Addressing modes that changes according to operation size

Left (R1L) is available when the operation size is a byte (8 bits).
Center (R4) is available when the operation size is a word (16 bits).
Right (A0) is available when the operation size is a long word (32 bits).

(7) Flags Affected

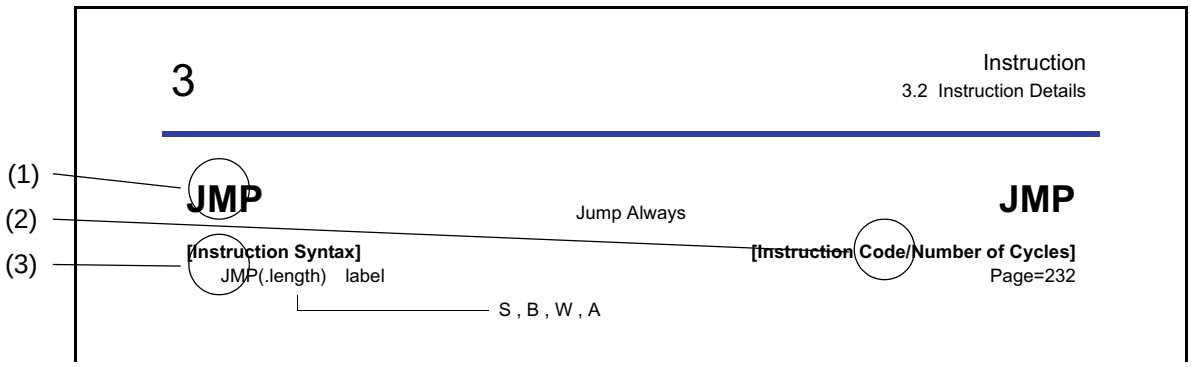
Indicates which flags are affected by this instruction. The symbols in the table have the following meanings:

"–" Not affected by this instruction
"✓" Affected by this instruction

(8) Sample description

Indicates sample descriptions of the instruction.

The following sample shows the syntax of each jump instruction: JMP, JPMI, JSR, and JSRI.



(3) Instruction Syntax

Indicates the syntax that specifies the operation to be performed and the operands used.

JMP(.length) label
S , B , W , A →(d)
↓ ↓ ↓
(a) (b) (c)

(a) Mnemonic JMP

Specifies the instruction.

(b) Jump distance specifier .length

Specifies the jump distance. In instructions JMP and JSR, the jump distance specifier can be omitted. The jump distance specifiers used in the instruction syntax are listed as below:

.S	PC-relative, 3-bit displacement, forward only
.B	PC-relative, 8-bit displacement
.W	PC-relative, 16-bit displacement
.A	PC-relative, 24-bit displacement
.L	Absolute, 32-bit address

(c) Operand label

Specifies the operand for label.

(d) Available jump distance specifiers

Indicates the available jump distances in (b).

3.2 Instruction Details

Detailed explanations of each instruction for the R32C/100 Series MCU begin on the next page.

ABS

Absolute

ABS

[Instruction Syntax]

ABS.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=182

[Operation]

```
if (dest < 0)
        dest =    -dest;
```

[Description]

- This instruction takes an absolute value of dest and stores it to dest.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag becomes 1 when dest before the operation is -128(.B), -32768(.W), or -2147483648(.L); otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag is undefined.

[Sample Description]

```
ABS.B     R0L
ABS.W     R2
ABS.L     R7R5
ABS.L     A0
ABS.W     mem[SB]
```

ADC

Add with Carry

ADC

[Instruction Syntax]

ADC.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=182

[Operation]

dest = dest + src + C;

[Description]

• This instruction adds dest, src and the C flag and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W), or 4294967295(.L); otherwise it becomes 0.

[Sample Description]

```
ADC.B   #2,R0L
ADC.W   R0,R2
ADC.L   A0,R7R5
ADC.B   R3L,[A0]
ADC.W   [A1],R4
ADC.L   R2R0,[A3]
```

ADCF

Add Carry Flag

ADCF

[Instruction Syntax]

ADCF.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=183

[Operation]

dest = dest + C;

[Description]

- This instruction adds dest and the C flag and stores the result to dest.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : This flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : This flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W) or 4294967295(.L); otherwise it becomes 0.

[Sample Description]

```
ADCF.B    R0L
ADCF.W    R2
ADCF.L    [A3]
ADCF.W    [mem[A0]]
```

ADD

Add

ADD

[Instruction Syntax]

ADD.size(:format) src,dest

G , Q , S
B , W , L

[Instruction Code/Number of Cycles]

Page=184

[Operation]

dest = dest + src;

[Description]

• This instruction adds dest and src and stores the result to dest.

[Available src/dest] **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:3	#IMM:4			SP *2			

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

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

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB =1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W), or 4294967295(.L); otherwise it becomes 0.

[Sample Description]

```
ADD.B     #2,R0L
ADD.W     R0,R2
ADD.L     A0,R7R5
ADD.B     R3L,[mem[A0]]
ADD.W     [[A1]],R4
ADD.L     R2R0,[[A3]]
```

[src/dest Classified by Format]

G format

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMMEX:8 *3	#IMMEX:16*3	#IMM:32 *3		SP *2,*3			

Notes:
*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. Operation is performed on the stack pointer specified by the U flag.
*3. Only ".L" can be specified as a size specifier (.size).

Q format

src	dest *1			
#IMM:4 *2	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
		dsp:8[FB]	dsp:16[FB]	dsp:24
		dsp:16	dsp:16[SB]	dsp:24[SB]
	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:3 *3	SP *4,*5			

Notes:
*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. The effective value is -8 ≤ #IMM:4 ≤ +7
*3. The effective value is #IMM:3=4,8,12,16 or 20.
*4. Operation is performed on the stack pointer specified by the U flag.
*5. Only ".L" can be specified as size specifier (.size).

S format

src	dest *1			
#IMM	R0L/R0/R2R0	dsp:16	dsp:8[SB]	dsp:8[FB]

Note:
*1. Indirect instruction addressing and Bank1 register direct addressing are available.

ADDF

Add Floating-point

ADDF

[Instruction Syntax]

ADDF src,dest

[Instruction Code/Number of Cycles]

Page=187

[Operation]

dest = dest + src;

[Description]

- This instruction executes floating-point addition of src and dest and stores the result to dest. Both operands and the result are interpreted as signed integers.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG).
- When the result is overflowed, it takes either the maximum positive normal number (7F7FFFFFh) or the maximum negative normal number (FF7FFFFFh) depending on whether signed or not.
- When the result is underflowed, it takes any of the following according to the rounding mode: zero (00000000h), the minimum positive number (00800000h) or the minimum negative normal number (80800000h).
- The result for invalid numbers is undefined.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

FU : The flag becomes 1 when the operand has invalid numbers or when the operation results in an underflow; otherwise it becomes 0.

O : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag is undefined.

[Sample Description]

```
ADDF    R2R0,R3R1
ADDF    [A1],R2R0
ADDF    mem[FB],R3R1
```

ADSF

Add Sign Flag

ADSF

[Instruction Syntax]

ADSF.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=188

[Operation]

dest = dest + S;

[Description]

- This instruction adds dest and the S flag and stores the result to dest.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W) or, -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag indicates the overflow of an operation as unsigned integers. It becomes 1 when the operation results in exceeding 255(.B), 65535(.W), or 4294967295(.L); otherwise it becomes 0.

[Sample Description]

```
ADSF.B    R0L
ADSF.W    R2
ADSF.L    [A3]
ADSF.W    mem[A0]
```

AND

And Logical

AND

[Instruction Syntax]

AND.size(:format) src,dest

G , S
B , W , L

[Instruction Code/Number of Cycles]

Page=189

[Operation]

dest = dest & src;

[Description]

• This instruction logically ANDs dest and src and stores the result to dest.

[Available src/dest] **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
				dsp:8[SB]			

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
AND.B     #2,R0L
AND.W     R0,R2
AND.L     A0,R7R5
AND.B     R3L,[mem[A0]]
AND.W     [[A1]],R4
AND.L     R2R0,[[A3]]
```

[src/dest Classified by Format]

G format

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

S format

src	dest *1			
#IMM	R0L/R0/R2R0	dsp:16	dsp:8[SB]	dsp:8[FB]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

BCLR

Clear a Bit

BCLR

[Instruction Syntax]

BCLR dest

[Instruction Code/Number of Cycles]

Page=191

[Operation]

dest = 0;

[Description]

- This instruction sets dest to 0.

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
BCLR    1,R0L
BCLR    4,R2H
BCLR    2,[A3]
BCLR    7,mem[SB]
```

BITINDEX

Index next Bit Instruction

BITINDEX

[Instruction Syntax]

BITINDEX.size src

B , W , L

[Instruction Code/Number of Cycles]

Page=192

[Operation]

[Description]

- This instruction modifies addressing of the next bit instruction.
- No interrupt request is accepted until the next instruction execution is completed.
- The specified src is the src or the dest index value (bit) of the next sequential instruction.

[Available src]

src *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
BITINDEX.B    R0L
BITINDEX.W    R2
BITINDEX.L    [A0]
BITINDEX.W    mem[A1]
```

BM*Cnd* Move a Bit Conditionally BM*Cnd*

[Instruction Syntax]

BM*Cnd* dest

[Instruction Code/Number of Cycles]

Page=192

[Operation]

```
if (true)
        dest =    1;
else
        dest =    0;
```

[Description]

- This instruction moves the truth value (1 if true, 0 if false) of the condition specified by *Cnd* to bit in dest.
- The following table lists Cnd types:

Cnd	Conditions		Expression	*Cnd*	Conditions		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 to / Z flag is 0.	≠
GTU	C & ~Z == 1	Greater than	<	LEU	C & ~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 as signed integer	≤	LE	(S ^ O) | Z == 1	Equal to or smaller than as signed integer	≥
GT	(S ^ O) | Z == 0	Greater than as signed integer	<	LT	S ^ O == 1	Smaller than as signed integer	>
O	O == 1	O flag is 1.		NO	O == 0	O flag is 0.	

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
BMC     1,R0L
BMLT    4,R2H
BMN     2,[A3]
BMNO    7,mem[SB]
```

BNOT

Not a Bit

BNOT

[Instruction Syntax]

BNOT dest

[Instruction Code/Number of Cycles]

Page=193

[Operation]

dest = ~dest;

[Description]

- This instruction inverts bit in dest.

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
BNOT    1,R0L
BNOT    4,R2H
BNOT    2,[A3]
BNOT    7,mem[SB]
```

BRK

Break

BRK

[Instruction Syntax]

BRK

[Instruction Code/Number of Cycles]

Page=193

[Operation]

When the address FFFFFFE7h is not FFh,

```
SP    =    SP - 4;
*SP   =    FLG;
SP    =    SP - 4;
*SP   =    PC + 1;
PC    =    *(FFFFFFE4h);
```

When the address FFFFFFE7h is FFh,

```
SP    =    SP - 4;
*SP   =    FLG;
SP    =    SP - 4;
*SP   =    PC + 1;
PC    =    *IntBase;
```

[Description]

- This instruction generates a BRK interrupt.
- The BRK interrupt is non-maskable.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	–	–	✓	–

Description

U : The flag becomes 0.
I : The flag becomes 0.
D : The flag becomes 0.

The flags are saved to the stack area before the BRK instruction is executed.

[Sample Description]

```
BRK
```

BRK2

Break 2

BRK2

[Instruction Syntax]

BRK2

[Instruction Code/Number of Cycles]

Page=193

[Operation]

```
SP    =    SP - 4;
*SP   =    FLG;
SP    =    SP - 4;
*SP   =    PC + 1;
PC    =    *(long *)0x0004C000;
```

[Description]

- This instruction is provided only for use in debuggers. Do not use it in user programs.
- This instruction generates a BRK2 interrupt.
- The BRK2 interrupt is non-maskable.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	–	–	✓	–

Description

U : The flag becomes 0.
I : The flag becomes 0.
D : The flag becomes 0.

The flags are saved to the stack area before the BRK2 instruction is executed.

[Sample Description]

BRK2

BSET

Set a Bit

BSET

[Instruction Syntax]

BSET dest

[Instruction Code/Number of Cycles]

Page=194

[Operation]

dest = 1;

[Description]

- This instruction sets bit in dest to 1.

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
BSET    1,R0L
BSET    4,R2H
BSET    2,[A3]
BSET    7,mem[SB]
```

BTST

Test a Bit

BTST

[Instruction Syntax]

BTST (:format) src

G , S

[Instruction Code/Number of Cycles]

Page=194

[Operation]

```
Z  =  ~src;
C  =  src;
```

[Description]

• This instruction moves inverted bit in src to the Z flag and non-inverted bit in src to the C flag.

[Available src]

G format

src *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

S format

src
bit,abs:16

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	–	✓	–	✓

Description

Z : The flag becomes 1 when src is 0; otherwise it becomes 0.
C : The flag becomes 1 when src is 1; otherwise it becomes 0.

[Sample Description]

```
BTST    1,R0L
BTST    4,R2H
BTST    2,[A3]
BTST    7,mem
```

BTSTC

Test a Bit and Clear

BTSTC

[Instruction Syntax]

BTSTC dest

[Instruction Code/Number of Cycles]

Page=195

[Operation]

```
Z    =  ~dest;
C    =  dest;
dest =  0;
```

[Description]

- This instruction moves inverted bit in dest to the Z flag and non-inverted bit in dest to the C flag, then set bit in dest to 0.
- Do not use this instruction for dest in SFR area.

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	–	✓	–	✓

Description

Z : The flag becomes 1 when dest is 0; otherwise it becomes 0.
C : The flag becomes 1 when dest is 1; otherwise it becomes 0.

[Sample Description]

```
BTSTC   1,R0L
BTSTC   4,R2H
BTSTC   2,[A3]
BTSTC   7,mem[SB]
```

BTSTS

Test a Bit and Set

BTSTS

[Instruction Syntax]

BTSTS dest

[Instruction Code/Number of Cycles]

Page=196

[Operation]

```
Z    =   ~dest;
C    =   dest;
dest =   1;
```

[Description]

- This instruction moves inverted bit in dest to the Z flag and non-inverted bit in dest to the C flag, then set bit in dest to 1.
- Do not use this instruction for dest in SFR area.

[Available dest]

dest *1			
bit,R0L	bit,R0H	bit,R2L	bit,R2H
bit,R1L	bit,R1H	bit,R3L	bit,R3H
	bit,dsp:8[FB]	bit,dsp:16[FB]	bit,dsp:24
	bit,dsp:16	bit,dsp:16[SB]	bit,dsp:24[SB]
bit,[A0]	bit,dsp:8[A0]	bit,dsp:16[A0]	bit,dsp:24[A0]
bit,[A1]	bit,dsp:8[A1]	bit,dsp:16[A1]	bit,dsp:24[A1]
bit,[A2]	bit,dsp:8[A2]	bit,dsp:16[A2]	bit,dsp:24[A2]
bit,[A3]	bit,dsp:8[A3]	bit,dsp:16[A3]	bit,dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	–	✓	–	✓

Description

Z : The flag becomes 1 when dest is 0; otherwise it becomes 0.
C : The flag becomes 1 when dest is 1; otherwise it becomes 0.

[Sample Description]

```
BTSTS   1,R0L
BTSTS   4,R2H
BTSTS   2,[A3]
BTSTS   7,mem[SB]
```

CLIP

Clip

CLIP

[Instruction Syntax]

CLIP.size src1,src2,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=196

[Operation]

```
if (dest < src1)
        dest =    src1;
else if (dest > src2)
        dest =    src2;
```

[Description]

- This instruction clips the dest value to satisfy the condition src1 ≤ dest ≤ src2.
- The values are compared as signed integers.

[Available src/dest]

src1 src2	dest *1			
#IMM	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
		dsp:8[FB]	dsp:16[FB]	dsp:24
		dsp:16	dsp:16[SB]	dsp:24[SB]
	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
CLIP.B      #-32,#32,R0L
CLIP.W      #1000,#4000,R2
CLIP.L      #-100000,#100000,R7R5
CLIP.W      #0,#10000,mem[A0]
```

CMP

Compare

CMP

[Instruction Syntax]

CMP.size(:format) src,dest

size: G , Q , S
format: B , W , L

[Instruction Code/Number of Cycles]

Page=197

[Operation]

dest - src;

[Description]

- This instruction varies each flag bit of the flag register (FLG) according to the result of subtraction of src from dest.

[Available src/dest] **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:4							

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : This flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when an operation as unsigned integers results in any value equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
CMP.B    #2,R0L
CMP.W    R0,R2
CMP.L    A0,R7R5
CMP.B    R3L,[mem[A0]]
CMP.W    [[A1]],R4
CMP.L    R2R0,[[A3]]
```

[src/dest Classified by Format]

G format

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

Q format

src	dest *1			
#IMM:4 *2	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
		dsp:8[FB]	dsp:16[FB]	dsp:24
		dsp:16	dsp:16[SB]	dsp:24[SB]
	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

*2. The effective value is -8 ≤ #IMM:4 ≤ +7.

S format

src	dest *1			
#IMM	R0L/R0/R2R0	dsp:16	dsp:8[SB]	dsp:8[FB]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

CMPF

Compare Floating-point

CMPF

[Instruction Syntax]

CMPF src,dest

[Instruction Code/Number of Cycles]

Page=199

[Operation]

dest - src;

[Description]

- This instruction varies each flag bit of the flag register (FLG) according to the result of subtraction of src from dest.
- The result for invalid numbers is undefined.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers; otherwise it becomes 0.
FU : The flag becomes 1 when the operand has invalid numbers; otherwise it becomes 0.
O : The flag becomes 1 when the operand has invalid numbers; otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag is undefined.

[Sample Description]

```
CMPF    R2R0,R3R1
CMPF    [A1],R2R0
CMPF    mem[FB],R3R1
```

CNVIF

Convert Integer to Floating-point

CNVIF

[Instruction Syntax]

CNVIF src,dest

[Instruction Code/Number of Cycles]

Page=201

[Operation]

dest = (float) src;

[Description]

- This instruction converts src to single precision floating-point number.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG).

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 0.
FU : The flag becomes 0.
O : The flag becomes 0.
S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag is undefined.

[Sample Description]

```
CNVIF   R2R0,R3R1
CNVIF   [A1],R2R0
CNVIF   mem[FB],R3R1
```

DADC

Add Decimal with Carry

DADC

[Instruction Syntax]

DADC.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=203

[Operation]

dest = dest + src + C;

[Description]

• This instruction executes decimal addition of dest, src and the C flag and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when the operation results in exceeding 99(.B), 9999(.W), or 99999999(.L): otherwise it becomes 0.

[Sample Description]

```
DADC.B    #12,R0L
DADC.W    R0,R2
DADC.L    A0,R7R5
DADC.B    R3L,[A0]
DADC.W    [A1],R4
DADC.L    R2R0,[A3]
```

DADD

Add Decimal

DADD

[Instruction Syntax]

DADD.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=205

[Operation]

dest = dest + src;

[Description]

- This instruction executes decimal addition of dest and src and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when the operation results in exceeding 99(.B), 9999(.W), or 99999999(.L); otherwise it becomes 0.

[Sample Description]

```
DADD.B    #12,R0L
DADD.W    R0,R2
DADD.L    A0,R7R5
DADD.B    R3L,[A0]
DADD.W    [A1],R4
DADD.L    R2R0,[A3]
```

DEC

Decrement

DEC

[Instruction Syntax]

DEC.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=206

[Operation]

dest = dest - 1;

[Description]

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

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
DEC.B   R0L
DEC.W   R2
DEC.L   R7R5
DEC.L   A0
DEC.W   mem[SB]
```

DIV

Signed Divide

DIV

[Instruction Syntax]

DIV.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=207

[Operation]

dest = dest / src;

[Description]

- This instruction divides dest by src and stores the result to dest. Both operand and the result are interpreted as signed integers. The result is rounded toward 0.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when src is 0 or the operation results over -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
DIV.B   #12,R0L
DIV.W   R0,R2
DIV.L   A0,R7R5
DIV.B   R3L,[A0]
DIV.W   [A1],R4
DIV.L   R2R0,[A3]
```

DIVF

Divide Floating-point

DIVF

[Instruction Syntax]

DIVF src,dest

[Instruction Code/Number of Cycles]

Page=209

[Operation]

dest = dest / src;

[Description]

- This instruction divides dest by src and stores the result to dest. Both operands and the result are interpreted as signed integers.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG).
- When the result is overflowed, it takes either the maximum positive normal number (7F7FFFFFh) or the maximum negative normal number (FF7FFFFFh) depending on whether signed or not.
- When the result is underflowed, it takes any of the following according to the rounding mode: zero (00000000h), the minimum positive number (00800000h) or the minimum negative normal number (80800000h).
- The result for invalid numbers is undefined.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

FU : The flag becomes 1 when the operand has invalid numbers or when the operation results in an underflow; otherwise it becomes 0.

O : The flag becomes 1 when src is 0, when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag is undefined.

[Sample Description]

```
DIVF    R2R0,R3R1
DIVF    [A1],R2R0
DIVF    mem[FB],R3R1
```

DIVU

Unsigned Divide

DIVU

[Instruction Syntax]

DIVU.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=211

[Operation]

dest = dest / src;

[Description]

- This instruction divides dest by src and stores the result to dest. Both operands and the result are interpreted as unsigned integers. The result is rounded toward 0.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when src is 0; otherwise it becomes 0.

[Sample Description]

```
DIVU.B    #12,R0L
DIVU.W    R0,R2
DIVU.L    A0,R7R5
DIVU.B    R3L,[A0]
DIVU.W    [A1],R4
DIVU.L    R2R0,[A3]
```

DIVX

Signed Divide extra

DIVX

[Instruction Syntax]

DIVX.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=213

[Operation]

dest = dest / src;

[Description]

- This instruction divides dest by src and stores the result to dest. Both operands and the result are interpreted as signed integers. The result is rounded toward -infinity.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when src is 0 or the operation results over -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
DIVX.B    #12,R0L
DIVX.W    R0,R2
DIVX.L    A0,R7R5
DIVX.B    R3L,[A0]
DIVX.W    [A1],R4
DIVX.L    R2R0,[A3]
```

DSBB

Subtract Decimal with Borrow

DSBB

[Instruction Syntax]

DSBB.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=215

[Operation]

dest = dest - src - ~C;

[Description]

- This instruction executes decimal subtraction of src and the inverted C flag from dest and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the operation results in any value equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
DSBB.B    #12,R0L
DSBB.W    R0,R2
DSBB.L    A0,R7R5
DSBB.B    R3L,mem[A0]
DSBB.W    [A1],R4
DSBB.L    R2R0,[A3]
```

DSUB

Subtract Decimal

DSUB

[Instruction Syntax]

DSUB.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=217

[Operation]

dest = dest - src;

[Description]

- This instruction executes decimal subtraction of src from dest and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when the operation results in any value equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
DSUB.B    #12,R0L
DSUB.W    R0,R2
DSUB.L    A0,R7R5
DSUB.B    R3L,[A0]
DSUB.W    mem[A1],R4
DSUB.L    R2R0,[A3]
```

EDIV

Extended Signed Divide with Remainder

EDIV

[Instruction Syntax]

EDIV.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=219

[Operation]

destL(quotient) = dest / src;
destH(remainder)= dest % src;

[Description]

- This instruction divides dest by src and stores the quotient of the result to lower half of the dest and the remainder to higher half. Both operands and the result are interpreted as signed integers. The quotient is rounded toward 0. The remainder has the same sign as that of the dividend (dest).
- When size specifier (.size) is ".B", src is operated in 8 bits and dest is in 16 bits. The quotient and remainder are respectively stored in 8 bits. Four types of addressing can be specified for dest: R0(R0H:R0L), R1(R1H:R1L), R2(R2H:R2L), or R3(R3H:R3L).
- When size specifier (.size) is ".W", src is operated in 16 bits and dest is in 32 bits. The quotient and remainder are respectively stored in 16 bits. 4 addressing modes can be specified for dest: R2R0(R2:R0), R3R1(R3:R1), R6R4(R6:R4), or R7R5(R7:R5).
- When size specifier (.size) is ".L", src is operated in 32 bits and dest is in 64 bits. The quotient and remainder are respectively stored in 32 bits. 4 addressing modes can be specified for dest: R3R1R2R0(R3R1:R2R0), R7R5R6R4(R7R5:R6R4), A1A0(A1:A0), or A3A2(A3:A2).

[Available src/dest]

src *1				dest *1	
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0/R2R0/R3R1R2R0	R2/R3R1/R7R5R6R4
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1/R6R4/A1A0	R3/R7R5/A3A2
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]		
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]		
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]		
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]		

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when src is 0 or the operation results over -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
EDIV.B    #12,R0          ; R0H:R0L ÷ 12
EDIV.W    R0,R3R1         ; R3:R1 ÷ R0
EDIV.L    A0,R7R5R6R4     ; R7R5:R6R4 ÷ A0
EDIV.W    [A1],R2R0       ; R2:R0 ÷ [A1]
```

EDIVU

Extended Unsigned Divide with Remainder

EDIVU

[Instruction Syntax]

EDIVU.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=220

[Operation]

destL(quotient) = dest / src;
destH(remainder)= dest % src;

[Description]

- This instruction divides dest by src and stores the quotient of the result to lower bits of the dest and the remainder to higher bits. Both operands and the result are interpreted as unsigned integers.
- When size specifier (.size) is ".B", src is operated in 8 bits and dest is in 16 bits. The quotient and remainder are respectively stored in 8 bits. 4 addressing modes can be specified for dest: R0(R0H:R0L), R1(R1H:R1L), R2(R2H:R2L), or R3(R3H:R3L).
- When size specifier (.size) is ".W", src is operated in 16 bits and dest is in 32 bits. The quotient and remainder are respectively stored in 16 bits. 4 addressing modes can be specified for dest: R2R0(R2:R0), R3R1(R3:R1), R6R4(R6:R4), or R7R5(R7:R5).
- When size specifier (.size) is ".L", src is operated in 32 bits and dest is in 64 bits. The quotient and remainder are respectively stored in 32 bits. 4 addressing modes can be specified for dest: R3R1R2R0(R3R1:R2R0), R7R5R6R4(R7R5:R6R4), A1A0(A1:A0), or A3A2(A3:A2).

[Available src/dest]

src *1				dest *1	
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0/R2R0/R3R1R2R0	R2/R3R1/R7R5R6R4
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1/R6R4/A1A0	R3/R7R5/A3A2
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]		
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]		
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]		
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]		

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 either when the quotient results in over 8 bits (.B), 16 bits (.W), or 32 bits (.L), or the divisor (src) is 0: otherwise it becomes 0.

[Sample Description]

```
EDIVU.B     #12,R0          ; R0H:R0L ÷ 12
EDIVU.W     R0,R3R1         ; R3:R1 ÷ R0
EDIVU.L     A0,R7R5R6R4     ; R7R5:R6R4 ÷ A0
EDIVU.W     [A1],R2R0       ; R2:R0 ÷ [A1]
```

EDIVX

Extended Signed Divide extra with Remainder

EDIVX

[Instruction Syntax]

EDIVX.size src,dest

size: B , W , L

[Instruction Code/Number of Cycles]

Page=221

[Operation]

destL(quotient) = dest / src;
destH(remainder)= dest % src;

[Description]

- This instruction divides dest by src and stores the quotient of the result to lower bits of the dest and the remainder to higher bits. Both operands and the result are interpreted as signed integers. The quotient is rounded toward -infinity. The remainder has the same sign as that of the divisor (src).
- When size specifier (.size) is ".B", src is operated in 8 bits and dest is in 16 bits. The quotient and remainder are respectively stored in 8 bits. 4 addressing modes can be specified for dest: R0(R0H:R0L), R1(R1H:R1L), R2(R2H:R2L), or R3(R3H:R3L).
- When size specifier (.size) is ".W", src is operated in 16 bits and dest is in 32 bits. The quotient and remainder are respectively stored in 16 bits. 4 addressing modes can be specified for dest: R2R0(R2:R0), R3R1(R3:R1), R6R4(R6:R4), or R7R5(R7:R5).
- When size specifier (.size) is ".L", src is operated in 32 bits and dest is in 64 bits. The quotient and remainder are respectively stored in 32 bits. 4 addressing modes can be specified for dest: R3R1R2R0(R3R1:R2R0), R7R5R6R4(R7R5:R6R4), A1A0(A1:A0), or A3A2(A3:A2).

[Available src/dest]

src *1				dest *1	
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0/R2R0/R3R1R2R0	R2/R3R1/R7R5R6R4
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1/R6R4/A1A0	R3/R7R5/A3A2
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]		
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]		
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]		
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]		

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when src is 0 or the operation results over -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
EDIVX.B     #12,R0          ; R0H:R0L ÷ 12
EDIVX.W     R0,R3R1         ; R3:R1 ÷ R0
EDIVX.L     A0,R7R5R6R4     ; R7R5:R6R4 ÷ A0
EDIVX.W     [A1],R2R0       ; R2:R0 ÷ [A1]
```

EMUL

Extended Signed Multiply

EMUL

[Instruction Syntax]

EMUL.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=222

[Operation]

destH:dest = dest * src;

[Description]

- This instruction multiplies src and dest and stores the result to destH:dest. Both operands and the result are interpreted as signed integers.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 16 bits. 4 addressing modes can be specified as dest: R0L(R0H:R0L), R1L(R1H:R1L), R2L(R2H:R2L), or R3L(R3H:R3L).
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 32 bits. 4 addressing modes can be specified as dest: R0(R2:R0), R1(R3:R1), R4(R6:R4), or R5(R7:R5).
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 64 bits. 4 addressing modes can be specified as dest: R2R0(R3R1:R2R0), R6R4(R7R5:R6R4), A0(A1:A0), or A2(A3:A2).

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	~~R0H/R2/R3R1~~	R2L/R1/R6R4	~~R2H/R3/R7R5~~
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	~~R1H/R6/A1~~	R3L/R5/A2	~~R3H/R7/A3~~
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24				
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]				
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]				
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]				
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]				
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]				

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
EMUL.B    #12,R0L      ; R0L × 12→R0H:R0L (=R0)
EMUL.W    R0,R1        ; R1 × R0→R3:R1 (=R3R1)
EMUL.L    A0,R6R4      ; R6R4 × A0→R7R5:R6R4
EMUL.W    [A1],R0      ; R0 × [A1]→R2:R0 (=R2R0)
```

EMULU

Extended Unsigned Multiply

EMULU

[Instruction Syntax]

EMULU.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=223

[Operation]

dest = dest * src;

[Description]

- This instruction multiplies src and dest and stores the result to dest. Both operands and the result are interpreted as unsigned integers.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 16 bits. 4 types of data can be specified as dest: R0(R0H:R0L), R1(R1H:R1L), R2(R2H:R2L), or R3(R3H:R3L).
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 32 bits. 4 types of data can be specified as dest: R2R0(R2:R0), R3R1(R3:R1), R6R4(R6:R4), or R7R5(R7:R5).
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 64 bits. 4 types of data can be specified as dest: R3R1R2R0(R3R1:R2R0), R7R5R6R4(R7R5:R6R4), A1A0(A1:A0), or A3A2(A3:A2).

[Available dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	~~R0H/R2/R3R1~~	R2L/R1/R6R4	~~R2H/R3/R7R5~~
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	~~R1H/R6/A1~~	R3L/R5/A2	~~R3H/R7/A3~~
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24				
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]				
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]				
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]				
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]				
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]				

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
EMULU.B   #12,R0L      ; R0L × 12→R0H:R0L (=R0)
EMULU.W   R0,R1        ; R1 × R0→R3:R1 (=R3R1)
EMULU.L   A0,R6R4      ; R6R4 × A0→R7R5:R6R4
EMULU.W   [A1],R0      ; R0 × [A1]→R2:R0 (=R2R0)
```

ENTER

Enter and Create Stack Frame

ENTER

[Instruction Syntax]

ENTER src

[Instruction Code/Number of Cycles]

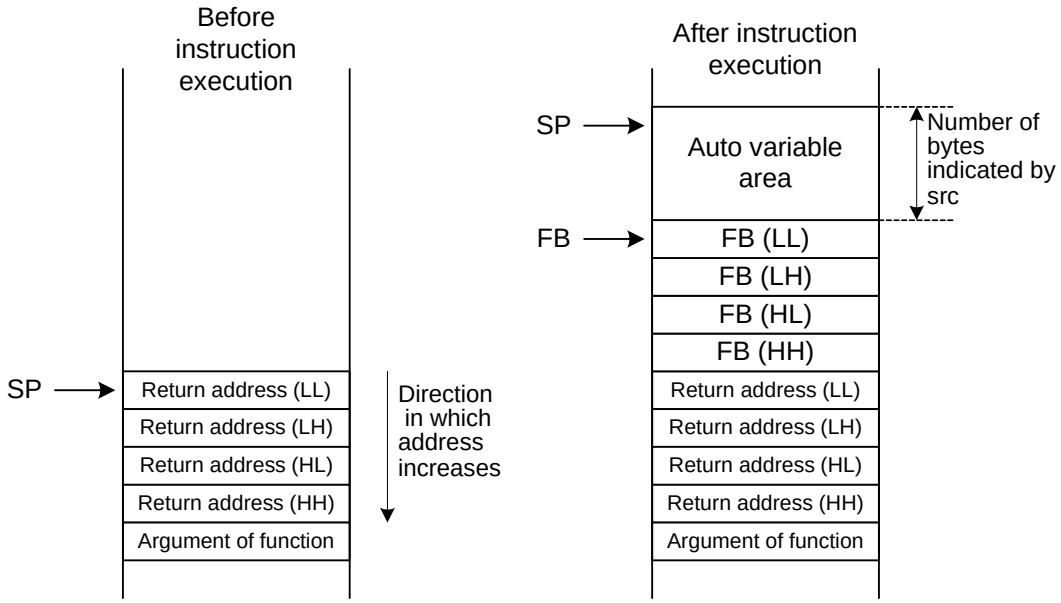
Page=223

[Operation]

```
SP   =   SP - 4;
*SP  =   FB;
FB   =   SP;
SP   =   SP - src;
```

[Description]

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



[Available src]

src	
#IMM:8 *1	#IMM:16 *1

Note:

*1. Set a number in multiples of 4 to #IMM.

[Flags Affected]

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

[Sample Description]

```
ENTER   #12
```

EXITD

Exit and Deallocate Stack Frame

EXITD

[Instruction Syntax]

EXITD

[Instruction Code/Number of Cycles]

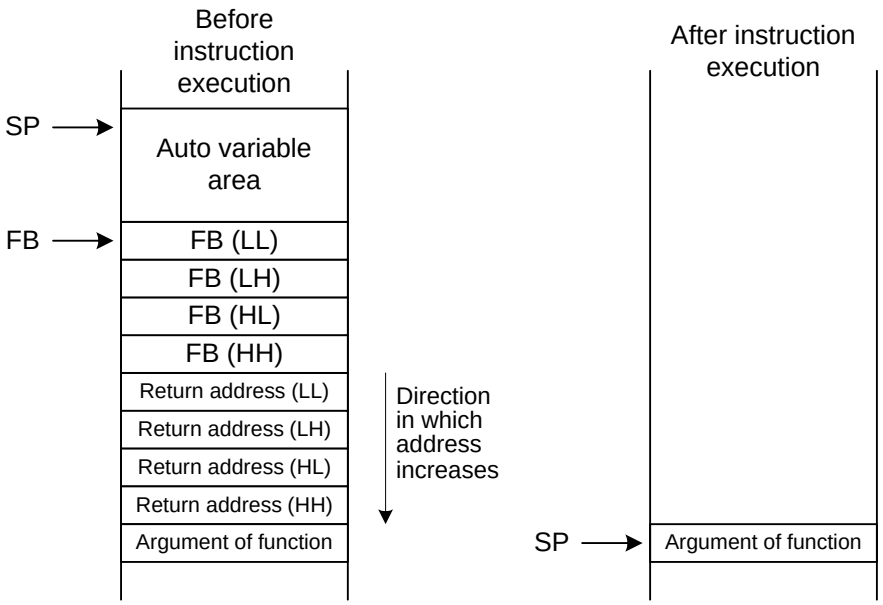
Page=224

[Operation]

```
SP  =  FB;
FB  =  *SP;
SP  =  SP + 4;
PC  =  *SP;
SP  =  SP + 4;
```

[Description]

- 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.



[Flags Affected]

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

[Sample Description]

EXITD

EXITI

Exit Interrupt and Deallocate Stack Frame

EXITI

[Instruction Syntax]

EXITI

[Instruction Code/Number of Cycles]

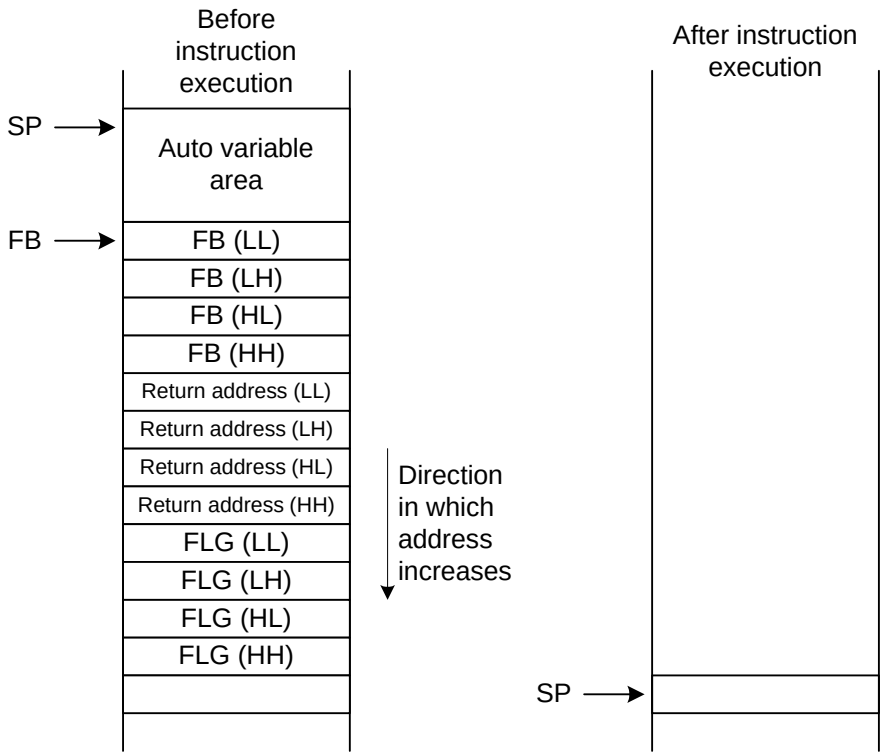
Page=224

[Operation]

```
SP  =  FB;
FB  =  *SP;
SP  =  SP + 4;
PC  =  *SP;
SP  =  SP + 4;
FLG =  *SP;
SP  =  SP + 4;
```

[Description]

- This instruction deallocates the stack frame and exits from the interrupt handler.
- Use this instruction in combination with the ENTER instruction.
- The diagrams below show the stack area status before and after the EXITI instruction is executed at the end of an interrupt handler.



[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	✓	✓	✓	✓	✓	✓	✓	✓

Note:

*1. Every flag has the value before the interrupt was generated, that is, the value saved in the stack area.

[Sample Description]

EXITI

EXTS

Sign Extend

EXTS

[Instruction Syntax]

EXTS.size src,dest

BW , BL , WL

[Instruction Code/Number of Cycles]

Page=224

[Operation]

dest = (short | long) src;

[Description]

- This instruction sign-extends src and stores the result to dest.
- When size specifier (.size) is ".BW", 8-bit src is sign-extended to 16 bits and stored to dest.
- When size specifier (.size) is ".BL", 8-bit src is sign-extended to 32 bits and stored to dest.
- When size specifier (.size) is ".WL", 16-bit src is sign-extended to 32 bits and stored to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/~~R2R0~~	R0H/R2/~~R3R1~~	R2L/R1/~~R6R4~~	R2H/R3/~~R7R5~~	~~R0L~~/R0/R2R0	~~R0H~~/R2/R3R1	~~R2L~~/R1/R6R4	~~R2H~~/R3/R7R5
R1L/R4/~~A0~~	R1H/R6/~~A1~~	R3L/R5/~~A2~~	R3H/R7/~~A3~~	~~R1L~~/R4/A0	~~R1H~~/R6/A1	~~R3L~~/R5/A2	~~R3H~~/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM *2	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

*2. The effective value is -128 ≤ IMM(EX) ≤ +127(.BW, .BL) or -32768 ≤ IMM(EX) ≤ +32767(.WL).

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
EXTS.BW   R0L,R1
EXTS.BL   R2L,R6R4
EXTS.WL   R3,[A0]
```

EXTZ

Zero Extend

EXTZ

[Instruction Syntax]

EXTZ.size(:format) src,dest

size: G , S
format: BW , BL , WL

[Instruction Code/Number of Cycles]

Page=226

[Operation]

dest = (unsigned short | unsigned long) src;

[Description]

- This instruction zero-extends src and stores the result to dest.
- When size specifier (.size) is ".BW", 8-bit src is zero-extended to 16 bits and stored to dest.
- When size specifier (.size) is ".BL", 8-bit src is zero-extended to 32 bits and stored to dest.
- When size specifier (.size) is ".WL", 16-bit src is zero-extended to 32 bits and stored to dest.

[Available src/dest] **(Refer to the next page for src/dest classified by format).**

src *1				dest *1			
R0L/R0/~~R2R0~~	R0H/R2/~~R3R1~~	R2L/R1/~~R6R4~~	R2H/R3/~~R7R5~~	~~R0L~~/R0/R2R0	~~R0H~~/R2/R3R1	~~R2L~~/R1/R6R4	~~R2H~~/R3/R7R5
R1L/R4/~~A0~~	R1H/R6/~~A1~~	R3L/R5/~~A2~~	R3H/R7/~~A3~~	~~R1L~~/R4/A0	~~R1H~~/R6/A1	~~R3L~~/R5/A2	~~R3H~~/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM *2	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. The effective value is 0 ≤ IMM(EX) ≤ +255 (.BW, .BL) or 0 ≤ IMM(EX) ≤ +65535 (.WL).

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag is always set to 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
EXTZ.BW     R0L,R1
EXTZ.BL     R2L,R6R4
EXTZ.WL     R3,[A0]
EXTZ.WL     R0,A0
```

[src/dest Classified by Format]

G format

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

S format

src *1,*2				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	A0	A1	A2	A3
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3				
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24				
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]				
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]				
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]				
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]				
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]				

Notes:

*1. Only ".WL" can be specified as size specifier (.size). Therefore, each operand has a fixed bit length; 16 bits for src and 32 bits for dest.

*2. Indirect instruction addressing and Bank1 register direct addressing are available.

FCLR

Clear a Flag

FCLR

[Instruction Syntax]

FCLR dest

[Instruction Code/Number of Cycles]

Page=228

[Operation]

dest = 0;

[Description]

- This instruction stores 0 to dest.

[Available dest]

dest							
U	I	O	B	S	Z	D	C

[Flags Affected]

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

Note:

*1. The selected flag becomes 0.

[Sample Description]

```
FCLR I
FCLR S
FCLR O
```

FREIT

Return from Fast Interrupt

FREIT

[Instruction Syntax]

FREIT

[Instruction Code/Number of Cycles]

Page=228

[Operation]

```
FLG  =  SVF;
PC   =  SVP;
```

[Description]

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

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status *1	✓	✓	✓	✓	✓	✓	✓	✓

Note:

*1. Every flag has the value before the interrupt was generated, that is, the value saved in the save flag register (SVF).

[Sample Description]

FREIT

FSET

Set a Flag

FSET

[Instruction Syntax]

FSET dest

[Instruction Code/Number of Cycles]

Page=229

[Operation]

dest = 1;

[Description]

- This instruction stores 1 to dest.

[Available dest]

dest							
U	I	O	B	S	Z	D	C

[Flags Affected]

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

Note:

*1. The selected flag becomes 1.

[Sample Description]

```
FSET I
FSET S
FSET O
```

INC

Increment

INC

[Instruction Syntax]

INC.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=229

[Operation]

dest = dest + 1;

[Description]

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

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
INC.B   R0L
INC.W   R2
INC.L   R7R5
INC.L   A0
INC.W   mem[SB]
```

INDEX*Type*

Index

INDEX*Type*

[Instruction Syntax]

INDEX*Type*.size src

B , W , L

[Instruction Code/Number of Cycles]

Page=230

[Operation]

[Description]

- This instruction modifies the addressing modes of next instruction.
- No interrupt request is accepted until the next instruction execution is completed.
- Use this instruction to access arrays.
- Three instructions for Type show as below:
- Refer to **3.3, "INDEX Instruction"** for details.

Type	Function
B	Modifies both src and dest if the next instruction has 2 generic-format operands.
1	Modifies src if the next instruction has 2 generic-format operands. If it has only one generic or short-format operand, it modifies said operand.
2	Modifies dest if the next instruction has 2 generic-format operands.

[Available src]

src *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
INDEX1.B    R0L
INDEXB.W    R2
```

INT

Interrupt

INT

[Instruction Syntax]

INT src

[Instruction Code/Number of Cycles]

Page=231

[Operation]

```
SP    =   SP - 4;
*SP   =   FLG;
SP    =   SP - 4;
*SP   =   PC + 2;
PC    =   *(IntBase + src * 4);
```

[Description]

- This instruction generates a software interrupt according to the interrupt number specified by src.
- The interrupt number (src) is 0 ≤ src≤ 255.
- If src is ≤ 127, the U flag is set to 0 and the interrupt stack pointer (ISP) is used.
- If 128 ≤ src, stack pointer specified by the U flag is used.
- The interrupts generated by INT instruction are non-maskable.

[Available src]

src
#IMM:8 *1

Note:

*1. #IMM:8 is a software interrupt number whose range is 0 ≤ IMM:8 ≤ 255.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	✓	✓	–	–	–	–	✓	–

Note:

*1. The flags are saved to the stack area before the INT instruction is executed.

Description

U : The flag becomes 0 when the software interrupt number is 127 or under. Otherwise it has no change.
I : The flag becomes 0.
D : The flag becomes 0.

[Sample Description]

```
INT     #0
```

INTO

Interrupt on Overflow

INTO

[Instruction Syntax]

INTO

[Instruction Code/Number of Cycles]

Page=231

[Operation]

```
SP    =    SP - 4;
*SP   =    FLG;
SP    =    SP - 4;
*SP   =    PC + 1;
PC    =    *(0xFFFFFFE0);
```

[Description]

- When the O flag is 1, this instruction generates an overflow interrupt.
- When the O flag is 0, no overflow interrupt is generated.
- The overflow interrupt is non-maskable.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	✓	✓	–	–	–	–	✓	–

Note:

*1. The flags are saved to the stack area before the INTO instruction is executed.

Description

U : The flag becomes 0.
I : The flag becomes 0.
D : The flag becomes 0.

[Sample Description]

INTO

JCnd

Jump Conditionally

JCnd

[Instruction Syntax]

JCnd label

[Instruction Code/Number of Cycles]

Page=232

[Operation]

if (true)

PC = label; (= PC + dsp)

[Description]

- This instruction branches to label if the condition specified by *Cnd* is true; if false, it falls through to the next sequential instruction.
- The following table lists *Cnd* types:

Cnd	Conditions		Expression	*Cnd*	Conditions		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 to/ Z flag is 0.	≠
GTU	C & ~Z == 1	Greater than	<	LEU	C & ~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 as signed integer	≤	LE	(S ^ O) | Z == 1	Equal to or smaller than as signed integer	≥
GT	(S ^ O) | Z==0	Greater than as signed integer	<	LT	S ^ O == 1	Smaller than as signed integer	>
O	O == 1	O flag is 1.		NO	O == 0	O flag is 0.	

[Available label]

label
PC *1 + dsp:8 *2

Notes:

*1. PC indicates an address of J*Cnd* instruction plus 1.

*2. The dsp:8 is -128 ≤ dsp ≤ +127.

[Flags Affected]

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

[Sample Description]

```
JC      label1
JLT     label2
```

JMP

Jump Always

JMP

[Instruction Syntax]

JMP(.length) label

.length: S , B , W , A

[Instruction Code/Number of Cycles]

Page=232

[Operation]

PC = label; (= PC + dsp)

[Description]

- This instruction branches to label.

[Available label]

.length	label	
.S	PC *1 + dsp:3	1 ≤ dsp:3 ≤ 8
.B	PC *1 + dsp:8	-128 ≤ dsp:8 ≤ 127
.W	PC *1 + dsp:16	-32768 ≤ dsp:16 ≤ 32767
.A	PC *1 + dsp:24	-8388608 ≤ dsp:24 ≤ 8388607

Note:

*1. PC is an address of JMP instruction plus 1.

[Flags Affected]

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

[Sample Description]

```
JMP.B   label1
JMP.A   label2
```

JMPI

Jump Indirectly

JMPI

[Instruction Syntax]

JMPI.length src

length: W , L

[Instruction Code/Number of Cycles]

Page=233

[Operation]

When jump distance specifier (.length) is ".W"
PC = PC *1 + src;

When jump distance specifier (.length) is ".L"
PC = src;

Note:

*1. PC is an address of JMP instruction.

[Description]

- This instruction executes a relative/absolute branch to the address according to the specified src.
- When jump distance specifier (.length) is ".W", this instruction executes jump operation to the address which is the result of addition operation as signed integers: PC+src. If src is in memory, 2 bytes of memory capacity is required.
- When jump distance specifier (.length) is ".L", this instruction executes jump operation to the address specified by src. If src is in memory, 4 bytes of memory capacity is required.

[Available src]

src *1			
~~R0L~~/R0/R2R0	~~R0H~~/R2/R3R1	~~R2L~~/R1/R6R4	~~R2H~~/R3/R7R5
~~R1L~~/R4/A0	~~R1H~~/R6/A1	~~R3L~~/R5/A2	~~R3H~~/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
JMPI.W    R0
JMPI.L    A2
JMPI.W    mem[A0]
```

JSR

Jump to Subroutine

JSR

[Instruction Syntax]

JSR(.length) label

└──────── W , A

[Instruction Code/Number of Cycles]

Page=234

[Operation]

SP = SP - 4;
*SP = (PC + n) *1;
PC = label; (= PC + dsp)

Note:

*1. (PC+n) is an address of the next instruction of JSR.

[Description]

• This instruction branches to a subroutine specified by label.

[Available label]

.length	label	
.W	PC *1+ dsp:16	-32768 ≤ dsp:16 ≤ 32767
.A	PC *1 + dsp:24	-8388608 ≤ dsp:24 ≤ 8388607

Note:

*1. PC is an address of JSR instruction plus 1.

[Flags Affected]

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

[Sample Description]

```
JSR.W    subroutine1
JSR.A    subroutine2
```

JSRI

Jump to Subroutine Indirectly

JSRI

[Instruction Syntax]

JSRI.length src

length: W , L

[Instruction Code/Number of Cycles]

Page=234

[Operation]

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

SP = SP - 4;
*SP = (PC + n) *1;
PC = PC *2 + src;

When jump distance specifier (.length) is ".L"

SP = SP - 4;
*SP = (PC + n) *1;
PC = src;

Notes:

*1. (PC+n) is an address of the next instruction of JSRI.
*2. PC is an address of JSRI instruction

[Description]

- This instruction executes a relative/absolute subroutine branch according to the specified src.
- When jump distance specifier (.length) is ".W", the instruction branches to the address which is the result of addition operation as signed integers: PC+src. If src is in memory, 2 bytes of memory capacity is required.
- When jump distance specifier (.length) is ".L", this instruction branches to the address specified by src. If src is in memory, 4 bytes of memory capacity is required.

[Available src]

src *1			
~~R0L~~/R0/R2R0	~~R0H~~/R2/R3R1	~~R2L~~/R1/R6R4	~~R2H~~/R3/R7R5
~~R1L~~/R4/A0	~~R1H~~/R6/A1	~~R3L~~/R5/A2	~~R3H~~/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
JSRI.W    R0
JSRI.L    A2
JSRI.W    mem[A0]
```

LDC

Load into Control Register

LDC

[Instruction Syntax]

LDC src,dest

[Instruction Code/Number of Cycles]

Page=235

[Operation]

dest = src;

[Description]

- This instruction moves src to dest.

[Available src/dest]

src *1				dest			
~~R0L/R0/~~R2R0	~~R0H/R2/~~R3R1	~~R2L/R1/~~R6R4	~~R2H/R3/~~R7R5	SB	FB	FLG	SP
~~R1L/R4/~~A0	~~R1H/R6/~~A1	~~R3L/R5/~~A2	~~R3H/R7/~~A3	ISP	SVF	SVP	INTB
#IMMEX *2	dsp:8[FB]	dsp:16[FB]	dsp:24	DSA0	DSA1	DSA2	DSA3
#IMM *3	dsp:16	dsp:16[SB]	dsp:24[SB]	DDA0	DDA1	DDA2	DDA3
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	DCT0	DCT1	DCT2	DCT3
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	DSR0	DSR1	DSR2	DSR3
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	DDR0	DDR1	DDR2	DDR3
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	DCR0	DCR1	DCR2	DCR3
				DMD0	DMD1	DMD2	DMD3
				VCT			

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

*2. #IMMEX cannot be used as src of DMAC related registers nor that of VCT. When #IMM:8 or #IMM:16 is specified instead, zero-extended src is loaded.

*3. #IMM:8 and #IMM:16 can only be used as src of DMAC related registers or that of VCT.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	*1	*1	*1	*1	*1	*1	*1	*1	*1	*1

Note:

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

[Sample Description]

```
LDC     #00800000h,SB
LDC     R6R4,DSA1
LDC     A1,DMD2
LDC     mem[A0],DCT1
```

LDCTX

Load Context

LDCTX

[Instruction Syntax]

LDCTX src,dest

[Instruction Code/Number of Cycles]

Page=236

[Operation]

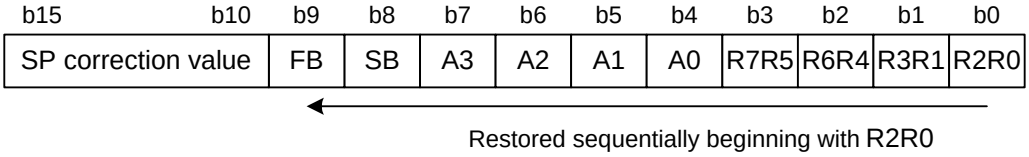
```
for (i=0 ; i<n *1 ; i++) {
        register (dest[src])   =     *SP;
        SP   =     SP + 4;
}
```

Note:

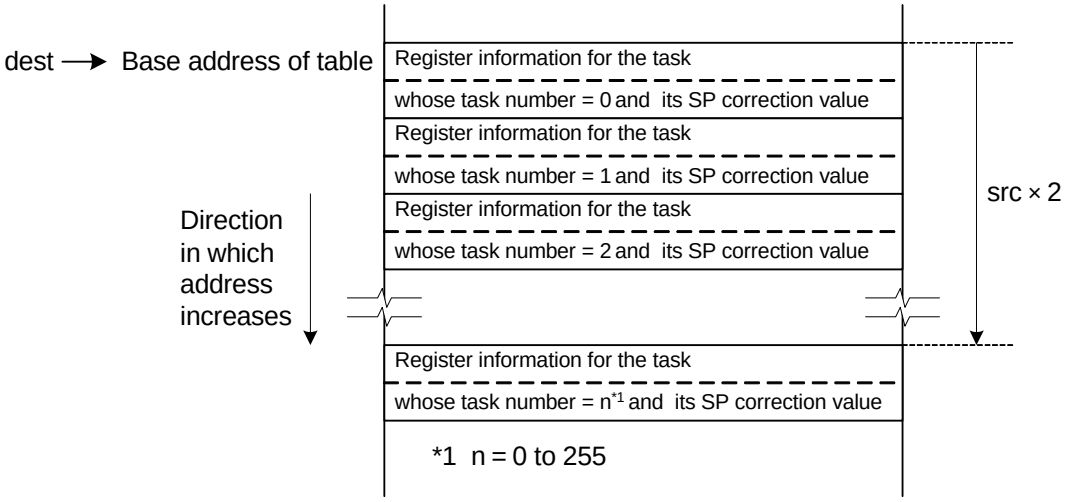
*1. n is the number of registers to be restored.

[Description]

- This instruction restores the task context from the stack area.
- Set the RAM address that contains the task number in src and the start address of table entries in dest.
- The required register information is specified from table entries by the task number, and the data in the stack area is restored to each register according to the specified register information. Then the stack pointer (SP) correction value is added to the stack pointer (SP). For this SP correction value, set the total bytes of registers to be restored. Do not set any table entries to 0000h.
- Information on registers is configured as shown below. 1 indicates a register to be restored and 0 indicates a register that is not to be restored.



- A table is configured as shown in the following page. The base address of the table is specified by dest. Each table entry is 16-bit data composed of a register information from bit 0 to bit 9 and a stack pointer correction value from bit 10 to bit 15 (Refer to the figure above). This word data is stored at an address in which the content of src is duplicated.



[Available src/dest]

src	dest
abs:16	dsp:24

[Flags Affected]

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

[Sample Description]

```
LDCTX    ram,rom_tbl
```

LDIPL

Load Interrupt Priority Level

LDIPL

[Instruction Syntax]

LDIPL src

[Instruction Code/Number of Cycles]

Page=237

[Operation]

IPL = src;

[Description]

- This instruction loads src to IPL.

[Available src]

src
#IMM:3

[Flags Affected]

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

[Sample Description]

LDIPL #2

MAX

Select Maximum Value

MAX

[Instruction Syntax]

MAX.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=237

[Operation]

```
if (src > dest)
        dest =  src;
```

[Description]

- This instruction compares src and dest as signed integers and stores the greater value to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
MAX.B       #-50,R1L
MAX.W       mem[A1],R2
MAX.L       R7R5,[A0]
```

MIN

Select Minimum Value

MIN

[Instruction Syntax]

MIN.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=239

[Operation]

```
if (src < dest)
        dest =  src;
```

[Description]

- This instruction compares src and dest as signed integers and stores the smaller value to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
MIN.B       #-50,R1L
MIN.W       mem[A1],R2
MIN.L       R7R5,[A0]
```

MOV

Move

MOV

[Instruction Syntax]

MOV.size(:format) src,dest

G , Q , Z , S
B , W , L

[Instruction Code/Number of Cycles]

Page=241

[Operation]

dest = src;

[Description]

- This instruction moves src to dest.

[Available src/dest] **(Refer to the next page for src/dest classified by format.)**

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:4	dsp:8[SP]			dsp:8[SP]	dsp:8[SB]		

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
MOV.B:Z     #0,R0L
MOV.W       R0,R2
MOV.L       A0,R7R5
MOV.B       R3L,[mem[A0]]
MOV.W       [[A1]],R4
MOV.L       R2R0,[[A3]]
```

[src/dest Classified by Format]

G format

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
dsp:8[SP] *2*4				dsp:8[SP] *2,*4			

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. Operation is performed on the stack pointer specified by the U flag. dsp:8 [SP]s for src and dest cannot be selected simultaneously.
*3. If dsp:8[SP] for src is selected, indirect instruction addressing for dest cannot be selected.
*4. If dsp:8[SP] for dest is selected, either #IMMEX or #IMM for src cannot be selected.

Q format

src	dest *1			
#IMM:4 *2	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
		dsp:8[FB]	dsp:16[FB]	dsp:24
		dsp:16	dsp:16[SB]	dsp:24[SB]
	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. The effective value is - 8 ≤ #IMM:4 ≤ +7.

S format

src *1				dest *1			
#IMM				R0L/R0/R2R0	dsp:16	dsp:8[SB]	dsp:8[FB]
R0L/R0/R2R0				dsp:16	dsp:8[SB]	dsp:8[FB]	
R0H/R2/R3R1	dsp:16	dsp:8[SB]	dsp:8[FB]	R0L/R0/R2R0			
dsp:16	dsp:8[SB]	dsp:8[FB]		A0 *2,*3			

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. Bank1 register direct addressing is unavailable.
*3. Only ".L" is specified as a size specifier (.size).

Z format

src	dest *1			
#0	R0L/R0/R2R0	dsp:16	dsp:8[SB]	dsp:8[FB]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

MOVA

Move Effective Address

MOVA

[Instruction Syntax]

MOVA src,dest

[Instruction Code/Number of Cycles]

Page=246

[Operation]

dest = &src;

[Description]

- This instruction moves the effective address in src to dest.

[Available src/dest]

src				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Bank1 register direct addressing is available.

[Flags Affected]

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

[Sample Description]

```
MOVA    mem[FB],A0
MOVA    mem[A1],R7R5
```

MOV*Dir*

Move Nibble Data

MOV*Dir*

[Instruction Syntax]

MOV*Dir* src,dest

[Instruction Code/Number of Cycles]

Page=247

[Operation]

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

[Description]

Dir	Function
HH	Transfers src (8 bits)'s upper 4 bits to dest (8 bits)'s upper 4 bits.
HL	Transfers src (8 bits)'s upper 4 bits to dest (8 bits)'s lower 4 bits.
LH	Transfers src (8 bits)'s lower 4 bits to dest (8 bits)'s upper 4 bits.
LL	Transfers src (8 bits)'s lower 4 bits to dest (8 bits)'s lower 4 bits.

- R0L must be specified for either src or dest.

[Available src/dest]

src *1				dest *1			
R0L/~~R0/R2R0~~	R0H/~~R2/R3R1~~	R2L/~~R1/R6R4~~	R2H/~~R3/R7R5~~	R0L/~~R0/R2R0~~*2			
R1L/~~R4/A0~~	R1H/~~R6/A1~~	R3L/~~R5/A2~~	R3H/~~R7/A3~~				
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24				
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]				
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]				
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]				
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]				
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]				
R0L/~~R0/R2R0~~*2				R0L/~~R0/R2R0~~	R0H/~~R2/R3R1~~	R2L/~~R1/R6R4~~	R2H/~~R3/R7R5~~
				R1L/~~R4/A0~~	R1H/~~R6/A1~~	R3L/~~R5/A2~~	R3H/~~R7/A3~~
					dsp:8[FB]	dsp:16[FB]	dsp:24
					dsp:16	dsp:16[SB]	dsp:24[SB]
				[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
				[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
				[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
				[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

*2. Bank1 register direct addressing is unavailable.

[Flags Affected]

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

[Sample Description]

```
MOVHH   R0L,[A0]
MOVHL   mem[A2],R0L
```

MUL

Signed Multiply

MUL

[Instruction Syntax]

MUL.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=249

[Operation]

dest = dest * src;

[Description]

- This instruction multiplies src and dest and stores the result to dest. Both operands and the result are interpreted as signed integers.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : This flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
MUL.B   #3,R0L
MUL.W   R2,R3
MUL.L   [A3],R6R4
```

MULF

Multiply Floating-point

MULF

[Instruction Syntax]

MULF src,dest

[Instruction Code/Number of Cycles]

Page=251

[Operation]

dest = dest * src;

[Description]

- This instruction multiplies src and dest and stores the result to dest. Both operands and the result are interpreted as unsigned integers.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG).
- When the result is overflowed, it takes either the maximum positive normal number (7F7FFFFFh) or the maximum negative normal number (FF7FFFFFh) depending on whether signed or not.
- When the result is underflowed, it takes any of the following according to the rounding mode: zero (00000000h), the minimum positive number (00800000h) or the minimum negative normal number (80800000h).
- The result for invalid numbers is undefined.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~R2R0	~~R0H/R2~~R3R1	~~R2L/R1~~R6R4	~~R2H/R3~~R7R5	~~R0L/R0~~R2R0	~~R0H/R2~~R3R1	~~R2L/R1~~R6R4	~~R2H/R3~~R7R5
~~R1L/R4~~A0	~~R1H/R6~~A1	~~R3L/R5~~A2	~~R3H/R7~~A3	~~R1L/R4~~A0	~~R1H/R6~~A1	~~R3L/R5~~A2	~~R3H/R7~~A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

FU : The flag becomes 1 when the operand has invalid numbers or when the operation results in an underflow; otherwise it becomes 0.

O : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results is 0; otherwise it becomes 0.

C : The flag is undefined.

[Sample Description]

```
MULF    R2R0,R3R1
MULF    [A0],R2R0
MULF    mem[FB],R3R1
```

MULU

Unsigned Multiply

MULU

[Instruction Syntax]

MULU.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=253

[Operation]

dest = dest * src;

[Description]

- This instruction multiplies src and dest and stores the result to dest. Both operands and the result are interpreted as unsigned integers.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when the operation results in exceeding 8 bits (.B), 16 bits (.W), or 32 bits (.L): otherwise it becomes 0.

[Sample Description]

```
MULU.B    #3,R0L
MULU.W    R2,R3
MULU.L    [A3],R6R4
```

MULX

Signed Multiply and Round

MULX

[Instruction Syntax]

MULX.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=255

[Operation]

```
short | long | long long        tmp;

tmp *1  =      (short | long | long long)dest * src;
if (DP==0)
        dest =      (char | short | long)(tmp >> n *2);
else
        dest =      (short | long)(tmp >> m *3);
```

Notes:

*1. tmp: temporary registers
*2. When size specifier (.size) is ".B", n = 6.
When size specifier (.size) is ".W", n = 14.
When size specifier (.size) is ".L", n = 30.
*3. When size specifier (.size) is ".W", n = 8.
When size specifier (.size) is ".L", n = 16.

[Description]

- This instruction multiplies src and dest, shifts and rounds off the result according to the DP bit (the bit 16 of the flag register (FLG)), and stores it to dest.
- When size specifier (.size) is ".B", both src and dest are operated in 8 bits and the result is stored in 8 bits.
- When size specifier (.size) is ".W", both src and dest are operated in 16 bits and the result is stored in 16 bits.
- When size specifier (.size) is ".L", both src and dest are operated in 32 bits and the result is stored in 32 bits.
- This instruction performs fixed-point multiplication. (Refer to **Figure 3.1**).

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24				
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]				
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]				
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]				
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]				
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]				

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

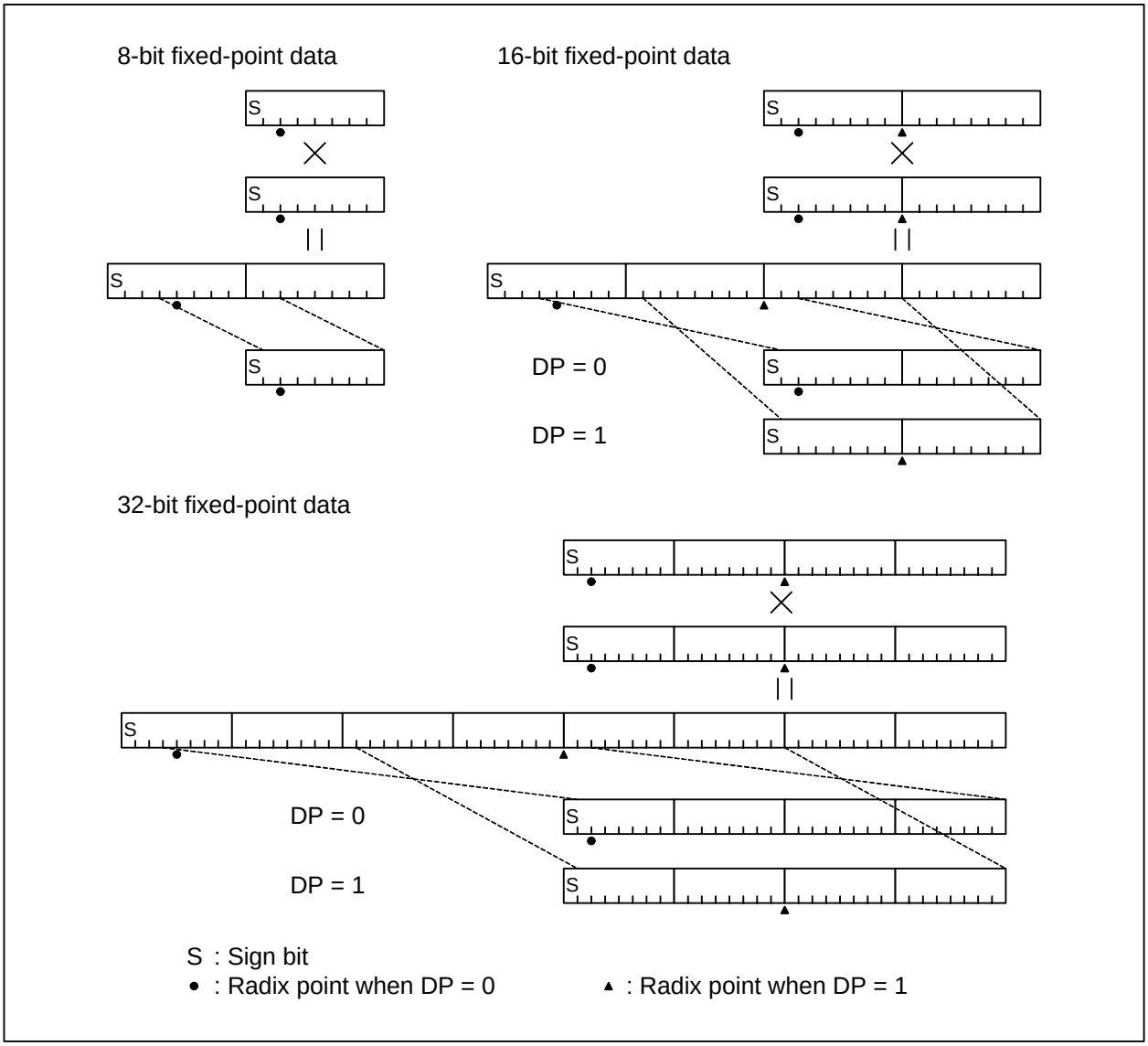


Figure 3.1 Application of Fixed-point Multiplication

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : This flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

[Sample Description]

```
MULX.B    #12,R0L        ; (R0L×12)>>6→R0L
MULX.W    R0,R1          ; (R1×R0)>>14 or 8→R1
MULX.L    A0,R6R4        ; (R6R4×A0)>>30 or 16→R6R
```

NEG

Negate

NEG

[Instruction Syntax]

NEG.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=255

[Operation]

dest = -dest;

[Description]

- This instruction takes the twos complement of dest and stores the result to dest.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag becomes 1 when dest before the operation is -128(.B), -32768(.W), or -2147483648(.L); otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the operation results in 1; otherwise it becomes 0.

[Sample Description]

```
NEG.B   R0L
NEG.W   R3
NEG.L   [A0]
```

NOP

No Operation

NOP

[Instruction Syntax]

NOP

[Instruction Code/Number of Cycles]

Page=256

[Operation]

PC = PC + 1;

[Description]

- This instruction adds 1 to PC.

[Flags Affected]

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

[Sample Description]

NOP

NOT

Logical Complement

NOT

[Instruction Syntax]

NOT.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=256

[Operation]

dest = ~dest;

[Description]

- This instruction logically inverts dest and stores the result to dest.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
NOT.B   R0L
NOT.W   R3
NOT.L   [A0]
```

OR

Or Logical

OR

[Instruction Syntax]

OR.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=257

[Operation]

dest = dest | src;

[Description]

- This instruction logically ORs dest and src and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
OR.B    #2,R0L
OR.W    R0,R2
OR.L    A0,R7R5
OR.B    R3L,[mem[A0]]
OR.W    [[A1]],R4
OR.L    R2R0,[[A3]]
```

POP

Pop Data off the Stack

POP

[Instruction Syntax]

POP.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=258

[Operation]

dest = *SP;
SP = SP + 4 *1;

Note:

*1. SP is always increased by 4 in any size specifier (.size) as ".B" or ".W".

[Description]

- This instruction moves the data restored from the stack area to dest.
- The stack pointer to be used is specified by the U flag.

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
POP.B   R0L
POP.W   R3
POP.L   R6R4
```

POPC

Pop Control Register off the Stack

POPC

[Instruction Syntax]

POPC dest

[Instruction Code/Number of Cycles]

Page=259

[Operation]

dest = *SP;
SP = SP + 4;

[Description]

- This instruction moves the data restored from the stack area to the control register specified by dest.
- The stack pointer to be used is specified by the U flag.

[Available dest]

dest			
SB	FB	FLG	SP *1
ISP	SVF	SVP	INTB

Note:

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

[Flags Affected]

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

Note:

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

[Sample Description]

```
POPC    SB
POPC    SVF
```

POPM

Pop Registers off the Stack

POPM

[Instruction Syntax]

POPM dest

[Instruction Code/Number of Cycles]

Page=259

[Operation]

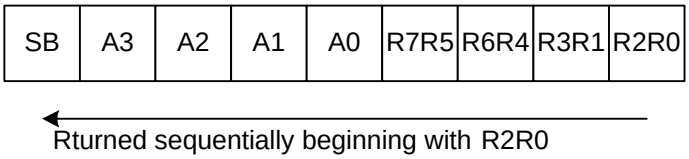
```
for (i=0 ; i< n *1 ; i++) {
        registers (dest) =    *SP;
        SP   =    SP + 4;
}
```

Note:

*1. n is the number of registers to be restored.

[Description]

- This instruction collectively restores the registers selected by dest from the stack area.
- The stack pointer to be used is specified by the U flag.
- Registers are restored from the stack area in the following order:



[Available dest]

dest *1,*2			
R2R0	R3R1	R6R4	R7R5
A0	A1	A2	A3
SB			

Notes:

*1. Multiple registers can be specified for dest.

*2. Bank 1 register direct addressing is available. Note that register direct addressing and bank 1 register direct addressing cannot be used simultaneously.

[Flags Affected]

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

[Sample Description]

```
POPM        R6R4,A3,SB
POPM        R2R0,R3R1,A0,A1
```

PUSH

Push Data on the Stack

PUSH

[Instruction Syntax]

PUSH.size(:format) src

G , S
B , W , L

[Instruction Code/Number of Cycles]

Page=260

[Operation]

SP = SP - 4 *1;
*SP = src;

Note:

*1. SP is always decreased by 4 in any size specifier (.size) as ".B" or ".W". The upper 24 bits in ".B" and the upper 16 bits in ".W" are undefined.

[Description]

- This instruction stores src to the stack area.
- The stack pointer to be used is specified by the U flag.

[Available src] **(Refer to the next page for src classified by format.)**

src *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
PUSH.B    R0L
PUSH.W    #1000
PUSH.L    [mem[A0]]
PUSH.W    #FF80h
```

[src classified by format]

G format

src *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

S format

src	
#IMMEX	#IMM

PUSHA

Push Effective Address on the Stack

PUSHA

[Instruction Syntax]

PUSHA src

[Instruction Code/Number of Cycles]

Page=262

[Operation]

```
SP   =   SP - 4;
*SP  =   &src;
```

[Description]

- This instruction stores the effective address in src to the stack area.
- The stack pointer to be used is specified by the U flag.

[Available src]

src			
~~R0L/R0/R2R0~~	~~R0H/R2/R3R1~~	~~R2L/R1/R6R4~~	~~R2H/R3/R7R5~~
~~R1L/R4/A0~~	~~R1H/R6/A1~~	~~R3L/R5/A2~~	~~R3H/R7/A3~~
~~#IMMEX~~	dsp:8[FB]	dsp:16[FB]	dsp:24
~~#IMM~~	dsp:16	dsp:16[SB]	dsp:24[SB]
~~[A0]~~	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
~~[A1]~~	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
~~[A2]~~	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
~~[A3]~~	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

[Flags Affected]

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

[Sample Description]

```
PUSHA   mem[FB]
PUSHA   mem[A0]
```

PUSHC

Push Control Register on the Stack

PUSHC

[Instruction Syntax]

PUSHC src

[Instruction Code/Number of Cycles]

Page=262

[Operation]

```
SP   =   SP - 4;
*SP  =   src;
```

[Description]

- This instruction stores the control register specified by dest to the stack area.
- The stack pointer to be used is specified by the U flag.

[Available src]

src			
SB	FB	FLG	SP *1
ISP	SVF	SVP	INTB

Note:

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

[Flags Affected]

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

[Sample Description]

```
PUSHC   SB
PUSHC   SVF
```

PUSHM

Push Registers on the Stack

PUSHM

[Instruction Syntax]

PUSHM src

[Instruction Code/Number of Cycles]

Page=262

[Operation]

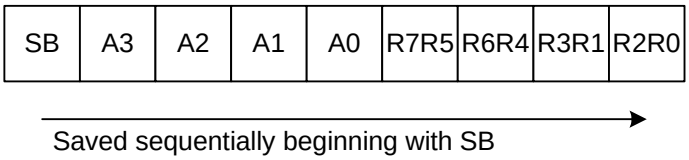
```
for (i=0 ; i< n *1 ;i++) {
        SP  =   SP - 4;
        *SP =   src;
}
```

Note:

*1. n is the number of registers to be stored.

[Description]

- This instruction collectively stores the registers selected by src to the stack area.
- The stack pointer to be used is specified by the U flag.
- Registers are stored to the stack area in the following order:



[Available src]

src *1,*2			
R2R0	R3R1	R6R4	R7R5
A0	A1	A2	A3
SB			

Notes:

*1. Multiple registers can be specified for src.

*2. Bank 1 register direct addressing is available. Note that register direct addressing and bank 1 register direct addressing cannot be used simultaneously.

[Flags Affected]

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

[Sample Description]

```
PUSHM   R6R4,A3,SB
PUSHM   R2R0,R3R1,A0,A1
```

REIT

Return from Interrupt

REIT

[Instruction Syntax]

REIT

[Instruction Code/Number of Cycles]

Page=263

[Operation]

```
PC    =    *SP;
SP    =    SP + 4;
FLG   =    *SP;
SP    =    SP + 4;
```

[Description]

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

[Flags Affected]

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

Note:

*1. It is the value on the stack.

[Sample Description]

REIT

RMPA

Repeat Multiply and Accumulation

RMPA

[Instruction Syntax]

RMPA.size

B , W , L

[Instruction Code/Number of Cycles]

Page=263

[Operation]

```
While (R7R5 != 0) *1 {
        R3R1:R2R0   =   R3R1:R2R0 + *A0 * *A1;
        A0          =   A0 + n *2;
        A1          =   A1 + n *2;
        R7R5        =   R7R5 - 1;
}
```

Notes:

*1. This instruction is ignored if it is executed setting 0 in R7R5.

*2. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.

[Description]

- This instruction performs sum-of-product operation, with the multiplicand address indicated by A0, the multiplier address indicated by A1, and the number of operation indicated by R7R5. The result is stored to R3R1:R2R0 as 64-bit data. Both operands and the result are interpreted as signed integers.
- The maximum value to be set in R7R5 is 00020000h (131072).
- R4 and R6 are used as working registers. The interim result of the operation is accumulated in R4:R3R1:R2R0.
- The contents of the address register, R4 and R6 on the instruction completion are undefined.
- Set the initial value in R3R1:R2R0 before instruction execution. Set FFFFh in R4 when R3R1:R2R0 is negative and set 0000h in the reverse case.
- An interrupt request during the instruction execution is accepted with an interruption of the execution.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	–	–	–	–

Description

O : The flag becomes 1 when the result is over $2^{63} - 1$ or -2^{63}; otherwise it becomes 0.

[Sample Description]

RMPA.W

ROLC

Rotate the bits to the Left with Carry

ROLC

[Instruction Syntax]

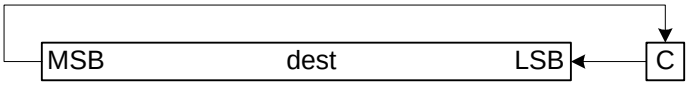
ROLC.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=263

[Operation]



[Description]

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

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the shifted-out bit is 1; otherwise it becomes 0.

[Sample Description]

```
ROLC.B    R0L
ROLC.W    [A0]
ROLC.L    R2R0
```

RORC

Rotate the bits to the Right with Carry

RORC

[Instruction Syntax]

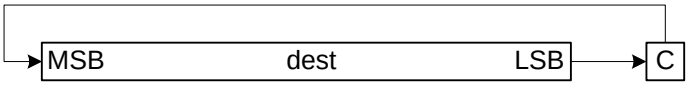
RORC.size dest

B , W , L

[Instruction Code/Number of Cycles]

Page=264

[Operation]



[Description]

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

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	✓

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the shifted-out bit is 1; otherwise it becomes 0.

[Sample Description]

```
RORC.B    R0L
RORC.W    [A0]
RORC.L    R2R0
```

ROT

Rotate the bits

ROT

[Instruction Syntax]

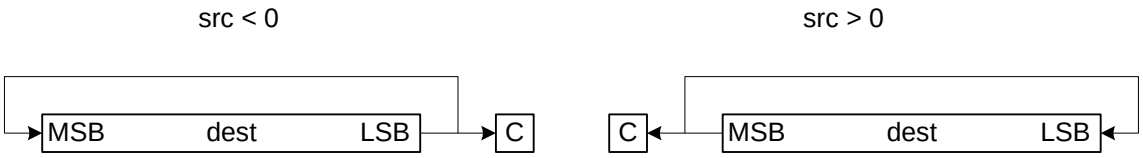
ROT.size src,dest

size: B , W , L

[Instruction Code/Number of Cycles]

Page=264

[Operation]



[Description]

- This instruction rotates dest left or right as many as the number specified by src. The bit overflowing from LSB/MSB is moved to MSB/LSB and the C flag.
- The rotation direction is specified by the sign of src. When src is positive, bits are rotated left; when it is negative, they are rotated right. No rotation occurs when it is 0.

[Available src/dest]

src *1,*2				dest *3			
R0L/~~R0/R2R0~~	R0H/~~R2/R3R1~~	R2L/~~R1/R6R4~~	R2H/~~R3/R7R5~~	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/~~R4/A0~~	R1H/~~R6/A1~~	R3L/~~R5/A2~~	R3H/~~R7/A3~~	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	~~dsp:8[FB]~~	~~dsp:16[FB]~~	~~dsp:24~~		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM:8	~~dsp:16~~	~~dsp:16[SB]~~	~~dsp:24[SB]~~		dsp:16	dsp:16[SB]	dsp:24[SB]
~~[A0]~~	~~dsp:8[A0]~~	~~dsp:16[A0]~~	~~dsp:24[A0]~~	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
~~[A1]~~	~~dsp:8[A1]~~	~~dsp:16[A1]~~	~~dsp:24[A1]~~	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
~~[A2]~~	~~dsp:8[A2]~~	~~dsp:16[A2]~~	~~dsp:24[A2]~~	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
~~[A3]~~	~~dsp:8[A3]~~	~~dsp:16[A3]~~	~~dsp:24[A3]~~	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Bank1 register direct addressing is available.
*2. The effective value for #IMM:8 is -8 to +8(.B), -16 to +16(.W) or -32 to +32(.L).
*3. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	–	–	–	–	✓	✓	–	✓

Note:

*1. When src is 0, any flag has no change.

Description

S : The flag becomes 1 when the operation resulted in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation resulted in 0; otherwise it becomes 0.
C : The flag becomes 1 when the shifted-out bit is 1; otherwise it becomes 0.

[Sample Description]

```
ROT.B   #1,R0L      ;Rotated left
ROT.B   #-1,R0L     ;Rotated right
ROT.W   R1L,[A0]
ROT.L   R0H,R6R
```

ROUND

Round Floating-point to Integer

ROUND

[Instruction Syntax]

ROUND src,dest

[Instruction Code/Number of Cycles]

Page=265

[Operation]

dest = (long) src;

[Description]

- This instruction converts src to integer and stores the result to dest.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG) and clipped to satisfy the condition -2147483648 ≤ the value ≤ +2147483647.
- The result is undefined when src has an invalid number.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers or when the operation results in exceeding -2147483648 or +2147483647; otherwise it becomes 0.

FU : The flag becomes 1 when the operand has invalid numbers; otherwise it becomes 0.

O : The flag becomes 1 when the operand has invalid numbers or when the operation results in exceeding -2147483648 or +2147483647; otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MBS = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag is undefined.

[Sample Description]

```
ROUND   R2R0,R3R1
ROUND   [A1],R2R0
ROUND   mem[FB],R3R1
```

RTS

Return from Subroutine

RTS

[Instruction Syntax]

RTS

[Instruction Code/Number of Cycles]

Page=266

[Operation]

```
PC  =  *SP;
SP  =  SP + 4;
```

[Description]

- This instruction executes return operation from a subroutine.

[Flags Affected]

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

[Sample Description]

```
RTS
```

SBB

Subtract with Borrow

SBB

[Instruction Syntax]

SBB.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=267

[Operation]

dest = dest - src - ~C;

[Description]

• This instruction subtracts src and the inverted C flag (borrow) from dest and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. It becomes 1 when the operation results in exceeding -128(.B) or+127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when an operation as unsigned integers results in any value equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
SBB.B    #2,R0L
SBB.W    R0,R2
SBB.L    A0,R7R5
SBB.B    R3L,[A0]
SBB.W    [A1],R4
SBB.L    R2R0,[A3]
```

SC*Cnd* — Store Condition Conditionally — SC*Cnd*

[Instruction Syntax]

SC*Cnd*.size dest

B , W, L

[Instruction Code/Number of Cycles]

Page=268

[Operation]

```
if (true)
        dest =    1;
else
        dest =    0;
```

[Description]

- This instruction sets 1 to dest if the condition specified by *Cnd* is true; if false, it sets 0 to dest.
- The following table lists *Cnd* types:

Cnd	Conditions		Expression	*Cnd*	Conditions		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 to/ Z flag is 0.	≠
GTU	C & ~Z == 1	Greater than	<	LEU	C & ~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 as signed integer	≤	LE	(S ^ O) | Z == 1	Equal to or smaller than as signed integer	≥
GT	(S ^ O) | Z==0	Greater than as signed integer	<	LT	S ^ O == 1	Smaller than as signed integer	>
O	O == 1	O flag is 1.		NO	O == 0	O flag is 0.	

[Available dest]

dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
	dsp:8[FB]	dsp:16[FB]	dsp:24
	dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
SCC.B     R0L
SCNE.W    [A0]
SCGT.L    A3
```

SCMPU

Compare Strings until not equal

SCMPU

[Instruction Syntax]

SCMPU.size

B , W

[Instruction Code/Number of Cycles]

Page=269

[Operation]

```
unsigned char *A1, *A0, tmp0, tmp1;
do {
        tmp0 = *A0++;
        tmp1 = *A1++;
} while (tmp0 == tmp1 && tmp0 != '\0');

tmp0,tmp1:temporary registers
```

[Description]

- This instruction compares strings in the address incrementing direction from the comparison address (A0) to the compared address (A1) until the data content does not match or Null character '\0'(=00h) is detected.
- It has the same operation in any size specifier (.size) as ".B" or ".W".
- The contents of the address registers A0 and A1 are undefined on the instruction completion.
- An interrupt request during instruction execution is accepted with an interruption of the execution.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag is undefined.
S : The flag is undefined.
Z : The flag becomes 1 when the two strings are matched; otherwise it becomes 0.
C : The flag becomes 1 when an operation of (*A0 - *A1) as unsigned integers results in any value equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

SCMPU.W

SHA

Arithmetic Shift

SHA

[Instruction Syntax]

SHA.size src,dest

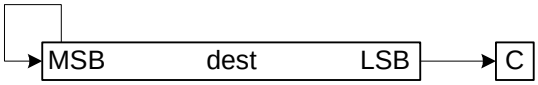
B , W , L

[Instruction Code/Number of Cycles]

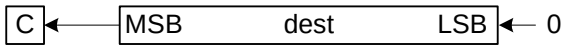
Page=269

[Operation]

src < 0



src > 0



[Description]

- This instruction arithmetically shifts dest left or right as many as the number specified by src. The bit overflowing from LSB/MSB is moved to the C flag.
- The shift direction is specified by the sign of src. When src is positive, bits are shifted left; when negative, bits are shifted right. They are not shifted when src is 0.

[Available src/dest]

src *1,*2				dest *3			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM:8	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Bank1 register direct addressing is available.
*2. The effective value for #IMM:8 is -8 to +8(.B), -16 to +16(.W), or -32 to +32(.L).
*3. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	–	–	✓	–	✓	✓	–	✓

Note:

*1. When src is 0, any flag has no change.

Description

O : On shifted left, the flag becomes 0 when all the shift resulted in MSB and shifted-out bit are the same value, that is, when no flags are changed during shifting; otherwise it becomes 1.
On shifted right, the flag becomes 0.
S : The flag becomes 1 when the operation resulted in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the shifted-out bit is 1; otherwise it becomes 0.

[Sample Description]

```
SHA.B    #3,R0L      ;shifted left
SHA.B    #-3,R0L     ;shifted right
SHA.W    R1L,[A0]
SHA.L    R2H,R6R4
```

SHL

Logical Shift

SHL

[Instruction Syntax]

SHL.size(:format) src,dest

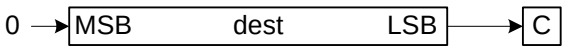
G , Q
B , W , L

[Instruction Code/Number of Cycles]

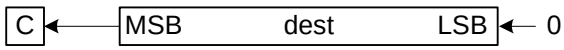
Page=270

[Operation]

src < 0



src > 0



[Description]

- This instruction logically shifts dest left or right as many as the number specified by src. The bit overflowing from LSB/MSB is moved to the C flag.
- The shift direction is specified by the sign of src. When src is positive, bits are shifted left; when negative, bits are shifted right.

[Available src/dest] **(Refer to the next page for src/dest classified by format).**

src *1				dest *2			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM:8 *3	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
#IMM:4 *4							

Notes:

*1. Bank1 register direct addressing is available.
*2. Indirect instruction addressing and Bank1 register direct addressing are available.
*3. The effective value for #IMM:8 is -8 to +8(.B), -16 to +16(.W), or -32 to +32(.L).
*4. The effective value is -8 ≤ #IMM:4 ≤ +7 (≠0).

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	–	–	–	–	✓	✓	–	✓

Note:

*1. When src is 0, any flag has no change.

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : he flag is set when the operation results in 0; otherwise it becomes 0.
C : The flag becomes 1 when the shifted-out bit is 1; otherwise it becomes 0.

[Sample Description]

```
SHL.B    #1,R0L      ;shifted left
SHL.B    #-1,R0L     ;shifted right
SHL.W    R1L,[A0]
SHL.L    R2H,R6R4
```

[src/dest Classified by Format]

G format

src *1				dest *2			
R0L/~~R0/R2R0~~	R0H/~~R2/R3R1~~	R2L/~~R1/R6R4~~	R2H/~~R3/R7R5~~	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/~~R4/A0~~	R1H/~~R6/A1~~	R3L/~~R5/A2~~	R3H/~~R7/A3~~	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
					dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM:8					dsp:16	dsp:16[SB]	dsp:24[SB]
				[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
				[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
				[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
				[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Bank1 register direct addressing is available.
*2. Indirect instruction addressing and Bank1 register direct addressing are available.

Q format

src *1	dest *1			
#IMM:4 *2	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
		dsp:8[FB]	dsp:16[FB]	dsp:24
		dsp:16	dsp:16[SB]	dsp:24[SB]
	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.
*2. The effective value is $-8 \leq$ #IMM:4 $\leq +7$ ($\neq 0$).

SIN

Input Strings

SIN

[Instruction Syntax]

SIN.size

B , W , L

[Instruction Code/Number of Cycles]

Page=272

[Operation]

```
while (R7R5 != 0) *1 {
        *A1     =   *A0;
        A1      =   A1 + n *2;
        R7R5    =   R7R5 - 1;
}
```

Notes:

*1. When the value 0 is set in R7R5, this instruction is ignored.
*2. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.

[Description]

- This instruction moves string in successively address incrementing direction from the fixed source address specified by A0 to the destination address specified by A1 as many as the number specified by R7R5.
- Set the source address in A0, the destination address in A1 and the number of moves in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- The address register A1 on the instruction completion indicates the next sequential address of the last data moved.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

SIN.W

SMOVB

Move Strings Backward

SMOVB

[Instruction Syntax]

SMOVB.size

B , W , L

[Instruction Code/Number of Cycles]

Page=272

[Operation]

```
while (R7R5 != 0) *1 {
        *A1     =   *A0;
        A0      =   A0 - n *2;
        A1      =   A1 - n *2;
        R7R5    =   R7R5 - 1;
}
```

Notes:

*1. When the value 0 is set in R7R5, this instruction is ignored.

*2. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.

[Description]

- This instruction moves string in successively address decrementing direction from the source address indicated by A0 to the destination address indicated by A1 as many as the number specified by R7R5.
- Set the source address in A0, the destination address in A1 and the number of moves in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- The address registers A0 and A1 on the instruction completion indicate the next sequential address of the last data transferred.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

SMOVB.W

SMOVF

Move Strings Forward

SMOVF

[Instruction Syntax]

SMOVF.size

B , W , L , Q

[Instruction Code/Number of Cycles]

Page=272

[Operation]

```
while (R7R5 != 0) *1 {
        *A1     =   *A0;
        A0      =   A0 + n *2;
        A1      =   A1 + n *2;
        R7R5    =   R7R5 - 1;
}
```

Notes:

*1. When the value 0 is set in R7R5, this instruction is ignored.
*2. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.
When size specifier (.size) is ".Q", n = 8.

[Description]

- This instruction moves string in successively address incrementing direction from the source address specified by A0 to the destination address specified by A1 as many as the number specified by R7R5.
- Set the source address in A0, the destination address in A1, and the number of moves in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- The address registers A0 and A1 on the instruction completion indicate the next sequential address of the last data transferred.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

SMOVF.W

SMOVU

Move Strings While Unequal to Zero

SMOVU

[Instruction Syntax]

SMOVU.size

B , W

[Instruction Code/Number of Cycles]

Page=273

[Operation]

```
unsigned char *A1, *A0, tmp0;
do {
        tmp0 = *A0++;
        *A1++ = tmp0
} while (tmp0 != '\0');

tmp0:   temporary registers
```

[Description]

- This instruction moves string in successively address incrementing direction from the source address specified by A0 to the destination address specified by A1 until 0 is detected.
- Set the source address in A0 and the destination address in A1.
- It has the same operation in any size specifier (.size) as ".B" or ".W".
- The contents of the address registers A0 and A1 are undefined on the instruction completion.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

SMOVU.W

SOUT

Output Strings

SOUT

[Instruction Syntax]

SOUT.size

B , W , L

[Instruction Code/Number of Cycles]

Page=273

[Operation]

```
while (R7R5 != 0) *1 {
        *A1     =   *A0;
        A0      =   A0 + n *2;
        R7R5    =   R7R5 - 1;
}
```

Notes:

*1. When the value 0 is set in R7R5, this instruction is ignored.

*2. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.

[Description]

- This instruction moves string in successively address incrementing direction from the source address specified by A0 to the fixed destination address specified by A1 as many as the number specified by R7R5.
- Set the source address in A0, the destination address in A1, and the number of moves in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- The address register A0 on the instruction completion indicates the next sequential address of the last data transferred.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

SOUT.W

SSTR

Store Strings

SSTR

[Instruction Syntax]

SSTR.size

└──── B , W , L , Q

[Instruction Code/Number of Cycles]

Page=274

[Operation]

```
while (R7R5 != 0) *1 {
        *A1     =   R0L/R0/R2R0/R3R1R2R0*2;
        A1      =   A1 + n *3;
        R7R5    =   R7R5 - 1;
}
```

Notes:

*1. When the value 0 is set in R7R5, this instruction is ignored.

*2. When size specifier (.size) is ".B", A1 = R0L.
When size specifier (.size) is ".W", A1 = R0.
When size specifier (.size) is ".L", A1 = R2R0.
When size specifier (.size) is ".Q", A1 = R3R1R2R0.

*3. When size specifier (.size) is ".B", n = 1.
When size specifier (.size) is ".W", n = 2.
When size specifier (.size) is ".L", n = 4.
When size specifier (.size) is ".Q", n = 8.

[Description]

- This instruction stores the contents of R0L/R0/R2R0/R3R1R2R0 in successively address incrementing direction to the destination address specified by A1 as many as the number specified by R7R5.
- Set the destination address in A1 and the number of moves in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- The address register A1 on the instruction completion indicates the next sequential address of the last data written.
- An interrupt request during instruction execution is accepted after one data is completely moved.

[Flags Affected]

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

[Sample Description]

```
SSTR.W
```

STC

Store from Control Register

STC

[Instruction Syntax]

STC src,dest

[Instruction Code/Number of Cycles]

Page=274

[Operation]

dest = src;

[Description]

- This instruction moves src to dest.

[Available src/dest]

src				dest *1			
SB	FB	FLG	SP	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
ISP	SVF	SVP	INTB	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
DSA0	DSA1	DSA2	DSA3		dsp:8[FB]	dsp:16[FB]	dsp:24
DDA0	DDA1	DDA2	DDA3		dsp:16	dsp:16[SB]	dsp:24[SB]
DCT0	DCT1	DCT2	DCT3	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
DSR0	DSR1	DSR2	DSR3	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
DDR0	DDR1	DDR2	DDR3	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
DCR0	DCR1	DCR2	DCR3	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
DMD0	DMD1	DMD2	DMD3				
VCT							

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
STC     SB,R2R0
STC     FLG,[A0]
STC     DMD2,A1
STC     DCT1,mem[A2]
```

STCTX

Store Context

STCTX

[Instruction Syntax]

STCTX src,dest

[Instruction Code/Number of Cycles]

Page=275

[Operation]

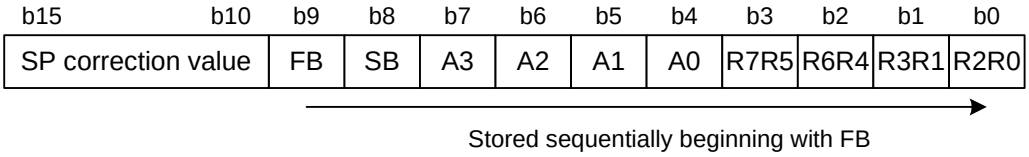
```
for (i=0 ; i< n *1 ; i++) {
        SP   =    SP - 4;
        *SP  =    registers(dest[src]);
}
```

Note:

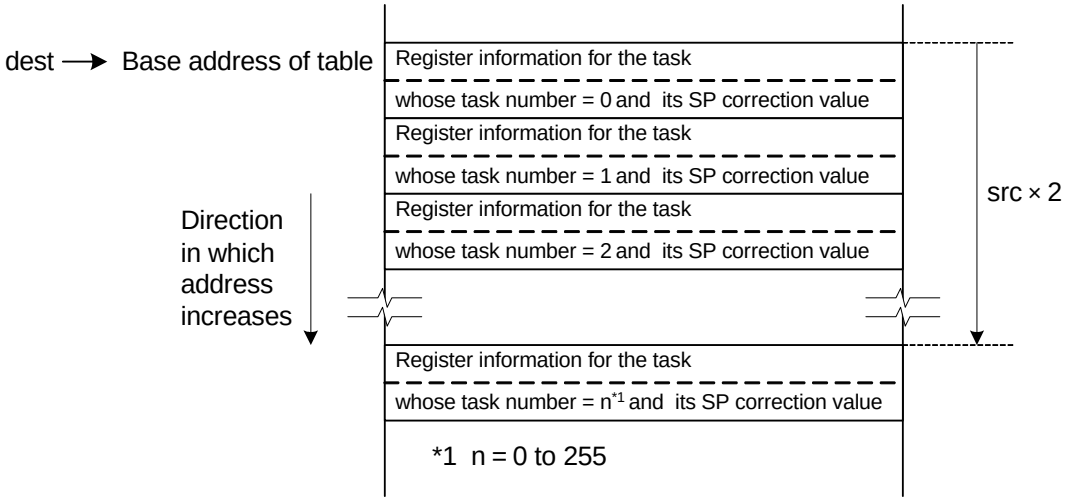
*1. n is the number of registers to be stored.

[Description]

- This instruction stores task context to the stack area.
- Set the RAM address that contains the task number in src and the start address of table entries in dest.
- The required register information is specified from table entries by the task number, and each register is stored to the stack area according to the specified register information. Then the SP correction value is subtracted from the stack pointer (SP). For this SP correction value, set the total bytes of registers to be stored. Do not set any table entries to 0000h.
- Information on registers is configured as shown below. 1 indicates a register to be stored and 0 indicates a register that is not to be stored.



- A table is configures as shown in the following page. The base address of the table is specified by dest. Each table entry is 16-bit data composed of a register information form bit 0 to bit 9 and a stack pointer correction value from bit 10 to bit 15 (Refer to the figure above). This word data is stored at an address in which the content of src is duplicated.



[Available src/dest]

src	dest
abs:16	dsp:24

[Flags Affected]

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

[Sample Description]

```
STCTX    ram,rom_tbl
```

STNZ

Store on Not Zero

STNZ

[Instruction Syntax]

STNZ.size src,dest

size: B , W , L

[Instruction Code/Number of Cycles]

Page=276

[Operation]

```
if (Z==0)
        dest =  src;
```

[Description]

- This instruction moves src to dest when the Z flag is 0. dest is not changed when the Z flag is 1.

[Available src/dest]

src				dest *1			
~~R0L/R0/R2R0~~	~~R0H/R2/R3R1~~	~~R2L/R1/R6R4~~	~~R2H/R3/R7R5~~	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
~~R1L/R4/A0~~	~~R1H/R6/A1~~	~~R3L/R5/A2~~	~~R3H/R7/A3~~	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	~~dsp:8[FB]~~	~~dsp:16[FB]~~	~~dsp:24~~		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	~~dsp:16~~	~~dsp:16[SB]~~	~~dsp:24[SB]~~		dsp:16	dsp:16[SB]	dsp:24[SB]
~~[A0]~~	~~dsp:8[A0]~~	~~dsp:16[A0]~~	~~dsp:24[A0]~~	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
~~[A1]~~	~~dsp:8[A1]~~	~~dsp:16[A1]~~	~~dsp:24[A1]~~	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
~~[A2]~~	~~dsp:8[A2]~~	~~dsp:16[A2]~~	~~dsp:24[A2]~~	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
~~[A3]~~	~~dsp:8[A3]~~	~~dsp:16[A3]~~	~~dsp:24[A3]~~	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
STNZ.B      #1,R0L
STNZ.W      #5,[A0]
STNZ.L      #1000h,R6R4
```

STOP

Stop

STOP

[Instruction Syntax]

STOP

[Instruction Code/Number of Cycles]

Page=276

[Operation]

[Description]

- This instruction stops the program execution. The execution resumes when an interrupt with a higher request level than that of the interrupt priority level for wake-up select bit is accepted, or when a reset is generated.

[Flags Affected]

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

[Sample Description]

STOP

STZ

Store on Zero

STZ

[Instruction Syntax]

STZ.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=277

[Operation]

```
if (Z==1)
        dest =    src;
```

[Description]

- This instruction moves src to dest when the Z Flag is 1. dest is not changed when the Z flag is 0.

[Available src/dest]

src				dest *1			
~~R0L/R0/R2R0~~	~~R0H/R2/R3R1~~	~~R2L/R1/R6R4~~	~~R2H/R3/R7R5~~	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
~~R1L/R4/A0~~	~~R1H/R6/A1~~	~~R3L/R5/A2~~	~~R3H/R7/A3~~	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	~~dsp:8[FB]~~	~~dsp:16[FB]~~	~~dsp:24~~		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	~~dsp:16~~	~~dsp:16[SB]~~	~~dsp:24[SB]~~		dsp:16	dsp:16[SB]	dsp:24[SB]
~~[A0]~~	~~dsp:8[A0]~~	~~dsp:16[A0]~~	~~dsp:24[A0]~~	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
~~[A1]~~	~~dsp:8[A1]~~	~~dsp:16[A1]~~	~~dsp:24[A1]~~	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
~~[A2]~~	~~dsp:8[A2]~~	~~dsp:16[A2]~~	~~dsp:24[A2]~~	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
~~[A3]~~	~~dsp:8[A3]~~	~~dsp:16[A3]~~	~~dsp:24[A3]~~	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
STZ.B       #1,R0L
STZ.W       #5,[A0]
STZ.L       #1000h,R6R4
```

STZX

Store according to Zero Flag

STZX

[Instruction Syntax]

STZX.size src1,src2,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=277

[Operation]

```
if (Z==1)
        dest =  src1;
else
        dest =  src2;
```

[Description]

- This instruction moves src1 to dest when the Z flag is 1. It moves src2 to dest when the flag is 0.

[Available src/dest]

src1 src2				dest *1			
~~R0L/R0/R2R0~~	~~R0H/R2/R3R1~~	~~R2L/R1/R6R4~~	~~R2H/R3/R7R5~~	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
~~R1L/R4/A0~~	~~R1H/R6/A1~~	~~R3L/R5/A2~~	~~R3H/R7/A3~~	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
~~#IMMEX~~	~~dsp:8[FB]~~	~~dsp:16[FB]~~	~~dsp:24~~		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	~~dsp:16~~	~~dsp:16[SB]~~	~~dsp:24[SB]~~		dsp:16	dsp:16[SB]	dsp:24[SB]
~~[A0]~~	~~dsp:8[A0]~~	~~dsp:16[A0]~~	~~dsp:24[A0]~~	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
~~[A1]~~	~~dsp:8[A1]~~	~~dsp:16[A1]~~	~~dsp:24[A1]~~	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
~~[A2]~~	~~dsp:8[A2]~~	~~dsp:16[A2]~~	~~dsp:24[A2]~~	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
~~[A3]~~	~~dsp:8[A3]~~	~~dsp:16[A3]~~	~~dsp:24[A3]~~	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
STZX.B      #1,#2,R0L
STZX.W      #5,#10,[A0]
STZX.L      #1000h,#4000h,R6R4
```

SUB

Subtract

SUB

[Instruction Syntax]

SUB.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=278

[Operation]

dest = dest - src;

[Description]

- This instruction subtracts src from dest and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. The flag becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag becomes 1 when an operation as unsigned integers results in equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
SUB.B   #2,R0L
SUB.W   R0,R2
SUB.L   A0,R7R5
SUB.B   R3L,[A0]
SUB.W   [A1],R4
SUB.L   R2R0,[A3]
```

SUBF

Subtract Floating-point

SUBF

[Instruction Syntax]

SUBF src,dest

[Instruction Code/Number of Cycles]

Page=280

[Operation]

dest = dest - src;

[Description]

- This instruction subtract src from dest and stores the result to dest.
- The operation result is rounded according to the rounding mode specified in the flag register (FLG).
- When the result is overflowed, it takes either the maximum positive normal number (7F7FFFFFh) or the maximum negative normal number (FF7FFFFFh) depending on whether signed or not.
- When the result is underflowed, it takes any of the following according to the rounding mode: zero (00000000h), the minimum positive number (00800000h) or the minimum negative normal number (80800000h).
- The result for invalid numbers is undefined.

[Available src/dest]

src *1				dest *1			
~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5	~~R0L/R0~~/R2R0	~~R0H/R2~~/R3R1	~~R2L/R1~~/R6R4	~~R2H/R3~~/R7R5
~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3	~~R1L/R4~~/A0	~~R1H/R6~~/A1	~~R3L/R5~~/A2	~~R3H/R7~~/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	FO	FU	U	I	O	B	S	Z	D	C
Status	✓	✓	–	–	✓	–	✓	✓	–	✓

Description

FO : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

FU : The flag becomes 1 when the operand has invalid numbers or when the operation results in an underflow; otherwise it becomes 0.

O : The flag becomes 1 when the operand has invalid numbers or when the operation results in an overflow; otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

C : The flag is undefined.

[Sample Description]

```
SUBF    R2R0,R3R1
SUBF    [A1],R2R0
SUBF    mem[FB],R3R1
```

SUNTIL

Search Equal String

SUNTIL

[Instruction Syntax]

SUNTIL.size

B , W , L

[Instruction Code/Number of Cycles]

Page=281

[Operation]

```
while (*A0 != R0L/R0/R2R0 *1 && R7R5 != 0 *2) {
        A0++;*3
        R7R5 = R7R5 - 1;
}
```

Notes:

*1. When size specifier (.size) is ".B", A0 !=R0L.
When size specifier (.size) is ".W", A0 !=R0.
When size specifier (.size) is ".L", A0 !=R2R0.

*2. When the value 0 is set in R7R5, the comparison operation is performed only once. The value of A0 and flags are undefined on the instruction completion.

*3. A0 is increased by 1, 2 and 4 in size specifier (.size) as ".B", ".W" and "L", respectively.

[Description]

- This instruction searches string in successively address incrementing direction from the comparison address specified by A0 to the compared address as many as the number specified by R7R5 until the data content matches that of R0L/R0/R2R0.
- Set the comparison address in A0 and the number of comparison in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- Flags vary according to the operation result of “*A0-R0L/R0/R2R0”.
- The address register A0 on the instruction completion indicates the address of matching data. When no matched data is found, it indicates the address of following data.
- The result of which “the number of comparison operation is subtracted from the initial value” is set in R7R5 when the instruction is completed.
- An interrupt request during instruction execution is accepted after one data comparison is completed.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. The flag becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when matched data is found; otherwise it becomes 0.

C : The flag becomes 1 when an operation as unsigned integers results in equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

SUNTIL.W

SWHILE

Search Unequal String

SWHILE

[Instruction Syntax]

SWHILE.size

B , W , L

[Instruction Code/Number of Cycles]

Page=282

[Operation]

```
while (*A0 == R0L/R0/R2R0 *1 && R7R5 != 0 *2) {
        A0++;*3
        R7R5 = R7R5 - 1;
}
```

Notes:

*1. When size specifier (.size) is ".B", A0=R0L.
When size specifier (.size) is ".W", A0=R0.
When size specifier (.size) is ".L", A0=R2R0.

*2. When the value 0 is set in R7R5, the comparison operation is performed only once. The value of A0 and flags are undefined on the instruction completion.

*3. A0 is respectively increased by 1, 2 and 4 in size specifier (.size) as ".B", ".W" and "L".

[Description]

- This instruction searches string in successively address incrementing direction from the comparison address specified by A0 to the compared address as many as the number specified by R7R5 until the data content does not match that of R0L/R0/R2R0.
- Set the comparison address in A0 and the number of comparison in R7R5.
- The maximum value to be set in R7R5 is 00FFFFFFh.
- Flags vary according to the operation result of “*A0-R0L/R0/R2R0”.
- The address register A0 on the instruction completion indicates the address of matching data. When no matched data is found, it indicates the address of following data.
- The result of which “the number of comparison operation is subtracted from the initial value” is set in R7R5 when the instruction is completed.
- An interrupt request during instruction execution is accepted after one data comparison is completed.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	✓	–	✓	✓	–	✓

Description

O : The flag indicates the overflow of an operation as signed integers. The flag becomes 1 when the operation results in exceeding -128(.B) or +127(.B), -32768(.W) or +32767(.W), or -2147483648(.L) or +2147483647(.L); otherwise it becomes 0.
S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.
Z : The flag becomes 1 when every data is matched; otherwise it becomes 0.
C : The flag becomes 1 when an operation as unsigned integers results in equal to or greater than 0; otherwise it becomes 0.

[Sample Description]

```
SWHILE.W
```

TST

Test Logical

TST

[Instruction Syntax]

TST.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=282

[Operation]

dest & src;

[Description]

- This instruction varies each flag status of the flag register (FLG) according to the result of logical AND of src and dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
TST.B   #2,R0L
TST.W   R0,R2
TST.L   A0,R7R5
TST.B   R3L,mem[A0]
TST.W   [A1],R4
TST.L   R2R0,[A3]
```

UND

Undefined Instruction Interrupt

UND

[Instruction Syntax]

UND

[Instruction Code/Number of Cycles]

Page=283

[Operation]

```
SP    =    SP - 4;
*SP   =    FLG;
SP    =    SP - 4;
*SP   =    PC + 1;
PC    =    *(long *)0xFFFFFFDC;
```

[Description]

- This instruction generates an undefined instruction interrupt.
- The undefined instruction interrupt is non-maskable.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status*1	✓	✓	–	–	–	–	✓	–

Note:

*1. The flags are saved to the stack area before the UND instruction is executed.

Description

U : The flag becomes 0.
I : The flag becomes 0.
D : The flag becomes 0.

[Sample Description]

UND

WAIT

Wait

WAIT

[Instruction Syntax]

WAIT

[Instruction Code/Number of Cycles]

Page=283

[Operation]

[Description]

- This instruction stops the program execution. The execution resumes when an interrupt with a higher request level than that of the interrupt priority level for wake-up select bit is accepted, or when a reset is generated.

[Flags Affected]

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

[Sample Description]

WAIT

XCHG

Exchange

XCHG

[Instruction Syntax]

XCHG.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=284

[Operation]

```
tmp0 =  src;
src  =  dest;
dest =  tmp0;

tmp0:  temporary registers
```

[Description]

- This instruction exchanges contents between src and dest.

[Available src/dest]

src *1				dest *2			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
					dsp:8[FB]	dsp:16[FB]	dsp:24
					dsp:16	dsp:16[SB]	dsp:24[SB]
				[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
				[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
				[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
				[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Notes:

*1. Bank1 register direct addressing is available.

*2. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

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

[Sample Description]

```
XCHG.B    R0H,R0L
XCHG.W    R0,R2
XCHG.L    A0,R7R5
XCHG.B    R3L,[A0]
XCHG.W    [A1],R4
XCHG.L    R2R0,[A3]
```

XOR

Exclusive Or Logical

XOR

[Instruction Syntax]

XOR.size src,dest

B , W , L

[Instruction Code/Number of Cycles]

Page=284

[Operation]

dest = dest ^ src;

[Description]

• This instruction exclusive ORs src and dest and stores the result to dest.

[Available src/dest]

src *1				dest *1			
R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5	R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3	R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
#IMMEX	dsp:8[FB]	dsp:16[FB]	dsp:24		dsp:8[FB]	dsp:16[FB]	dsp:24
#IMM	dsp:16	dsp:16[SB]	dsp:24[SB]		dsp:16	dsp:16[SB]	dsp:24[SB]
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:

*1. Indirect instruction addressing and Bank1 register direct addressing are available.

[Flags Affected]

Flag	U	I	O	B	S	Z	D	C
Status	–	–	–	–	✓	✓	–	–

Description

S : The flag becomes 1 when the operation results in MSB = 1; otherwise it becomes 0.

Z : The flag becomes 1 when the operation results in 0; otherwise it becomes 0.

[Sample Description]

```
XOR.B    #2,R0L
XOR.W    R0,R2
XOR.L    A0,R7R5
XOR.B    R3L,mem[A0]
XOR.W    [A1],R4
XOR.L    R2R0,[A3]
```

3.3 INDEX Instruction

This section explains each INDEX instruction individually.
The INDEX instruction is provided for use in arrays. The instruction adds the content of src of the current INDEX to the contents of src and dest of the next sequential instruction to obtain an effective address. All the operands and the result are interpreted as unsigned integers.
The 4GB from 0 to FFFFFFFFh can be modified. The src range for the INDEX instruction varies as below depending on the size specifier of the next sequential instruction to be executed:

Size Specifier of the Next Instruction to be Executed	Size Specifier of INDEX Instruction	src Range	Values to be Added
.B	.B	0 to FFh	src contents
	.W	0 to FFFFh	
	.L	0 to FFFFFFFFh	
.W	.B	0 to FFh	src contents x 2
	.W	0 to FFFFh	
	.L	0 to 7FFFFFFFh	
.L	.B	0 to FFh	src contents x 4
	.W	0 to FFFFh	
	.L	0 to 3FFFFFFFh	

An interrupt request cannot be accepted directly after an INDEX instruction.
Four kinds of INDEX instructions are as shown below:

INDEXB
INDEX1
INDEX2
BITINDEX

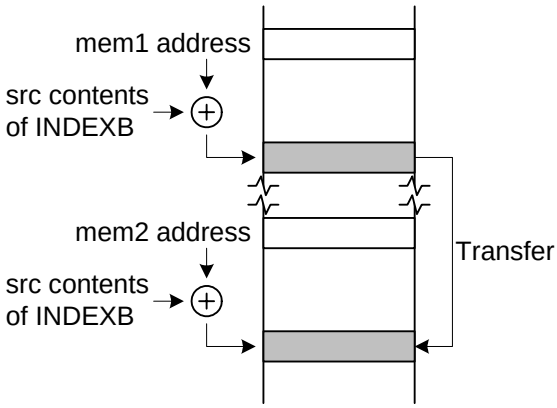
3.3.1 INDEXB.size src

The INDEXB (Index Both Operands) instruction adds the content of the src of the current INDEXB to the contents of the src and dest of the next sequential instruction to obtain an effective address. All operands and the result are interpreted as unsigned integers.
Specify the addressing to access memory for the src and dest of the next instruction of INDEXB.

Example 1: When ".B" is specified as the size specifier of the next instruction to be executed,

```
INDEXB.W    src
MOV.B:G     mem1, mem2
```

Memory



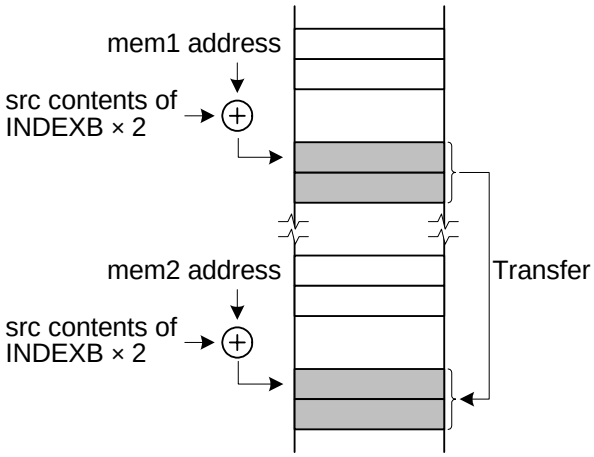
Operation in C language

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

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

Example 2: When ".W" is specified as the size specifier of the next instruction to be executed

```
INDEXB.W    src
MOV.W:G     mem1, mem2
                \  /
               Memory
```



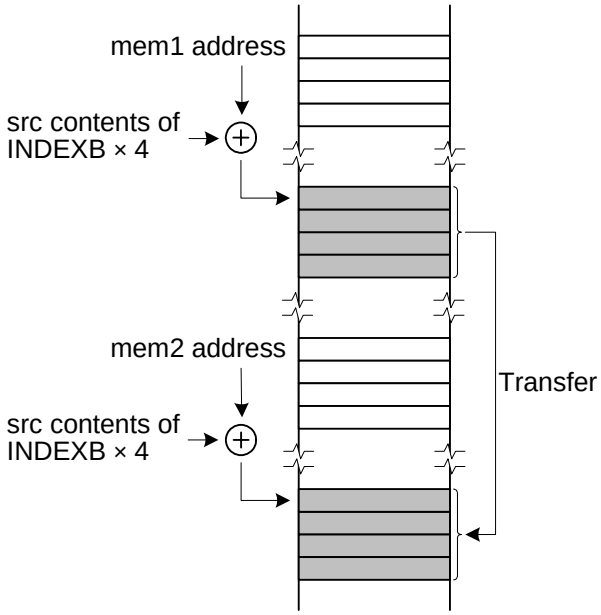
Operation in C language

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

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

Example 3: When ".L" is specified as the size specifier of the next instruction to be executed

```
INDEXB.W    src
MOV.L:G     mem1, mem2
                \  /
               Memory
```



Operation in C language

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

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

Instructions modified by INDEXB

The src and dest of ADC, ADD:G *1,*2, ADDF, AND:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EXTS *3, EXTZ:G *1,*3, MAX, MIN, MOV:G *1,*4, MUL, MULF, MULU, OR, ROUND, SBB, SUB, SUBF, TST and XOR

Notes:

*1. Only the G format can be specified.
*2. dsp:8[SP] cannot be used for the dest of an ADD instruction.
*3. src contents of INDEXB multiplied by n (n = 1, 2, or 4) depending on the size before the sign extended/zero extended for src (mem1) and the size after the sign extended/zero extended for dest (mem2).
*4. dsp:8[SP] cannot be used for the src or dest of a MOV instruction.

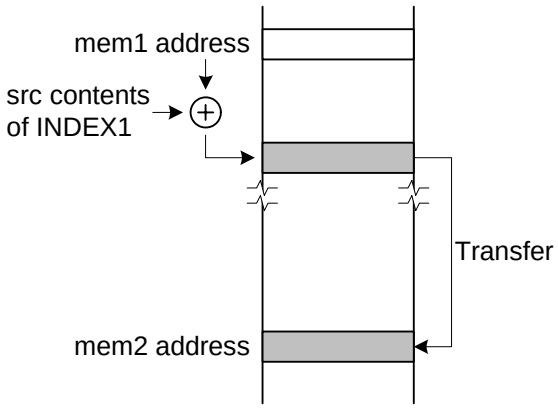
For instructions following the INDEXB instruction, use only those shown above.

3.3.2 INDEX1.size src

The INDEX1 (Index 1st Operand) instruction adds the content of the src of the current INDEX1 to the contents of the first operand of the next sequential instruction to obtain an effective address. All operands and the result are interpreted as unsigned integers.
Specify the addressing to access memory for the first operand of the next instruction of INDEX1.

Example 1: When ".B" is specified as the size specifier of the next instruction to be executed

```
INDEX1.W    src
MOV.B:G     mem1, mem2
              \
            Memory
```



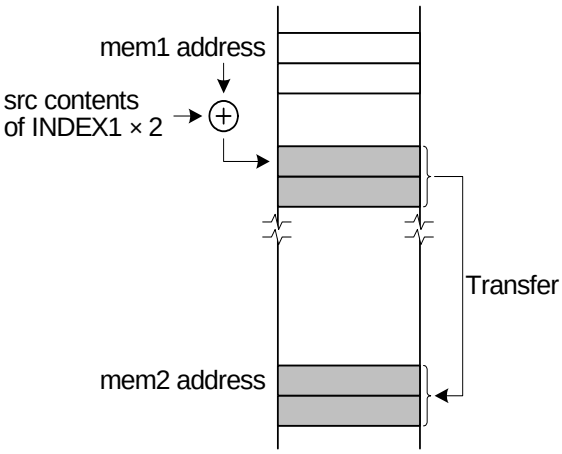
Operation in C language

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

mem2 = mem1[src];
```

Example 2: When ".W" is specified as the size specifier of the next instruction to be executed

```
INDEX1.W    src
MOV.W:G     mem1, mem2
              \
            Memory
```



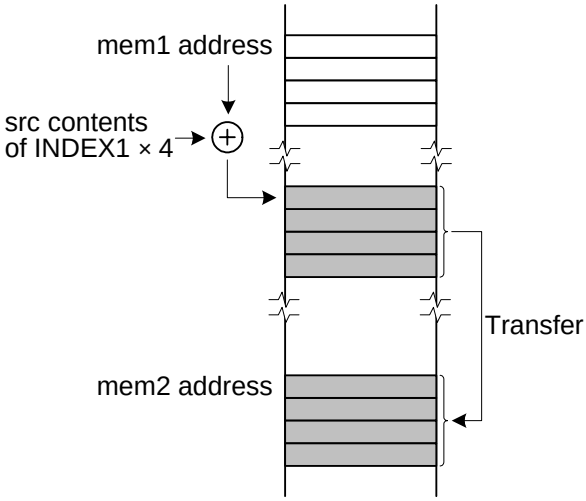
Operation in C language

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

mem2 = mem1[src];
```

Example 3: When ".L" is specified as the size specifier of the next instruction to be executed

```
INDEX1.W    src
MOV.L:G     mem1, mem2
              |
            Memory
```



Operation in C language

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

mem2 = mem1[src];
```

Instructions modified by INDEX1

The src of ADC, ADD:G *1,*2, ADDF, AND:G *1, BTST:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EDIV, EDIVU, EDIVX, EMUL, EMULU, EXTS*3, EXTZ*3, JMPI, JSRI, LDC, MAX, MIN, MOV:G *1,*4, MOV:S *5, MOV*Dir* *6, MUL, MULF, MULU, OR, PUSH, ROUND, SBB, SUB, SUBF, TST and XOR

The dest of ABS, ADCF, ADD:Q*2, ADD:S, ADSF, AND:S, BCLR, BM*Cnd*, BNOT, BSET, BTSTC, BTSTS, CLIP, CMP:Q, CMP:S, DEC, INC, MOV:G *7, MOV:Q, MOV:S *8, MOV:Z, MOV*Dir* *9, NEG, NOT, POP, ROLC, RORC, ROT, SC*Cnd*, SHA, SHL, STC, STNZ, STZ, STZX and XCHG

Notes:

*1. Only the G format can be specified.
*2. SP cannot be used for the dest of an ADD instruction.
*3. src contents of INDEX1 multiplied by n (n = 1, 2, or 4) depend on the size before sign extended/ zero extended.
*4. dsp:8[SP] cannot be used for the src of a MOV instruction.
*5. MOV:S src,R2R0/R0/R0L or MOV:S.L src, A0 is enabled.
*6. MOV*Dir* src, R0L is enabled. “.B” is designated as the size specifier.
*7. MOV:G dsp:8[SP], dest is enabled.
*8. MOV:S R2R0/R0/R0L, dest or MOV:S #IMM, dest is enabled.
*9. MOV*Dir* R0L, dest is enabled. “.B” is designated as the size specifier.

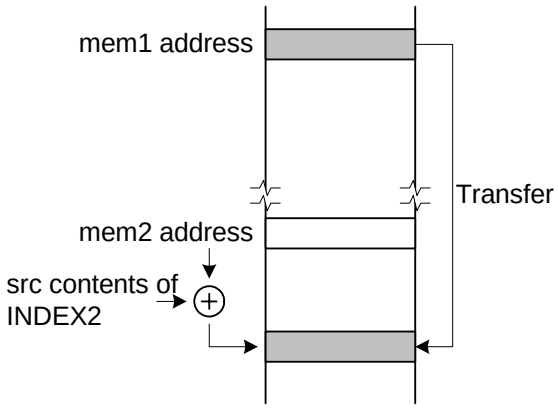
For instructions following the INDEX1 instruction, use only those shown above.

3.3.3 INDEX2.size src

The INDEX2 (Index 2nd Operand) instruction adds the content of the src of the current INDEX2 to the contents of the first operand of the next sequential instruction to obtain an effective address. All operands and the result are interpreted as unsigned integers.
Specify the addressing to access memory for the dest of the next instruction of INDEX2.

Example 1: When ".B" is specified as the size specifier of the next instruction to be executed

```
INDEX2.W    src
MOV.B:G     mem1, mem2
                  |
                Memory
```



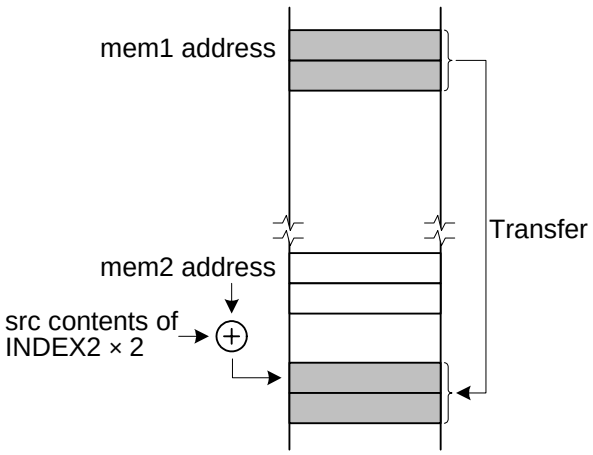
Operation in C language

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

mem2[src] = mem1;
```

Example 2: When ".W" is specified as the size specifier of the next instruction to be executed

```
INDEX2.W    src
MOV.W:G     mem1, mem2
                  |
                Memory
```



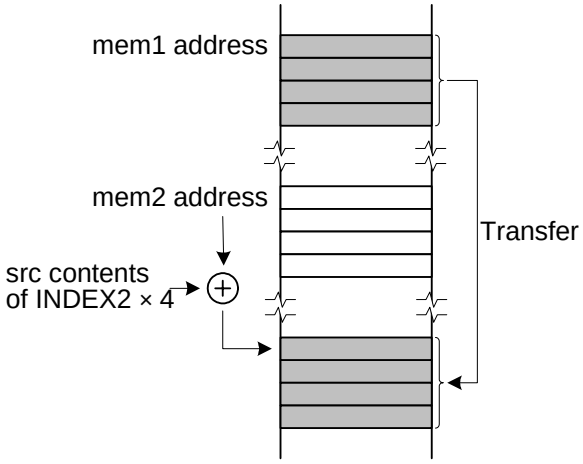
Operation in C language

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

mem2[src] = mem1;
```

Example 3: When ".L" is specified as the size specifier of the next instruction to be executed

```
INDEX2.W    src
MOV.L:G     mem1, mem2
                  |
               Memory
```



Operation in C language

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

mem2[src] = mem1;
```

Instructions to be modified by INDEX2

The dest of ADC, ADD:G *1,*2, ADDF, AND:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EXTS *3, EXTZ:G *1,*3, MAX, MIN, MOV:G *1,*4, MUL, MULF, MULU, OR, ROUND, SBB, SUB, SUBF, TST and XOR

Notes:

*1. Only the G format can be specified.
*2. SP cannot be used for the dest of ADD instruction.
*3. src contents of INDEX2 multiplied by n (n = 1, 2, or 4) depend on the size after sign extended/zero extended.
*4. dsp:8[SP] cannot be used for the src and dest of a MOV instruction.

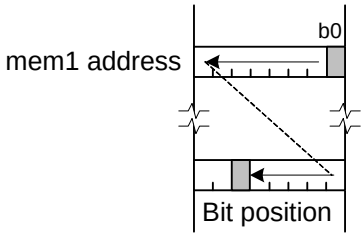
For instructions following the INDEX2 instruction, use only those shown above.

3.3.4 BITINDEX.size src

BITINDEX instruction (Bit Index) specifies a bit position of the src or dest of the next instruction to be executed by the src of BITINDEX.
Specify a bit instruction for the next sequential instruction of BITINDEX and the addressing to access memory for the src or dest of the next instruction.

Example:

```
BITINDEX.B/W    src
BSET            3, mem1
   \             /        \
Bit instruction  invalid    Memory
```



Instructions modified by BITINDEX

The src of BTST:G *1
The dest of BCLR, BM*Cnd*, BNOT, BSET, BTSTC and BTSTS

Note:

*1. Only the G format can be specified.

For instructions following the BITINDEX instruction, use only those shown above.

3.3.5 Enabled Instruction List to Be Executed Next to INDEX Instruction

The following table lists enabled instructions to be executed next to each INDEX instruction.

INDEX Instruction	Enabled Instructions	
INDEXB.B/W/L	src and dest of the following instructions: ADC, ADD:G *1,*2, ADDF, AND:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EXTS, EXTZ:G *1, MAX, MIN, MOV:G *1,*3, MUL, MULF, MULU, OR, ROUND, SBB, SUB, SUBF, TST, XOR	
INDEX1.B/W/L	src of the following instructions: ADC, ADD:G *1,*2, ADDF, AND:G *1, BTST:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EDIV, EDIVU, EDIVX, EMUL, EMULU, EXTS, EXTZ, JMPI, JSRI, LDC, MAX, MIN, MOV:G *1,*4, MOV:S *5, MOV*Dir* *6, MUL, MULF, MULU, OR, PUSH, ROUND, SBB, SUB, SUBF, TST, XOR	dest of the following instructions: ABS, ADCF, ADD:Q *2, ADD:S, ADSF, AND:S, BCLR, BM*Cnd*, BNOT, BSET, BTSTC, BTSTS, CLIP, CMP:Q, CMP:S, DEC, INC, MOV:G *7, MOV:Q, MOV:S *8, MOV:Z, MOV*Dir* *9, NEG, NOT, POP, ROLC, RORC, ROT, SC*Cnd*, SHA, SHL, STC, STNZ, STZ, STZX, XCHG
INDEX2.B/W/L	dest of the following instructions: ADC, ADD:G *1,*2, ADDF, AND:G *1, CMP:G *1, CMPF, CNVIF, DADC, DADD, DIV, DIVF, DIVU, DIVX, DSBB, DSUB, EXTS, EXTZ:G *1, MAX, MIN, MOV:G *1,*3, MUL, MULF, MULU, OR, ROUND, SBB, SUB, SUBF, TST, XOR	
BITINDEX.B/W/L	src of the following instruction: BTST:G *1	dest of the following instructions: BCLR, BM*Cnd*, BNOT, BSET, BTSTC, BTSTS

Note:

*1. Only the G format can be specified.
*2. SP cannot be used for dest of an ADD instruction.
*3. dsp:8[SP] cannot be used for the src or dest of a MOV instruction.
*4. dsp:8[SP] cannot be used for the src of a MOV instruction.
*5. MOV:S src, R0L/R0/R2R0 or MOV:S.L src, A0 is enabled.
*6. MOV*Dir* src, R0L is enabled.
*7. MOV:G dsp:8[SP], dest is enabled.
*8. MOV:S R0L/R0/R2R0, dest or MOV:S #IMM, dest is enabled.
*9. MOV*Dir* R0L, dest is enabled.

3.3.6 Addressing Mode

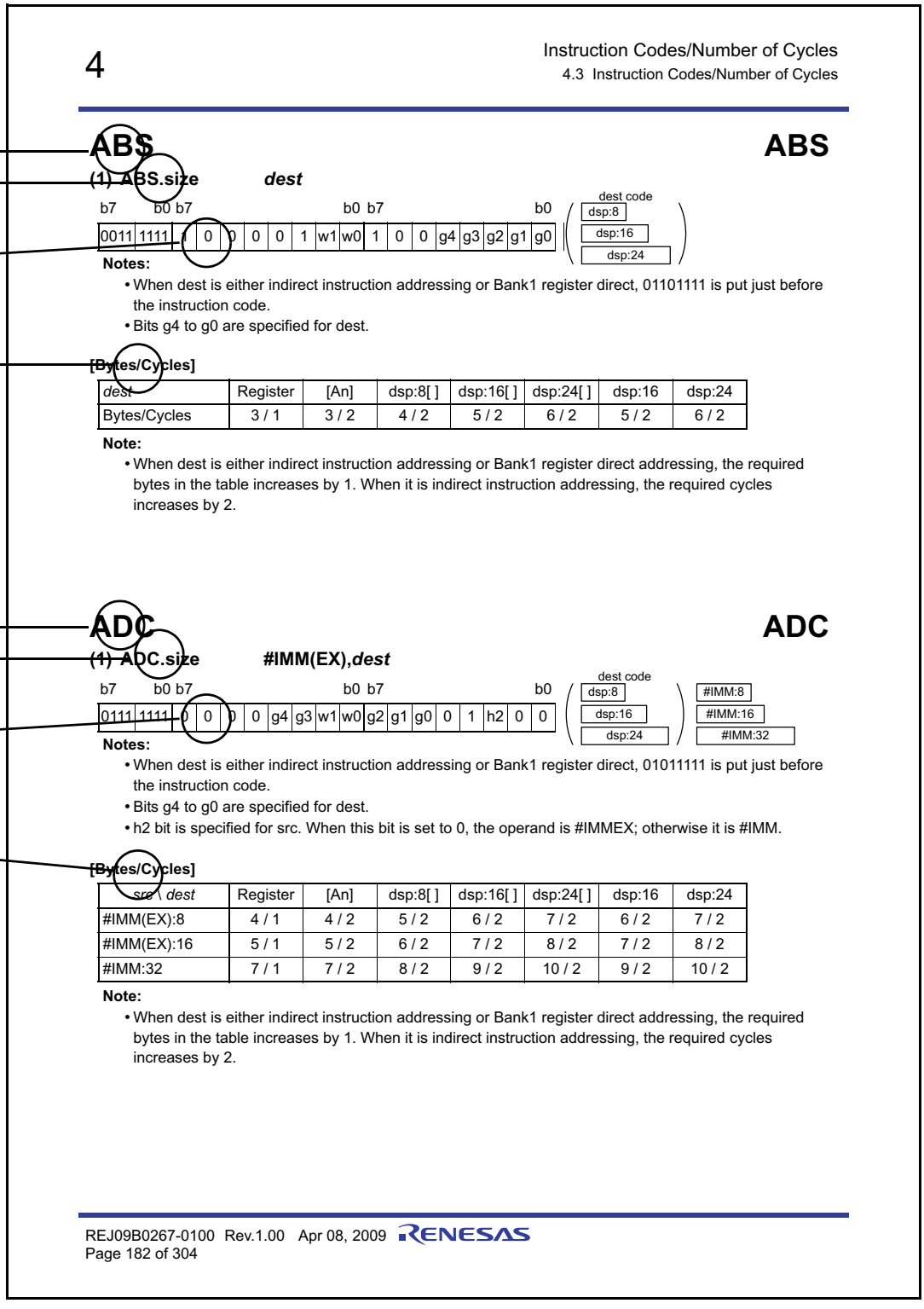
The following table lists the enabled addressing modes for the instructions to be executed after the INDEX instruction. The indirect instruction addressing mode can be used in each instruction.

src				dest			
[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]	[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]	[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]	[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]	[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]
	dsp:8[FB]	dsp:16[FB]			dsp:8[FB]	dsp:16[FB]	
	dsp:8[SB]	dsp:16[SB]	dsp:24[SB]		dsp:8[SB]	dsp:16[SB]	dsp:24[SB]
		dsp:16	dsp:24			dsp:16	dsp:24

4. Instruction Codes/Number of Cycles

4.1 Guide to This Chapter

This chapter describes the instruction code and number of cycles for each opcode.
The following sample shows how to read this chapter.



4

Instruction Codes/Number of Cycles
4.3 Instruction Codes/Number of Cycles

(1) (2) **ABS** **ABS**

(1) ABS.size ***dest***

b7 b0 b7 b0 b7 b0

0011	1111	1	0	0	0	0	1	w1	w0	1	0	0	g4	g3	g2	g1	g0

dest code: dsp:8 / dsp:16 / dsp:24

(3)

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

(4) **[Bytes/Cycles]**

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

(1) (2) **ADC** **ADC**

(1) ADC.size **#IMM(EX),*dest***

b7 b0 b7 b0 b7 b0

0111	1111	0	0	0	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code: dsp:8 / dsp:16 / dsp:24 — #IMM:8 / #IMM:16 / #IMM:32

(3)

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

(4) **[Bytes/Cycles]**

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM(EX):16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
#IMM:32	7 / 1	7 / 2	8 / 2	9 / 2	10 / 2	9 / 2	10 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

REJ09B0267-0100 Rev.1.00 Apr 08, 2009
Page 182 of 304

(1) Mnemonic

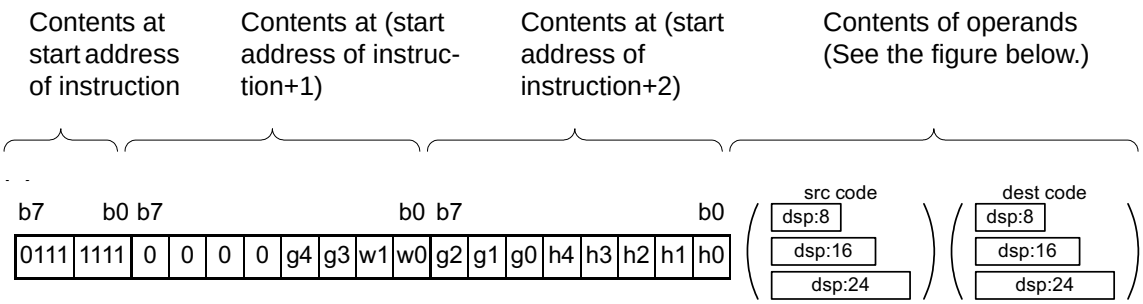
Indicates the instruction mnemonic explained.

(2) Syntax

Indicates the instruction syntax using symbols.

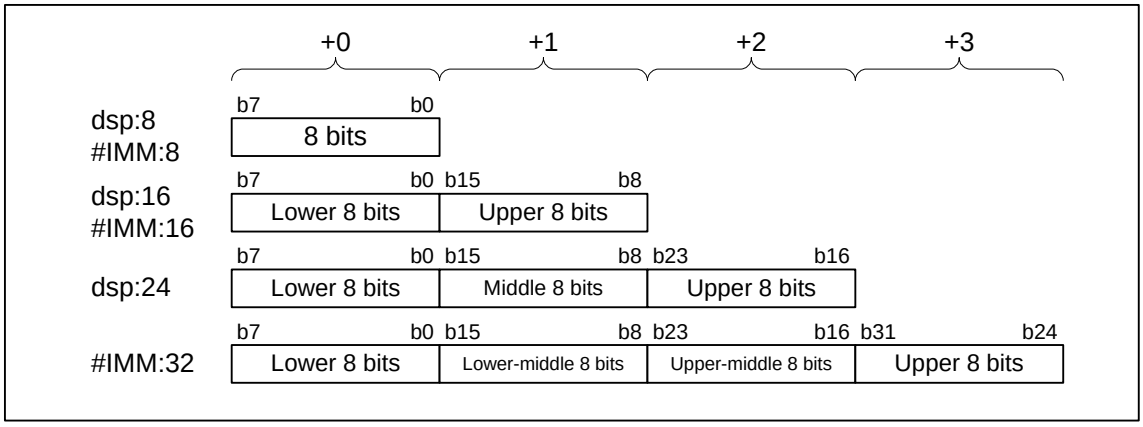
(3) Instruction code

Indicates the instruction code. Contents in brackets () may be omitted depending on selected src/dest.



Refer to **4.2, "Addressing"** for details on the contents of bits w1, w0, g4 to g0, and h4 to h0 bits.

The contents of operands (4th byte or later in the example above) are aligned as shown below:



(4) Number of Bytes/Number of Cycles Table

Indicates the number of bytes (on the left of the slash) and the minimum number of cycles required for an instruction (on the right of the slash).
The number of cycles shown is the minimum possible, and it may increase depending on the following factors:

- Instruction queue buffer status
- External bus width
- Presence of wait cycle/no wait

4.2 Addressing

4.2.1 Specifying Operation Length

The operation length is specified by bits w1 and w0 in an instruction code.

Table 4.1 Specified Operation Length

w1	w0	Operation length*1	#IMMEX operand and operation length
0	0	Byte	8 bits immediate, word
0	1	Word	8 bits immediate, long word
1	0	Long word	16 bit immediate, long word
1	1	Reserved	Reserved

Note:
*1. When src is #IMMEX in a two-operand instruction, bits w1 and w0 follow the #IMMEX rule.

4.2.2 Specifying the Operand

Selecting an operand varies with the instruction and the instruction format.
The tables below show the methods of specifying operands in each addressing mode.

(1) General instruction addressing, generic format

The operands gen1 and gen2 used in generic format are specified in bits g4 to g0 and bits h4 to h0, respectively.

Table 4.2 Specification of Operand in Generic Format

g4 / h4	g3 / h3	g2 / h2		gen1 / gen2			
			g1 / h1	0	0	1	1
			g0 / h0	0	1	0	1
0	0	0		R0L/R0/R2R0	R0H/R2/R3R1	R2L/R1/R6R4	R2H/R3/R7R5
0	0	1		R1L/R4/A0	R1H/R6/A1	R3L/R5/A2	R3H/R7/A3
0	1	0		#IMMEX*1	dsp:8[FB]	dsp:16[FB]	dsp:24
0	1	1		#IMM*1	dsp:16	dsp:16[SB]	dsp:24[SB]
1	0	0		[A0]	dsp:8[A0]	dsp:16[A0]	dsp:24[A0]
1	0	1		[A1]	dsp:8[A1]	dsp:16[A1]	dsp:24[A1]
1	1	0		[A2]	dsp:8[A2]	dsp:16[A2]	dsp:24[A2]
1	1	1		[A3]	dsp:8[A3]	dsp:16[A3]	dsp:24[A3]

Note:
*1. #IMMEX and #IMM cannot be specified as dest.

(2) General instruction addressing, Quick format

The operand #IMM:4 used in the quick format is specified in bits q3 to q0.

.

Table 4.3 Specification of Operand in Quick Format (1)

q3	q2	q1	q0	IMM:4	q3	q2	q1	q0	IMM:4
0	0	0	0	0	1	0	0	0	-8
0	0	0	1	1	1	0	0	1	-7
0	0	1	0	2	1	0	1	0	-6
0	0	1	1	3	1	0	1	1	-5
0	1	0	0	4	1	1	0	0	-4
0	1	0	1	5	1	1	0	1	-3
0	1	1	0	6	1	1	1	0	-2
0	1	1	1	7	1	1	1	1	-1

The operand #IMM:3 used in quick format of ADD instruction is specified in bits q2 to q0.

Table 4.4 Specification of Operand in Quick Format (2)

q2	q1	q0	IMM:3
0	0	0	Reserved
0	0	1	4
0	1	0	8
0	1	1	Reserved
1	0	0	12
1	0	1	16
1	1	0	20
1	1	1	Reserved

(3) General instruction addressing, Short format

The operand gen0 used in short format is specified in s1 and s0 bits.

Table 4.5 Operand Specification in Short Format

s1	s0	gen0
0	0	R0L/R0/R2R0 (R0H/R2/R3R1) *1
0	1	dsp:16
1	0	dsp:8[FB]
1	1	dsp:8[SB]

Note:

*1. R0H/R2/R3R1 is only used for MOV:S src, R0L/R0/R2R0.

(4) Register direct addressing

The operand greg is specified in bits q2 to q0. It is used as dest for instructions that are only available in register direct addressing mode: EDIV, EDIVX, EMUL, EMULU, EXTZ:S, and MOVA, and as src for instructions ROT, SHA, SHL, and XCHG. Tables below show what registers are used with what instructions:

Table 4.6 Specification of Operand in Instructions EDIV, EDIVU, and EDIVX

q2	q1	q0	greg
0	0	0	R0/R2R0/R3R1R2R0
0	0	1	—
0	1	0	R2/R3R1/R7R5R6R4
0	1	1	—
1	0	0	R1/R6R4/A1A0
1	0	1	—
1	1	0	R3/R7R5/A3A2
1	1	1	—

Table 4.7 Specification of Operand in Instructions EMUL and EMULU

q2	q1	q0	greg
0	0	0	R0L/R0/R2R0
0	0	1	—
0	1	0	R2L/R1/R6R4
0	1	1	—
1	0	0	R1L/R4/A0
1	0	1	—
1	1	0	R3L/R5/A2
1	1	1	—

Table 4.8 Specification of Operand in the EXTZ: S Instruction

q2	q1	q0	greg
0	0	0	—
0	0	1	—
0	1	0	—
0	1	1	—
1	0	0	A0
1	0	1	A1
1	1	0	A2
1	1	1	A3

Table 4.9 Specification of Operand in Instructions MOVA, MULX, ROT, SHA, SHL, and XCHG

q2	q1	q0	greg
0	0	0	R0L/R0/R2R0
0	0	1	R0H/R2/R3R1
0	1	0	R2L/R1/R6R4
0	1	1	R2H/R3/R7R5
1	0	0	R1L/R4/A0
1	0	1	R1H/R6/A1
1	1	0	R3L/R5/A2
1	1	1	R3H/R7/A3

(5) Control register direct addressing (CPU)

The operand creg used in control register direct addressing is specified q2 to q0 bits.

Table 4.10 Specification of Operand in Control Register Direct Addressing

q2	q1	q0	creg
0	0	0	SVP
0	0	1	INTB
0	1	0	SVF
0	1	1	FLG
1	0	0	SP
1	0	1	ISP
1	1	0	FB
1	1	1	SB

(6) Control register direct addressing (DMAC,VCT)

When a DMAC-associated control register and vector register are specified as operands, append #IMM:8 to the end of the instruction code. Specify each register in the #IMM:8 bit column.

Table 4.11 Specification of Register in Control Register Direct Addressing (1)

#IMM:8 (Hexadecimal)	Register
03h	VCT
00h to 07h (except 03h)	Channel 0 DMA register
08h to 0Fh (except 0Bh)	Channel 1 DMA register
10h to 17h (except 13h)	Channel 2 DMA register
18h to 1Fh (except 1Bh)	Channel 3 DMA register

Table 4.12 Specification of Register in Control Register Direct Addressing (2)

lower 3 bits of #IMM:8			Register	
0	0	0	DSAn*1	DMA source address register
0	0	1	DDAn*1	DMA destination address register
0	1	0	DCTn*1	DMA terminal count register
0	1	1	Reserved	
1	0	0	DSRn*1	DMA source address reload register
1	0	1	DDRn*1	DMA destination address reload register
1	1	0	DCRn*1	DMA terminal count reload register
1	1	1	DMDn*1	DMA mode register

Note:
*1. n is a channel number.

(7) Program counter relative addressing, short format

The operand dsp3 used in the short format of the program counter relative addressing is specified in bits q2 to q0.

Table 4.13 Specification of Operand in the Short Format of Program Counter Relative Addressing

q2	q1	q0	dsp3
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

(8) Bit instruction addressing

The operand bit used in bit instruction addressing is specified in b2 to b0 bits.

Table 4.14 Specification of Bit Position in Bit Instruction Addressing

b2	b1	b0	bit
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

(9) FLG direct addressing

The operand flg used in FLG direct addressing is specified in bits q2 to q0.

Table 4.15 Specification of Operand in FLG Direct Addressing

q2	q1	q0	flg
0	0	0	C
0	0	1	D
0	1	0	Z
0	1	1	S
1	0	0	B
1	0	1	O
1	1	0	I
1	1	1	U

4.2.3 Specifying Conditions

Conditions are specified in bits c3 to c0.

Table 4.16 Specification of Condition Code

c3	c2	c1	c0	Conditions
0	0	0	0	LTU / NC
0	0	0	1	LEU
0	0	1	0	NE / NZ
0	0	1	1	PZ
0	1	0	0	NO
0	1	0	1	GT
0	1	1	0	GE
0	1	1	1	Reserved

c3	c2	c1	c0	Conditions
1	0	0	0	GEU / C
1	0	0	1	GTU
1	0	1	0	EQ / Z
1	0	1	1	N
1	1	0	0	O
1	1	0	1	LE
1	1	1	0	LT
1	1	1	1	Reserved

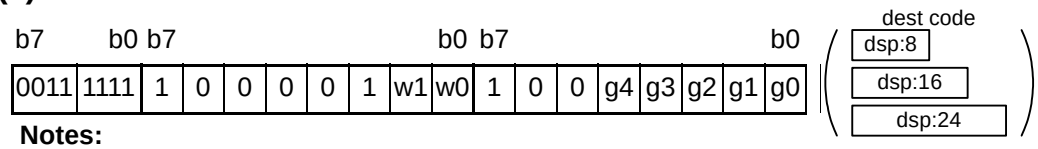
4.3 Instruction Codes/Number of Cycles

The instruction codes and number of cycles in the R32C/100 Series MCU are shown from the next page.

ABS

ABS

(1) ABS.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

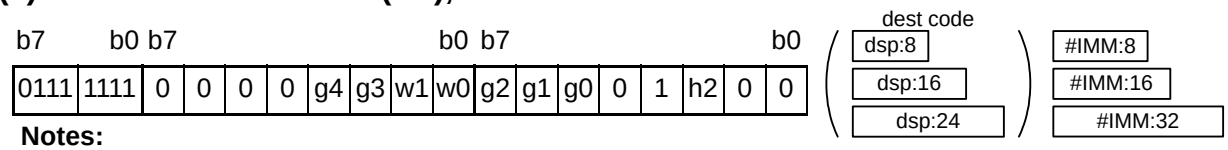
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADC

ADC

(1) ADC.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM(EX):16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
#IMM:32	7 / 1	7 / 2	8 / 2	9 / 2	10 / 2	9 / 2	10 / 2

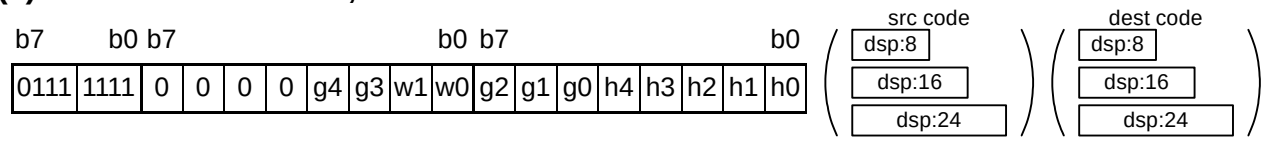
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADC

ADC

(2) ADC.size *src,dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
[An]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:8[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:16[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24[ ]	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3
dsp:16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3

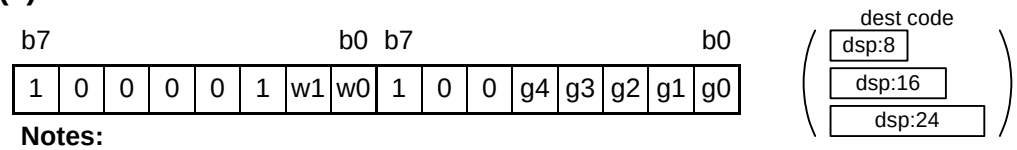
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

ADCF

ADCF

(1) ADCF.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADD

ADD

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

b7 b0 b7 b0

1	0	0	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dest code	#IMM
dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Note:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

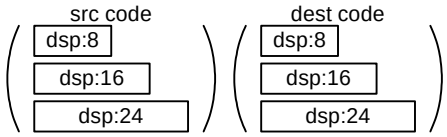
ADD

ADD

(2) ADD.size:G *src*,*dest*

b7 b0 b7 b0

1	0	0	0	g4	g3	w1	w0	g2	g1	g0	h4	h3	h2	h1	h0



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
[An]	2 / 2	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3
dsp:8[ ]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:16[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:16	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

ADD

ADD

(3) ADD.L:G #IMM,SP

b7							b0	b7							b0
1	0	0	0	0	1	w1	w0	0	0	0	0	1	0	0	0

- #IMM:8
- #IMM:16
- #IMM:32

Notes:

- Bits w1 and w0 are specified for the operand length of #IMM.
- #IMM is sign-extended to 32 bits irrespective of the value of bits w1 and w0.

[Bytes/Cycles]

src	#IMM:8	#IMM:16	#IMM:32
Bytes/Cycles	3 / 2	4 / 2	6 / 2

ADD

ADD

(4) ADD.size:Q #IMM:4,*dest*

b7							b0	b7							b0
1	1	1	1	0	q3	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

dest code

- dsp:8
- dsp:16
- dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 4 bits (q3 to q0) are specified for #IMM:4 and bits g4 to g0 are for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADD

ADD

(5) ADD.L:Q #IMM:3,SP

b7							b0
0	q2	0	0	0	0	q1	q0

Note:

- Bits q2 to q0 are specified for #IMM:3. Refer to **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

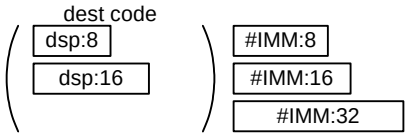
Bytes/Cycles	1 / 1

ADD

ADD

(6) ADD.size:S #IMM,*dest*

b7							b0
0	1	1	1	s1	s0	w1	w0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits s1 to s0 are specified for dest.

[Bytes/Cycles]

src \ *dest*	Register	dsp:8[ ]	dsp:16
#IMM:8	2 / 1	3 / 2	4 / 2
#IMM:16	3 / 1	4 / 2	5 / 2
#IMM:32	5 / 1	6 / 2	7 / 2

Note:

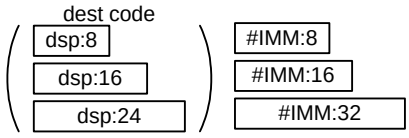
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADDF

ADDF

(1) ADDF #IMM(EX),*dest*

b7	b0	b7							b0	b7							b0
0111	1111	1	0	0	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMMEX:8	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
#IMMEX:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
#IMM:32	7 / 4	7 / 5	8 / 5	9 / 5	10 / 5	9 / 5	10 / 5

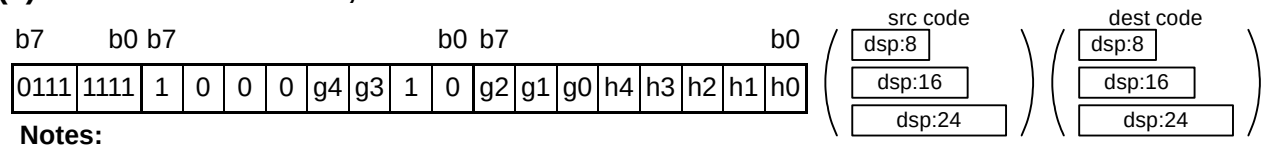
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ADDF

ADDF

(2) ADDF *src,dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
[An]	3 / 5	3 / 6	4 / 6	5 / 6	6 / 6	5 / 6	6 / 6
dsp:8[ ]	4 / 5	4 / 6	5 / 6	6 / 6	7 / 6	6 / 6	7 / 6
dsp:16[ ]	5 / 5	5 / 6	6 / 6	7 / 6	8 / 6	7 / 6	8 / 6
dsp:24[ ]	6 / 5	6 / 6	7 / 6	8 / 6	9 / 6	8 / 6	9 / 6
dsp:16	5 / 5	5 / 6	6 / 6	7 / 6	8 / 6	7 / 6	8 / 6
dsp:24	6 / 5	6 / 6	7 / 6	8 / 6	9 / 6	8 / 6	9 / 6

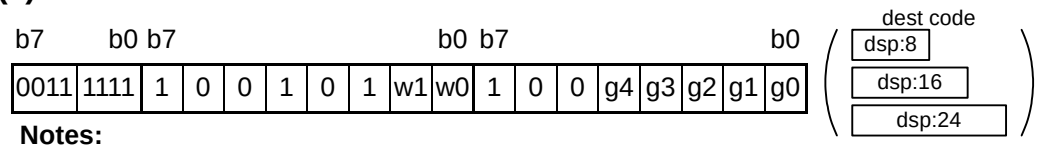
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

ADSF

ADSF

(1) ADSF.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

AND

AND

(1) AND.size:G #IMM(EX),*dest*

b7 b0 b7 b0

1	1	0	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

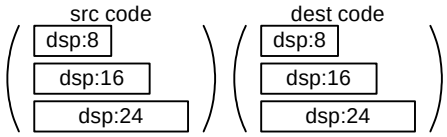
AND

AND

(2) AND.size:G *src*,*dest*

b7 b0 b7 b0

1	1	0	0	g4	g3	w1	w0	g2	g1	g0	h4	h3	h2	h1	h0



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
[An]	2 / 2	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3
dsp:8[ ]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:16[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:16	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3

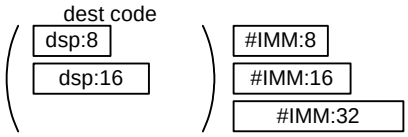
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

AND AND

(3) AND.size:S #IMM,*dest*

b7							b0
0	1	1	0	s1	s0	w1	w0



Notes:
- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for dest.

[Bytes/Cycles]

src \ *dest*	Register	dsp:8[ ]	dsp:16
#IMM:8	2 / 1	3 / 2	4 / 2
#IMM:16	3 / 1	4 / 2	5 / 2
#IMM:32	5 / 1	6 / 2	7 / 2

Note:
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BCLR BCLR

(1) BCLR bit,*dest*

b7							b0	b7							b0
0	1	0	0	0	1	0	0	b2	b1	b0	g4	g3	g2	g1	g0

Notes:
- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Operation length is one-byte basis.

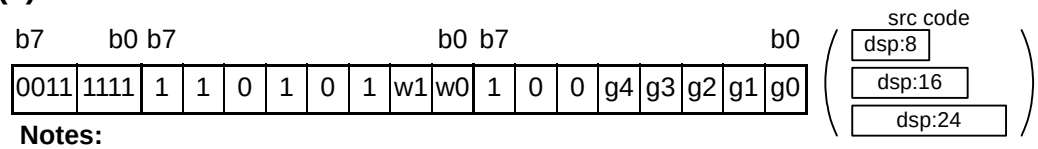
[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BITINDEX BITINDEX

(1) BITINDEX.size *src*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

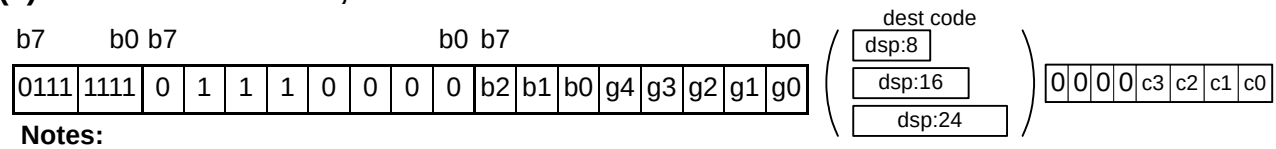
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4

Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.
- The cycles for the next sequential bit instruction increases by 2.

BM*Cnd* BM*Cnd*

(1) BM*Cnd* bit,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Bits c3 to c0 are specified for *Cnd.* Refer to **4.2.3, "Specifying Conditions"** for details.
- Operation length is one-byte basis.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 5	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5

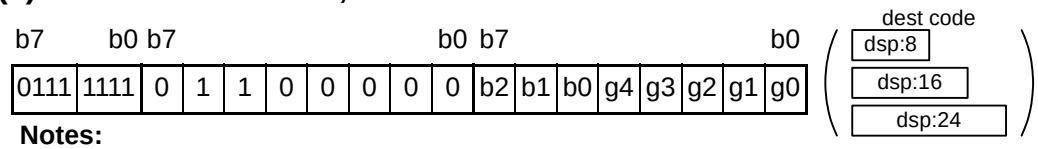
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BNOT

BNOT

(1) BNOT bit,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Operation length is one-byte basis.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

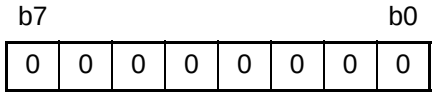
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BRK

BRK

(1) BRK



[Bytes/Cycles]

Bytes/Cycles	1 / 16

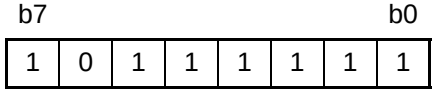
Note:

- Cycles for relocatable vectors. 19 cycles are adopted for fixed vectors.

BRK2

BRK2

(1) BRK2



[Bytes/Cycles]

Bytes/Cycles	1 / 19

BSET

BSET

(1) BSET bit,*dest*

b7							b0	b7							b0
0	1	0	0	1	1	0	0	b2	b1	b0	g4	g3	g2	g1	g0

dest code
- dsp:8
- dsp:16
- dsp:24

Notes:
- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Operation length is one-byte basis.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BTST

BTST

(1) BTST:G bit,*src*

b7							b0	b7							b0
0	1	0	0	1	0	0	0	b2	b1	b0	g4	g3	g2	g1	g0

src code
- dsp:8
- dsp:16
- dsp:24

Notes:
- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for src.
- Operation length is one-byte basis.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BTST

BTST

(2) BTST:S bit,abs:16

b7							b0
0	b2	b1	b0	0	1	1	1

src code: abs:16

Notes:

- Operation length is one-byte basis.
- 3 bits (b2 to b0) are specified for the bit position.

[Bytes/Cycles]

Bytes/Cycles	3 / 2

BTSTC

BTSTC

(1) BTSTC bit,*dest*

b7 b0	b7							b0	b7							b0
0111	1111	0	1	1	0	0	1	0	0	b2	b1	b0	g4	g3	g2	g1	g0

dest code: dsp:8 / dsp:16 / dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Operation length is one-byte basis.

[Bytes/Cycles]

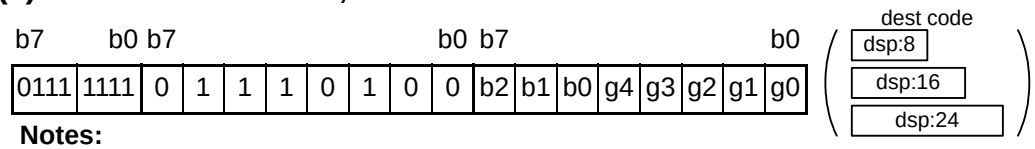
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

BTSTS BTSTS

(1) BTSTS bit,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 3 bits (b2 to b0) are specified for the bit position and bits g4 to g0 are for dest.
- Operation length is one-byte basis.

[Bytes/Cycles]

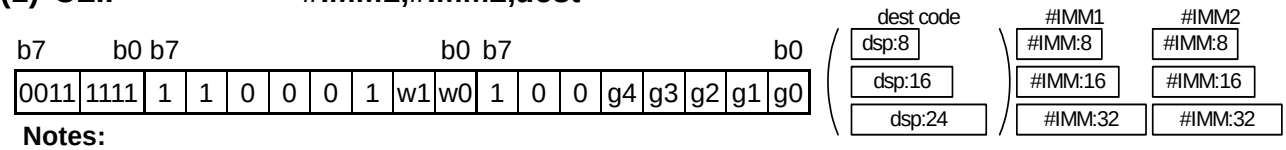
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

CLIP CLIP

(1) CLIP #IMM1,#IMM2,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5

Notes:

- When size specifier (.size) is ".W", the required bytes in the table increases by 2. In the case of ".L ", it increases by 6.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

CMP

CMP

(1) CMP.size:G #IMM(EX),*dest*

b7 b0 b7 b0

1	0	1	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

Note:

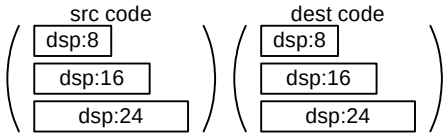
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

CMP CMP

(2) CMP.size:G *src*,*dest*

b7 b0 b7 b0

1	0	1	0	g4	g3	w1	w0	g2	g1	g0	h4	h3	h2	h1	h0



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 and bits h4 to h0 are respectively specified for dest and for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
[An]	2 / 2	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3
dsp:8[ ]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:16[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:16	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

CMP CMP

(3) CMP.size:Q #IMM:4,*dest*

b7 b0 b7 b0

1	1	1	0	0	q3	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 4 bits (q3 to q0) are specified for #IMM:4 and bits g4 to g0 are for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

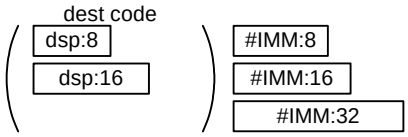
CMP

CMP

(4) CMP.size:S #IMM,*dest*

b7 b0

0	0	1	1	s1	s0	w1	w0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for dest.

[Bytes/Cycles]

src \ *dest*	Register	dsp:8[ ]	dsp:16
#IMM:8	2 / 1	3 / 2	4 / 2
#IMM:16	3 / 1	4 / 2	5 / 2
#IMM:32	5 / 1	6 / 2	7 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

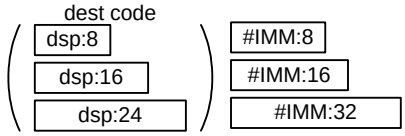
CMPF

CMPF

(1) CMPF #IMM(EX),*dest*

b7 b0 b7 b0 b7 b0

0111	1111	1	0	1	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMMEX:8	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
#IMMEX:16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
#IMM:32	7 / 3	7 / 4	8 / 4	9 / 4	10 / 4	9 / 4	10 / 4

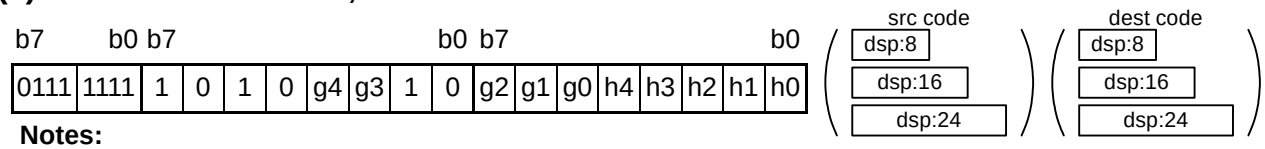
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

CMPF

CMPF

(2) CMPF *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

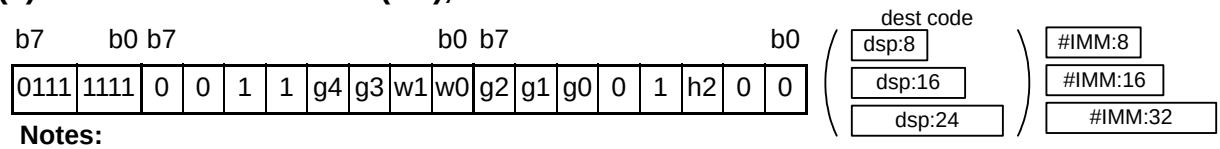
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

CNVIF

CNVIF

(1) CNVIF #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMMEX:8	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
#IMMEX:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
#IMM:32	7 / 4	7 / 5	8 / 5	9 / 5	10 / 5	9 / 5	10 / 5

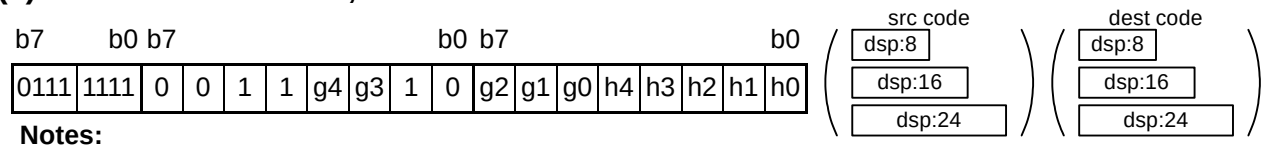
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

CNVIF

CNVIF

(2) CNVIF *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 4	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 5	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 5	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 5	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 5	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 5	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 5	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

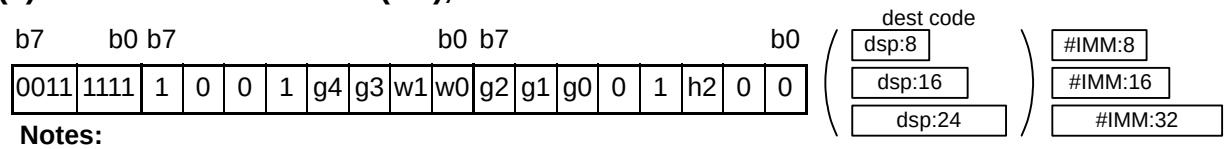
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DADC

DADC

(1) DADC.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 6	4 / 7	5 / 7	6 / 7	7 / 7	6 / 7	7 / 7
#IMM(EX):16	5 / 6	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
#IMM:32	7 / 6	7 / 7	8 / 7	9 / 7	10 / 7	9 / 7	10 / 7

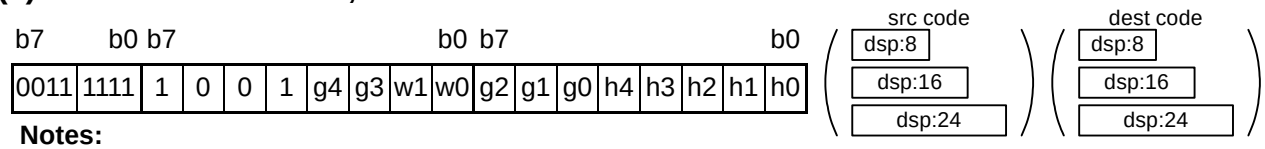
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DADC

DADC

(2) DADC.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 6	3 / 6	4 / 6	5 / 6	6 / 6	5 / 6	6 / 6
[An]	3 / 7	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7
dsp:8[ ]	4 / 7	4 / 7	5 / 7	6 / 7	7 / 7	6 / 7	7 / 7
dsp:16[ ]	5 / 7	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
dsp:24[ ]	6 / 7	6 / 7	7 / 7	8 / 7	9 / 7	8 / 7	9 / 7
dsp:16	5 / 7	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
dsp:24	6 / 7	6 / 7	7 / 7	8 / 7	9 / 7	8 / 7	9 / 7

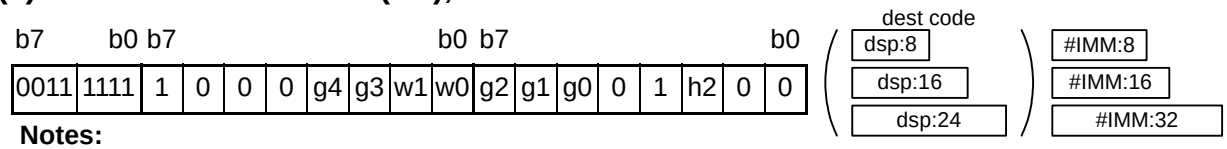
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DADD

DADD

(1) DADD.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 6	4 / 7	5 / 7	6 / 7	7 / 7	6 / 7	7 / 7
#IMM(EX):16	5 / 6	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
#IMM:32	7 / 6	7 / 7	8 / 7	9 / 7	10 / 7	9 / 7	10 / 7

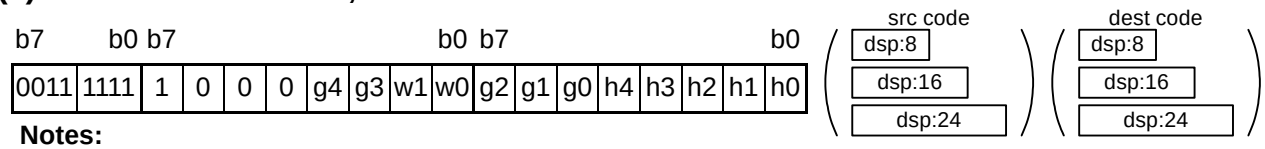
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DADD

DADD

(2) DADD.size *src,dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 6	3 / 6	4 / 6	5 / 6	6 / 6	5 / 6	6 / 6
[An]	3 / 7	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7
dsp:8[ ]	4 / 7	4 / 7	5 / 7	6 / 7	7 / 7	6 / 7	7 / 7
dsp:16[ ]	5 / 7	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
dsp:24[ ]	6 / 7	6 / 7	7 / 7	8 / 7	9 / 7	8 / 7	9 / 7
dsp:16	5 / 7	5 / 7	6 / 7	7 / 7	8 / 7	7 / 7	8 / 7
dsp:24	6 / 7	6 / 7	7 / 7	8 / 7	9 / 7	8 / 7	9 / 7

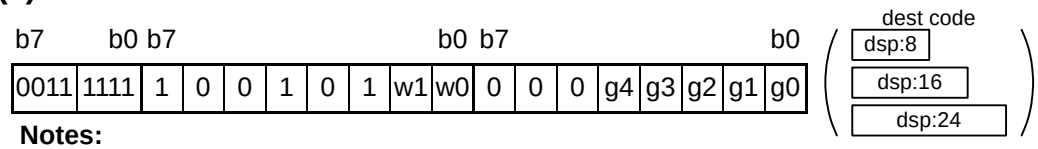
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DEC

DEC

(1) DEC.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

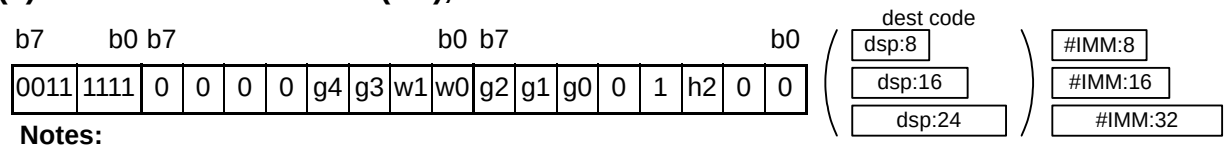
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DIV

DIV

(1) DIV.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 10	4 / 11	5 / 11	6 / 11	7 / 11	6 / 11	7 / 11
#IMM(EX):16	5 / 10	5 / 11	6 / 11	7 / 11	8 / 11	7 / 11	8 / 11
#IMM:32	7 / 10	7 / 11	8 / 11	9 / 11	10 / 11	9 / 11	10 / 11

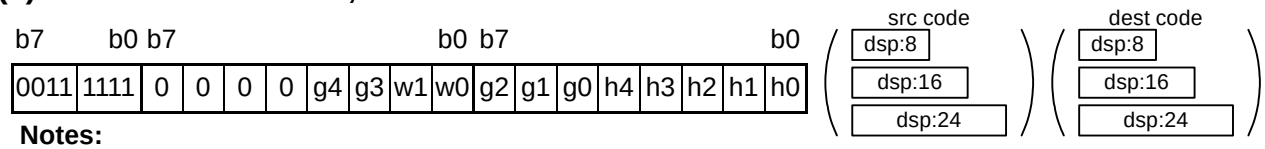
Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DIV

DIV

(2) DIV.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 10	3 / 11	4 / 11	5 / 11	6 / 11	5 / 11	6 / 11
[An]	3 / 11	3 / 12	4 / 12	5 / 12	6 / 12	5 / 12	6 / 12
dsp:8[ ]	4 / 11	4 / 12	5 / 12	6 / 12	7 / 12	6 / 12	7 / 12
dsp:16[ ]	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24[ ]	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12
dsp:16	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12

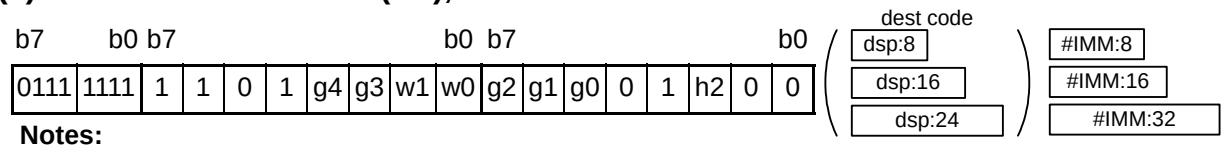
Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DIVF

DIVF

(1) DIVF #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 16	4 / 17	5 / 17	6 / 17	7 / 17	6 / 17	7 / 17
#IMM(EX):16	5 / 16	5 / 17	6 / 17	7 / 17	8 / 17	7 / 17	8 / 17
#IMM:32	7 / 16	7 / 17	8 / 17	9 / 17	10 / 17	9 / 17	10 / 17

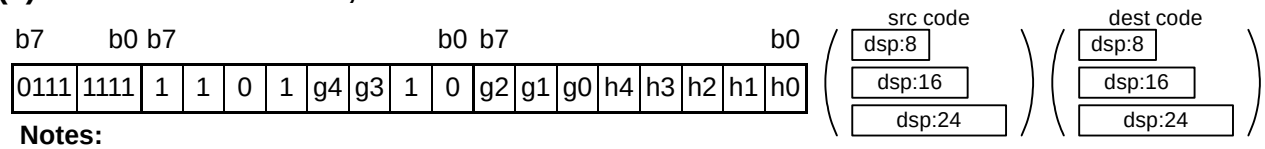
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DIVF

DIVF

(2) DIVF *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 16	3 / 17	4 / 17	5 / 17	6 / 17	5 / 17	6 / 17
[An]	3 / 17	3 / 18	4 / 18	5 / 18	6 / 18	5 / 18	6 / 18
dsp:8[ ]	4 / 17	4 / 18	5 / 18	6 / 18	7 / 18	6 / 18	7 / 18
dsp:16[ ]	5 / 17	5 / 18	6 / 18	7 / 18	8 / 18	7 / 18	8 / 18
dsp:24[ ]	6 / 17	6 / 18	7 / 18	8 / 18	9 / 18	8 / 18	9 / 18
dsp:16	5 / 17	5 / 18	6 / 18	7 / 18	8 / 18	7 / 18	8 / 18
dsp:24	6 / 17	6 / 18	7 / 18	8 / 18	9 / 18	8 / 18	9 / 18

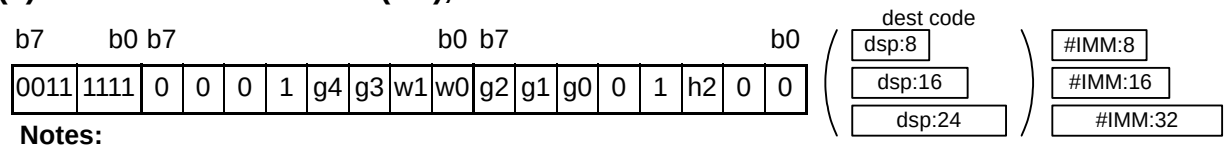
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DIVU

DIVU

(1) DIVU.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 10	4 / 11	5 / 11	6 / 11	7 / 11	6 / 11	7 / 11
#IMM(EX):16	5 / 10	5 / 11	6 / 11	7 / 11	8 / 11	7 / 11	8 / 11
#IMM:32	7 / 10	7 / 11	8 / 11	9 / 11	10 / 11	9 / 11	10 / 11

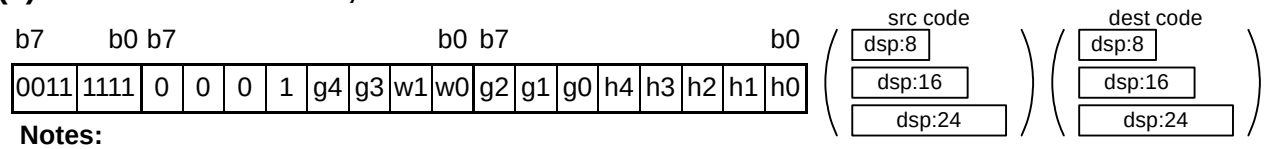
Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DIVU

DIVU

(2) DIVU.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 10	3 / 11	4 / 11	5 / 11	6 / 11	5 / 11	6 / 11
[An]	3 / 11	3 / 12	4 / 12	5 / 12	6 / 12	5 / 12	6 / 12
dsp:8[ ]	4 / 11	4 / 12	5 / 12	6 / 12	7 / 12	6 / 12	7 / 12
dsp:16[ ]	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24[ ]	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12
dsp:16	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12

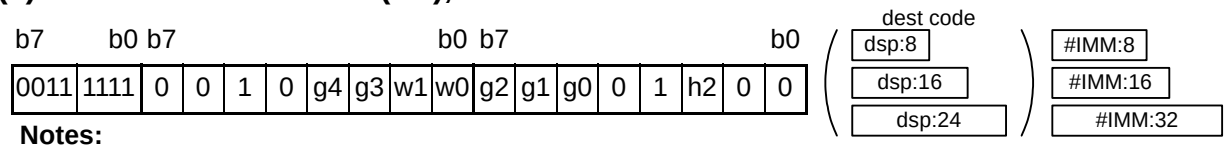
Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DIVX

DIVX

(1) DIVX.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 10	4 / 11	5 / 11	6 / 11	7 / 11	6 / 11	7 / 11
#IMM(EX):16	5 / 10	5 / 11	6 / 11	7 / 11	8 / 11	7 / 11	8 / 11
#IMM:32	7 / 10	7 / 11	8 / 11	9 / 11	10 / 11	9 / 11	10 / 11

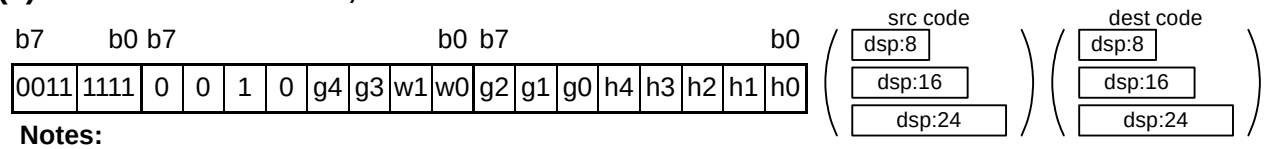
Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DIVX

DIVX

(2) DIVX.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 10	3 / 11	4 / 11	5 / 11	6 / 11	5 / 11	6 / 11
[An]	3 / 11	3 / 12	4 / 12	5 / 12	6 / 12	5 / 12	6 / 12
dsp:8[ ]	4 / 11	4 / 12	5 / 12	6 / 12	7 / 12	6 / 12	7 / 12
dsp:16[ ]	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24[ ]	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12
dsp:16	5 / 11	5 / 12	6 / 12	7 / 12	8 / 12	7 / 12	8 / 12
dsp:24	6 / 11	6 / 12	7 / 12	8 / 12	9 / 12	8 / 12	9 / 12

Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DSBB

DSBB

(1) DSBB.size #IMM(EX),*dest*

b7 b0	b7							b0	b7							b0
0011 1111	1	1	0	1	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
#IMM(EX):16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
#IMM:32	7 / 3	7 / 4	8 / 4	9 / 4	10 / 4	9 / 4	10 / 4

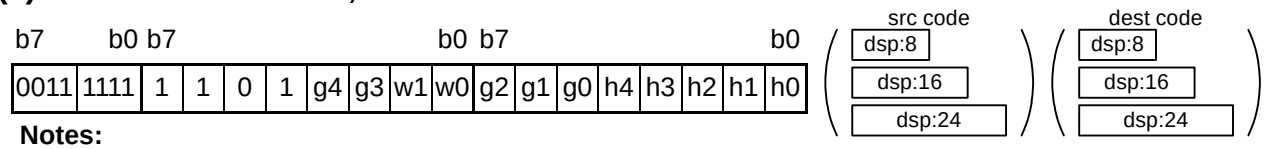
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DSBB

DSBB

(2) DSBB.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

DSUB

DSUB

(1) DSUB.size #IMM(EX),*dest*

b7 b0	b7							b0	b7							b0
0011 1111	1	1	0	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dest code	#IMM
dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
#IMM(EX):16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
#IMM:32	7 / 3	7 / 4	8 / 4	9 / 4	10 / 4	9 / 4	10 / 4

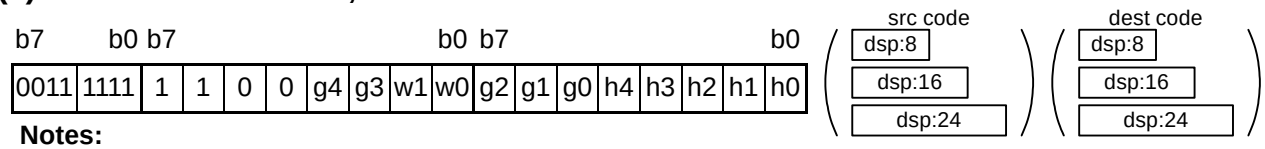
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

DSUB

DSUB

(2) DSUB.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

EDIV

EDIV

(1) EDIV.size *src,dest*

b7 b0	b7 b0	b7 b0															
0011	1111	0	1	1	0	0	1	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

src code
- #IMM:8
- #IMM:16
- #IMM:32
- dsp:8
- dsp:16
- dsp:24

Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Regis-ter	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 16	4 / 16	5 / 16	7 / 16	3 / 16	4 / 16	5 / 16	6 / 16	5 / 16	6 / 16

Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

EDIVU

EDIVU

(1) EDIVU.size *src*,*dest*

b7 b0	b7 b0	b7 b0								b7 b0							
0011	1111	0	1	1	1	0	1	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

src code
- #IMM:8
- #IMM:16
- #IMM:32
- dsp:8
- dsp:16
- dsp:24

Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Regis-ter	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 14	4 / 14	5 / 14	7 / 14	3 / 14	4 / 14	5 / 14	6 / 14	5 / 14	6 / 14

Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

EDIVX

EDIVX

(1) EDIVX.size *src,dest*

b7 b0 b7 b0 b7 b0

0011	1111	0	1	1	0	1	1	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

src code
- #IMM:8
- #IMM:16
- #IMM:32
- dsp:8
- dsp:16
- dsp:24

Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Regis-ter	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 16	4 / 16	5 / 16	7 / 16	3 / 16	4 / 16	5 / 16	6 / 16	5 / 16	6 / 16

Notes:

- When size specifier (.size) is ".W", the required cyles in the table increases by 3. In the case of ".L", it increases by 10.
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

EMUL

EMUL

(1) EMUL.size *src*,*dest*

b7 b0	b7 b0	b7 b0															
0011	1111	0	1	1	0	0	0	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

src code
- #IMM:8
- #IMM:16
- #IMM:32
- dsp:8
- dsp:16
- dsp:24

Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Regis-ter	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	4 / 2	5 / 2	7 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

Notes:

- Cycles when size specifier (.size) is “.B” or ".W". In the case of ".L", it increases by 1.
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

EMULU

EMULU

(1) EMUL.size *src*,*dest*

b7	b0	b7							b0	b7							b0
0011	1111	0	1	1	1	0	0	w1	w0	q2	q1	q0	g4	g3	g2	g1	g0

src code
- #IMM:8
- #IMM:16
- #IMM:32
- dsp:8
- dsp:16
- dsp:24

Notes:
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Regis-ter	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	4 / 2	5 / 2	7 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

Notes:
- Cycles when size specifier (.size) is “.B” or ".W". In the case of ".L", it increases by 1.
- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

ENTER

ENTER

(1) ENTER #IMM:8

b7							b0
0	0	0	1	1	0	1	1

- #IMM:8

[Bytes/Cycles]

Bytes/Cycles	2 / 3

ENTER

ENTER

(2) ENTER #IMM:16

b7							b0
1	1	0	1	1	0	1	1

- #IMM:16

[Bytes/Cycles]

Bytes/Cycles	3 / 3

EXITD

EXITD

(1) EXITD

b7							b0
1	1	1	0	1	1	1	1

[Bytes/Cycles]

Bytes/Cycles	1 / 7

EXITI

EXITI

(1) EXITI

b7							b0	b7							b0
1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 8

EXTS

EXTS

(1) EXTS.size #IMM,*dest*

b7		b0	b7							b0	b7							b0
0011	1111	1	0	1	1	g4	g3	w1	w0	g2	g1	g0	0	1	1	0	0	

dest code: dsp:8, dsp:16, dsp:24 → #IMM:8, #IMM:16

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- Bits w1 and w0 are specified for the operand length of src and dst.

w1	w0	src	dest
0	0	B	W
0	1	B	L
1	0	W	L

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM:8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2

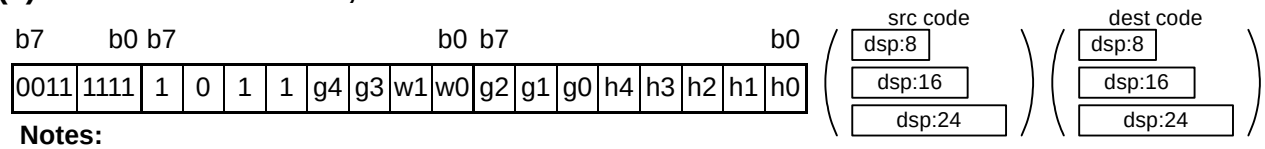
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

EXTS

EXTS

(2) EXTS.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.
- Bits w1 and w0 are specified for the operand length of src and dst.

w1	w0	src	dest
0	0	B	W
0	1	B	L
1	0	W	L

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 1	4 / 1	5 / 1	6 / 1	5 / 1	6 / 1
[An]	3 / 2	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
dsp:8[ ]	4 / 2	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
dsp:16[ ]	5 / 2	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
dsp:24[ ]	6 / 2	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2
dsp:16	5 / 2	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
dsp:24	6 / 2	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

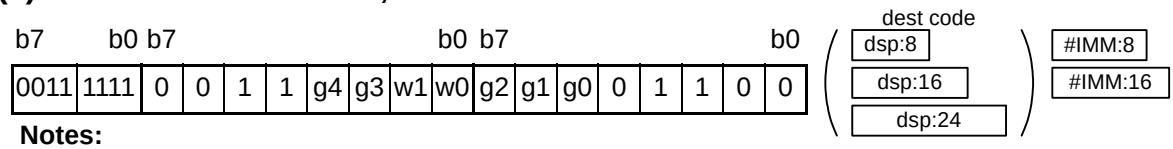
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

EXTZ

EXTZ

(1) EXTZ.size:G #IMM,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- Bits w1 and w0 are specified for the operand length of src and dst.

w1	w0	src	dest
0	0	B	W
0	1	B	L
1	0	W	L

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM:8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2

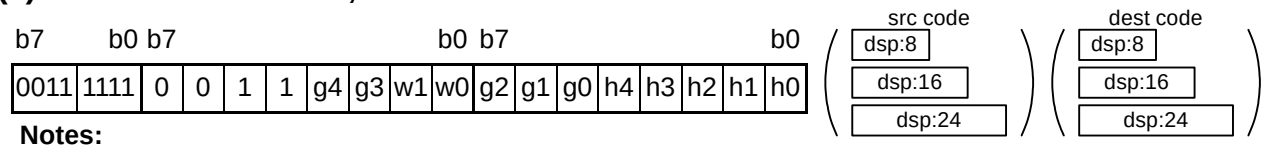
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

EXTZ

EXTZ

(2) EXTZ.size:G *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.
- Bits w1 and w0 are specified for the operand length of src and dst.

w1	w0	src	dest
0	0	B	W
0	1	B	L
1	0	W	L

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
[An]	3 / 3	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:8[ ]	4 / 3	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:16[ ]	5 / 3	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24[ ]	6 / 3	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3
dsp:16	5 / 3	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24	6 / 3	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

EXTZ

EXTZ

(3) EXTZ.WL:S *src*,An

b7							b0	b7							b0
0	1	0	0	0	1	0	1	q2	q1	q0	g4	g3	g2	g1	g0

src code
- dsp:8
- dsp:16
- dsp:24

Notes:
- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3

Note:
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

FCLR

FCLR

(1) FCLR *dest*

b7							b0	b7							b0
0	1	0	0	1	1	0	0	q2	q1	q0	0	1	0	0	0

Note:
- 3 bits (q2 to q0) are specified for the flag of dest. Refer to **(9), "FLG direct addressing"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

Bytes/Cycles	2 / 3

FREIT

FREIT

(1) FREIT

b7							b0
1	0	0	0	1	1	1	1

[Bytes/Cycles]

Bytes/Cycles	1 / 4

FSET

FSET

(1) FSET *dest*

b7							b0	b7							b0
0	1	0	0	1	1	0	1	q2	q1	q0	0	1	0	0	0

Note:

- 3 bits (q2 to q0) are specified for the flag of dest. Refer to **(9), "FLG direct addressing"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

Bytes/Cycles	2 / 3

INC

INC

(1) INC.size *dest*

b7		b0	b7							b0	b7							b0
0011	1111	1	0	0	0	0	1	w1	w0	0	0	0	g4	g3	g2	g1	g0	

dest code: dsp:8 / dsp:16 / dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

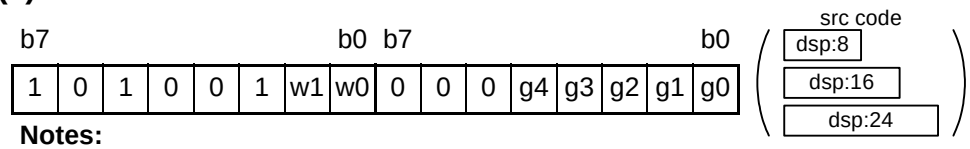
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

INDEX1 INDEX1

(1) INDEX1.size *src*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

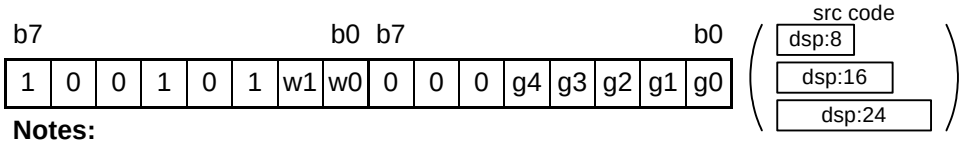
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Notes:

- The cycles for the next sequential bit instruction increases by 2.
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

INDEX2 INDEX2

(1) INDEX2.size *src*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

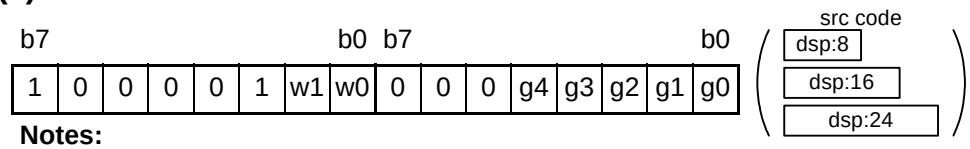
Notes:

- The cycles for the next sequential bit instruction increases by 2.
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

INDEXB

INDEXB

(1) INDEXB.size *src*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

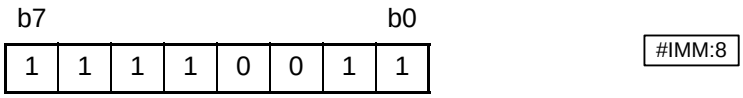
Notes:

- The cycles for the next sequential instruction increase by 4. If this instruction is EXTS or EXTZ, the cycles increase by 5.
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

INT

INT

(1) INT #IMM:8



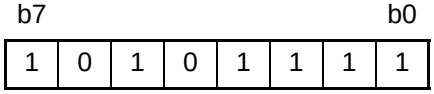
[Bytes/Cycles]

Bytes/Cycles	2 / 11

INTO

INTO

(1) INTO



[Bytes/Cycles]

Bytes/Cycles	1 / 1(12)*1

*1. The cycles are 12 when the O flag is 1.

JCnd JCnd

(1) JCnd label

b7							b0
c3	c2	c1	c0	0	0	1	1

label code: dsp:8

Note:
- Bits c3 to c0 are specified for *Cnd.* Refer to **4.2.3, "Specifying Conditions"** for details.

[Bytes/Cycles]

Bytes/Cycles	2 / 1(3)*1

Note:
*1. When the instruction is jumped to label, the cyles is 3.

JMP JMP

(1) JMP.S label

b7							b0
1	q2	q1	q0	0	1	1	1

Note:
- Bits q2 to q0 are specified for dsp3. Refer to **(7), "Program counter relative addressing, short format"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

Bytes/Cycles	1 / 1

JMP JMP

(2) JMP.B label

b7							b0
0	1	1	1	0	0	1	1

label code: dsp:8

[Bytes/Cycles]

Bytes/Cycles	2 / 3

JMP JMP

(3) JMP.W label

b7							b0
1	0	1	0	1	0	1	1

label code: dsp:16

[Bytes/Cycles]

Bytes/Cycles	3 / 3

JMP

JMP

(4) JMP.A label

b7							b0
1	1	1	0	1	0	1	1

label code: dsp:24

[Bytes/Cycles]

Bytes/Cycles	4 / 3

JMPI

JMPI

(1) JMPI.W *src*

b7	b0	b7							b0	b7							b0
0111	1111	1	0	1	0	0	1	0	1	0	0	0	g4	g3	g2	g1	g0

src code: dsp:8 / dsp:16 / dsp:24

Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 6	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

JMPI

JMPI

(2) JMPI.L *src*

b7	b0	b7							b0	b7							b0
0111	1111	1	0	1	0	0	1	1	0	0	0	0	g4	g3	g2	g1	g0

src code: dsp:8 / dsp:16 / dsp:24

Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 6	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

JSR

JSR

(1) JSR.W label

b7							b0
1	0	1	1	1	0	1	1

label code: dsp:16

[Bytes/Cycles]

Bytes/Cycles	3 / 3

JSR

JSR

(2) JSR.A label

b7							b0
1	1	1	1	1	0	1	1

label code: dsp:24

[Bytes/Cycles]

Bytes/Cycles	4 / 3

JSRI

JSRI

(1) JSRI.W *src*

b7		b0 b7						b0 b7								b0
0111	1111	1	0	1	0	0	1	0	1	1	0	0	g4	g3	g2	g1	g0

src code: dsp:8 / dsp:16 / dsp:24

Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 6	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

JSRI JSRI

(2) JSRI.L *src*

b7 b0	b7 b0	b7 b0	b7 b0
0111 1111	1 0 1 0 0 1 1 0	1 0 0 g4 g3 g2 g1 g0	

src code: dsp:8 / dsp:16 / dsp:24

Notes:
- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 6	3 / 7	4 / 7	5 / 7	6 / 7	5 / 7	6 / 7

Note:
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

LDC LDC

(1) LDC *src,dest*

b7 b0	b7 b0	b7 b0	b7 b0
0011 1111	1 1 1 0 1 0 w1 w0	q2 q1 q0 g4 g3 g2 g1 g0	

src code: #IMM:8 / #IMM:16 / #IMM:32 / dsp:8 / dsp:16 / dsp:24

Notes:
- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.
- Bits q2 to q0 are specified for the control register of dest. Refer to **(5), "Control register direct addressing (CPU)"** in **4.2.2, "Specifying the Operand"** for details.

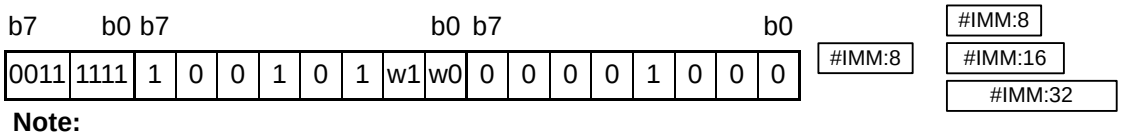
[Bytes/Cycles]

src	Regis-ter	#IMMEX:8	#IMMEX:16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	4 / 3	5 / 3	7 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4

Note:
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

LDC LDC

(2) LDC #IMM,*dest*



Note:

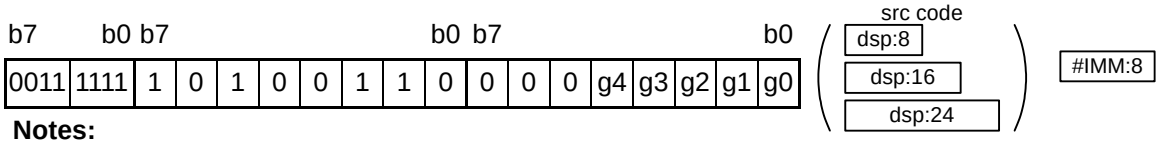
- #IMM:8 is specified for the register of dest. Refer to **(6), "Control register direct addressing (DMAC,VCT)"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

src	#IMM:8	#IMM:16	#IMM:32
Bytes/Cycles	5 / 3	6 / 3	8 / 3

LDC LDC

(3) LDC *src*,*dest*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.
- #IMM:8 is specified for the register of dest. Refer to **(6), "Control register direct addressing (DMAC,VCT)"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

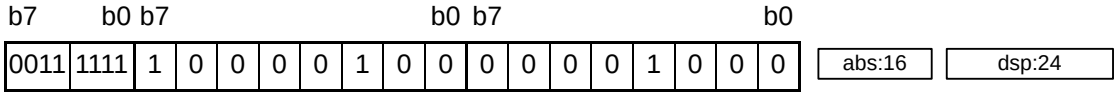
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 3	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

LDCTX LDCTX

(1) LDCTX abs:16,dsp:24



[Bytes/Cycles]

Bytes/Cycles	8 / 10+m*1

Note:

*1. m is the number of registers to be transferred.

LDIPL

LDIPL

(1) LDIPL #IMM:3

b7							b0	b7							b0
0	1	0	0	0	1	0	1	q2	q1	q0	0	1	0	0	0

Note:

- 3 bits (q2 to q0) are specified for #IMM:3

[Bytes/Cycles]

Bytes/Cycles	2 / 2

MAX

MAX

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

b7		b0	b7							b0	b7							b0
0111	1111	0	1	0	1	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0	

dest code

dest code	#IMM
dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

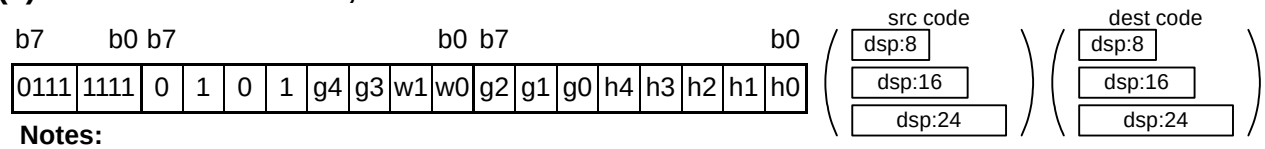
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
#IMM(EX):16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
#IMM:32	7 / 2	7 / 3	8 / 3	9 / 3	10 / 3	9 / 3	10 / 3

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MAX MAX

(2) MAX.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

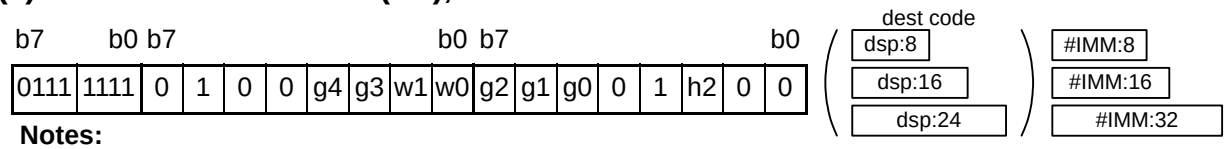
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
[An]	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
dsp:8[ ]	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
dsp:16[ ]	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24[ ]	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4
dsp:16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MIN MIN

(1) MIN.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

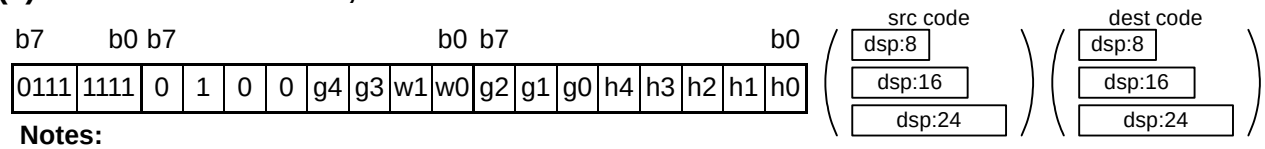
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
#IMM(EX):16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
#IMM:32	7 / 2	7 / 3	8 / 3	9 / 3	10 / 3	9 / 3	10 / 3

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MIN MIN

(2) MIN.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
[An]	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
dsp:8[ ]	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
dsp:16[ ]	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24[ ]	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4
dsp:16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MOV

MOV

(1) MOV.size:G #IMM(EX),*dest*

b7 b0 b7 b0

1	0	1	1	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

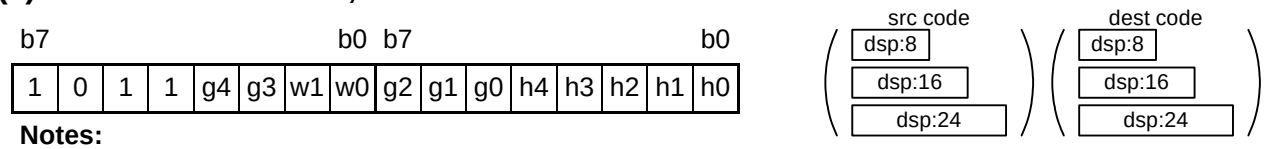
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV MOV

(2) MOV.size:G *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

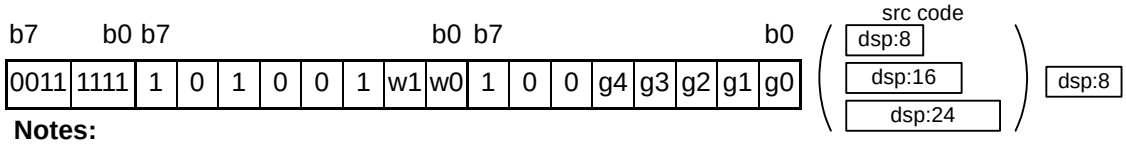
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 1	3 / 1	4 / 1	5 / 1	4 / 1	5 / 1
[An]	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
dsp:8[ ]	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
dsp:16[ ]	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
dsp:24[ ]	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
dsp:16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
dsp:24	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MOV MOV

(3) MOV.size:G *src*,dsp:8[SP]



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

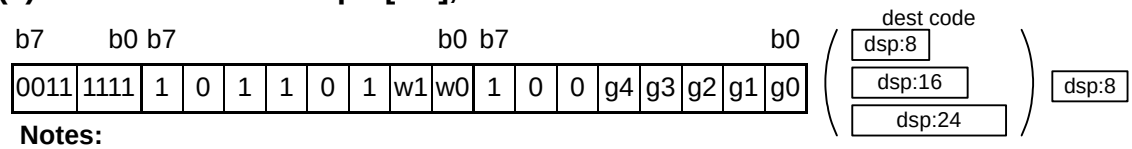
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV MOV

(4) MOV.size:G dsp:8[SP],*dest*



Notes:

- When dest is Bank1 register direct addressing, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

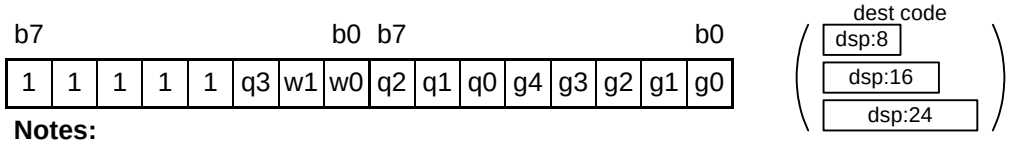
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 4	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4

Note:

- When dest is Bank1 register direct addressing, the required bytes in the table increases by 1.

MOV MOV

(5) MOV.size:Q #IMM:4,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 4 bits (q3 to q0) are specified for #IMM:4 and bits g4 to g0 are for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 1	3 / 1	4 / 1	5 / 1	4 / 1	5 / 1

Note:

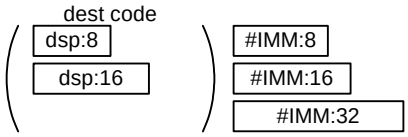
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV MOV

(6) MOV.size:S #IMM,*dest*

b7 b0

0	0	1	0	s1	s0	w1	w0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for dest.

[Bytes/Cycles]

src \ *dest*	Register	dsp:8[ ]	dsp:16
#IMM:8	2 / 1	3 / 2	4 / 2
#IMM:16	3 / 1	4 / 2	5 / 2
#IMM:32	5 / 1	6 / 2	7 / 2

Note:

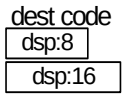
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV MOV

(7) MOV.size:S R0L/R0/R2R0,*dest*

b7 b0

0	0	0	0	s1	s0	w1	w0



Notes:

- When dest is indirect instruction addressing, 01101111 is put just before the insruction code.
- Bits s1 and s0 are specified for dest.

[Bytes/Cycles]

dest	dsp:8[ ]	dsp:16
Bytes/Cycles	2 / 1	3 / 1

Note:

- When dest is indirect instruction addressing, the required bytes and cycles in the table increase by 1 and 2, respectively.

MOV MOV

(8) MOV.size:S *src*,R0L/R0/R2R0

b7							b0
0	1	0	1	s1	s0	w1	w0

src code
dsp:8
dsp:16

Notes:
- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for src.

[Bytes/Cycles]

src	Register	dsp:8[ ]	dsp:16
Bytes/Cycles	1 / 1	2 / 1	3 / 1

Note:
- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV MOV

(9) MOV.L:S *src*,A0

b7							b0
0	1	0	0	s1	s0	1	0

src code
dsp:8
dsp:16

Notes:
- When src is indirect instruction addressing, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for src.

[Bytes/Cycles]

src	dsp:8[ ]	dsp:16
Bytes/Cycles	2 / 1	3 / 1

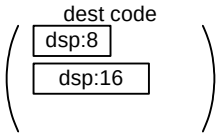
Note:
- When src is indirect instruction addressing, the required bytes and cycles in the table increase by 1 and 2, respectively.

MOV

MOV

(10)MOV.size:Z #0,*dest*

b7							b0
0	0	0	1	s1	s0	w1	w0



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits s1 and s0 are specified for dest.

[Bytes/Cycles]

dest	Register	dsp:8[ ]	dsp:16
Bytes/Cycles	1 / 1	2 / 1	3 / 1

Note:

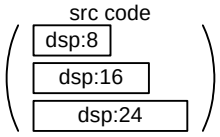
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOVA

MOVA

(1) MOVA *src*,*dest*

b7							b0	b7							b0
0	1	0	0	1	0	0	1	q2	q1	q0	g4	g3	g2	g1	g0



Notes:

- When dest is Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.
- Operation length is long word basis.

[Bytes/Cycles]

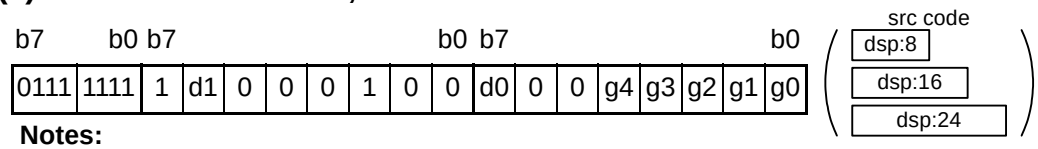
src	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	3 / 1	4 / 1	5 / 1	4 / 1	5 / 1

Note:

- When dest is Bank1 register direct addressing, the required bytes in the table increases by 1.

MOV*Dir* MOV*Dir*

(1) MOV*Dir* *src*,R0L



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.
- Bits d1 and d0 are specified for the operation (*Dir*).

d1	d0	*Dir*
0	0	LL
0	1	LH
1	0	HL
1	1	HH

[Bytes/Cycles]

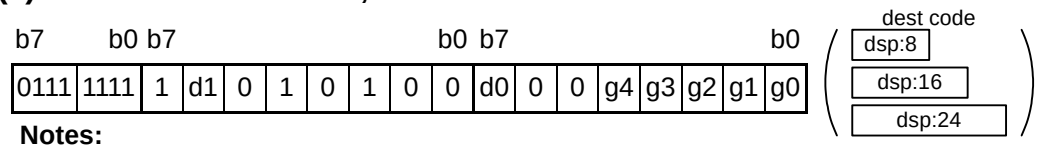
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MOV*Dir* MOV*Dir*

(2) MOV*Dir* R0L,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- Bits d1 and d0 are specified for the operation (*Dir*).

d1	d0	*Dir*
0	0	LL
0	1	LH
1	0	HL
1	1	HH

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

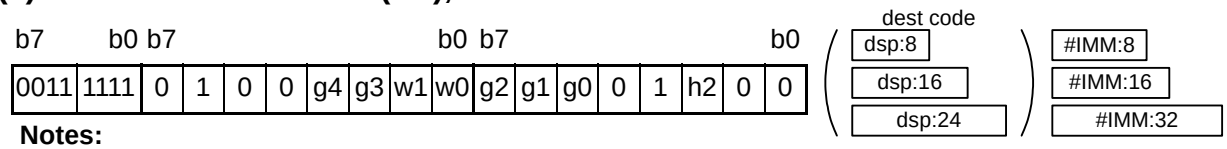
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MUL

MUL

(1) MUL.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
#IMM(EX):16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
#IMM:32	7 / 2	7 / 3	8 / 3	9 / 3	10 / 3	9 / 3	10 / 3

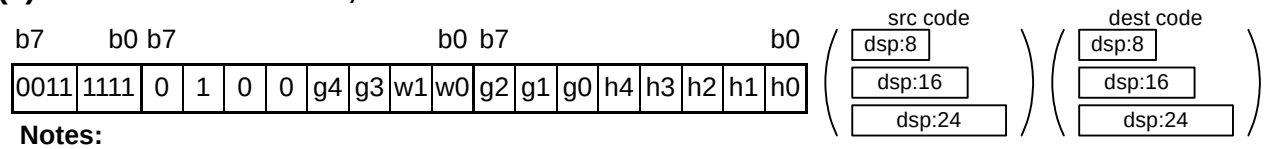
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MUL

MUL

(2) MUL.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
[An]	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
dsp:8[ ]	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
dsp:16[ ]	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24[ ]	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4
dsp:16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4

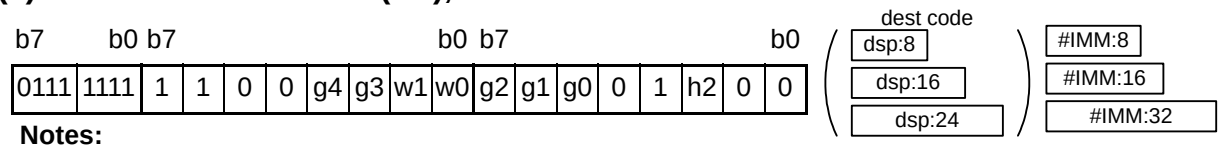
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MULF

MULF

(1) MULF #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
#IMM(EX):16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
#IMM:32	7 / 3	7 / 4	8 / 4	9 / 4	10 / 4	9 / 4	10 / 4

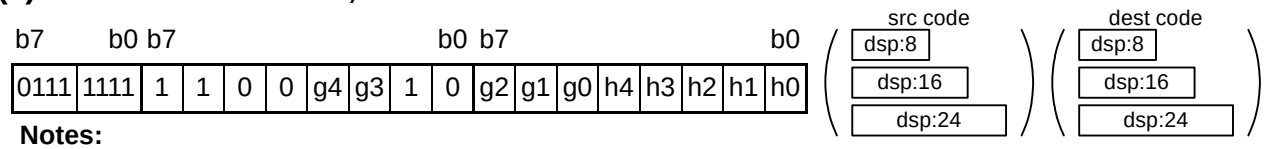
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MULF

MULF

(2) MULF *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 4	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

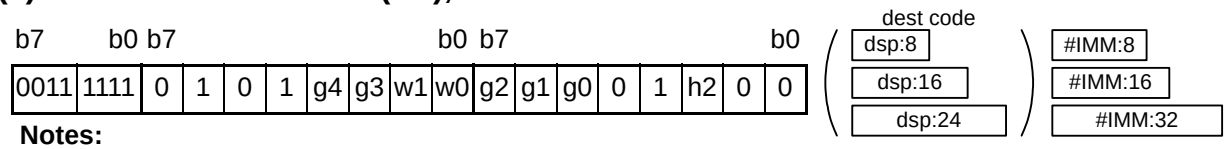
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MULU

MULU

(1) MULU.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
#IMM(EX):16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
#IMM:32	7 / 2	7 / 3	8 / 3	9 / 3	10 / 3	9 / 3	10 / 3

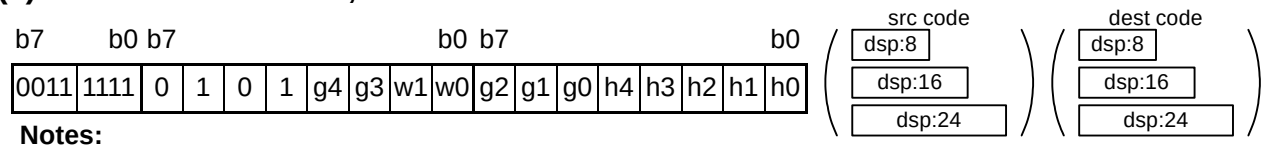
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

MULU

MULU

(2) MULU.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
[An]	3 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
dsp:8[ ]	4 / 3	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4
dsp:16[ ]	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24[ ]	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4
dsp:16	5 / 3	5 / 4	6 / 4	7 / 4	8 / 4	7 / 4	8 / 4
dsp:24	6 / 3	6 / 4	7 / 4	8 / 4	9 / 4	8 / 4	9 / 4

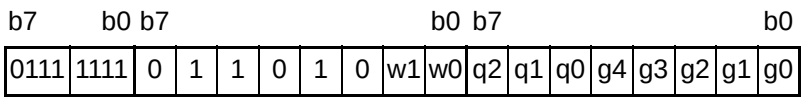
Note:

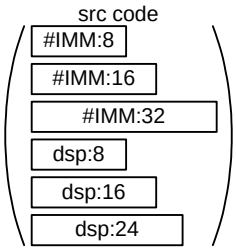
- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

MULX

MULX

(1) MULX.size *src,dest*





Notes:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is Bank1 register direct addressing
  01001111, when src is either indirect instruction addressing or Bank1 register direct addressing, and when dest is Bank1 register direct addressing
- Bits g4 to g0 are specified for src and bits q2 to q0 are for dest.

[Bytes/Cycles]

src	Register	#IMM(EX):8	#IMM(EX):16	#IMM:32	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	4 / 3	5 / 3	7 / 3	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4

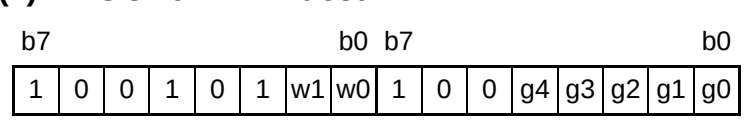
Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, or when dest is Bank1 register direct addressing, the required bytes in the table increse by 1. When it is either indirect instruction addressing or Bank1 register direct addressing, and dest is Bank1 register direct, the required cycles increse by 2.

NEG

NEG

(1) NEG.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

NOP

NOP

(1) NOP

b7							b0
1	0	0	1	1	1	1	1

[Bytes/Cycles]

Bytes/Cycles	1 / 1

NOT

NOT

(1) NOT.size *dest*

b7							b0	b7							b0
1	0	1	0	0	1	w1	w0	1	0	0	g4	g3	g2	g1	g0

dest code: dsp:8 / dsp:16 / dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

OR

OR

(1) OR.size #IMM(EX),*dest*

b7 b0 b7 b0

1	1	0	1	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dest code	#IMM
dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

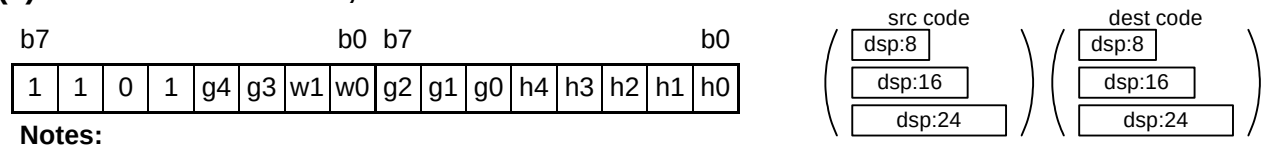
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

OR

OR

(2) OR.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
[An]	2 / 2	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3
dsp:8[ ]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:16[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:16	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3

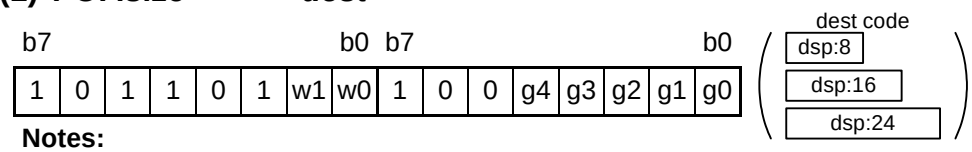
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

POP

POP

(1) POP.size *dest*



Notes:

- When dest is Bank1 register direct addressing, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 3	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3

Note:

- When dest is Bank1 register direct addressing, the required bytes in the table increases by 1.

POPC

POPC

(1) POPC ***dest***

b7							b0	b7							b0
0	1	0	0	1	0	0	1	q2	q1	q0	0	1	0	0	0

Notes:

- Bits q2 to q0 are specified for the control register of dest.
- Operation length is long word basis.

[Bytes/Cycles]

Bytes/Cycles	2 / 4

POPM

POPM

(1) POPM ***dest***

b7							b0
0	1	1	SB	1	0	1	1

#IMM:8

Notes:

- When SB register is popped, this bit is 1; otherwise it is 0.
- Other registers are specified in #IMM:8.

bit	7	6	5	4	3	2	1	0
Register	A3	A2	A1	A0	R7R5	R6R4	R3R1	R2R0

Note:

- When dest is Bank1 register direct addressing, 01101111 is put just before the instruction code.

[Bytes/Cycles]

Bytes/Cycles	2 / n*1

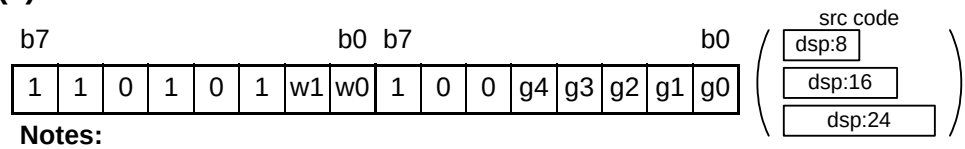
Note:

*1. n is the number of registers to be restored.

- When dest is Bank1 register direct addressing, the required bytes in the table increases by 1.

PUSH PUSH

(1) PUSH.size:G *src*



Notes:

- When src is either indirect instruction addressing or Bank 1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

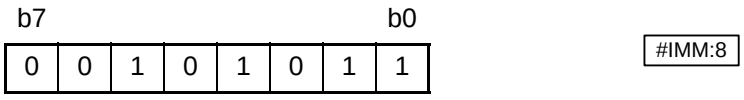
src	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When src is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

PUSH PUSH

(2) PUSH.B:S #IMM:8

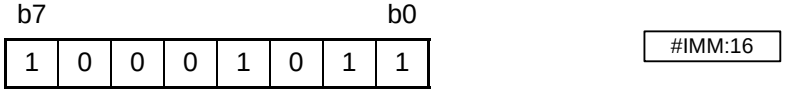


[Bytes/Cycles]

Bytes/Cycles	2 / 1

PUSH PUSH

(3) PUSH.W:S #IMM:16



[Bytes/Cycles]

Bytes/Cycles	3 / 1

PUSH

PUSH

(4) PUSH.L:S #IMM:32

b7							b0
1	1	0	0	1	0	1	1

#IMM:32

[Bytes/Cycles]

Bytes/Cycles	5 / 1

PUSH

PUSH

(5) PUSH.W:S #IMMEX:8

b7							b0
0	0	0	0	1	0	1	1

#IMM:8

[Bytes/Cycles]

Bytes/Cycles	2 / 1

PUSH

PUSH

(6) PUSH.L:S #IMMEX:8

b7							b0
0	0	1	1	1	0	1	1

#IMM:8

[Bytes/Cycles]

Bytes/Cycles	2 / 1

PUSH

PUSH

(7) PUSH.L:S #IMMEX:16

b7							b0
1	0	0	1	1	0	1	1

#IMM:16

[Bytes/Cycles]

Bytes/Cycles	3 / 1

PUSHA

PUSHA

(1) PUSHA *src*

b7 b0 b7 b0

0	1	0	0	1	1	0	1	0	0	0	g4	g3	g2	g1	g0

src code
dsp:8
dsp:16
dsp:24

Note:
- Bits g4 to g0 are specified for src.

[Bytes/Cycles]

src	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	4 / 1	5 / 1	4 / 1	5 / 1

PUSHC

PUSHC

(1) PUSHC *src*

b7 b0 b7 b0

0	1	0	0	1	0	0	0	q2	q1	q0	0	1	0	0	0

Notes:
- Bits q2 to q0 are specified for the control register of src.
- Operation length is long word basis.

[Bytes/Cycles]

Bytes/Cycles	2 / 3

PUSHM

PUSHM

(1) PUSHM *src*

b7 b0

0	1	0	SB	1	0	1	1

#IMM:8

Notes:
- When SB register is pushed, this bit is 1; otherwise it is 0.
- Other registers are specified in #IMM:8.

bit	7	6	5	4	3	2	1	0
Register	R2R0	R3R1	R6R4	R7R5	A0	A1	A2	A3

Note:
- When src is Bank1 register direct addressing, 01101111 is put just before the instruction code.

[Bytes/Cycles]

Bytes/Cycles	2 / n*1

Notes:
*1. n is the number of registers to be restored.
- When src is Bank1 register direct addressing, the required bytes in the table increases by 1.

REIT

REIT

(1) REIT

b7							b0
1	1	0	0	1	1	1	1

[Bytes/Cycles]

Bytes/Cycles	1 / 6

RMPA

RMPA

(1) RMPA.size

b7							b0	b7							b0
1	0	1	0	0	1	w1	w0	0	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 11+1.5m*1

Note:

*1. m is the number of operation.

ROLC

ROLC

(1) ROLC.size *dest*

b7		b0	b7							b0	b7								b0
0011	1111	1	1	0	0	0	1	w1	w0	0	0	0	g4	g3	g2	g1	g0		

dest code: dsp:8 / dsp:16 / dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

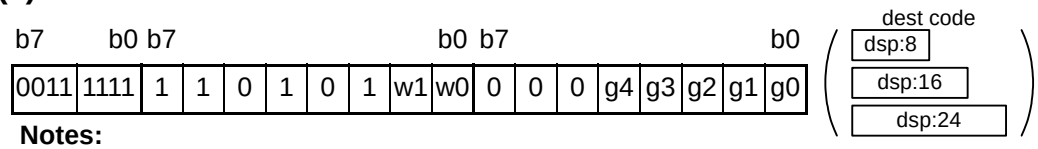
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

RORC

RORC

(1) RORC.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

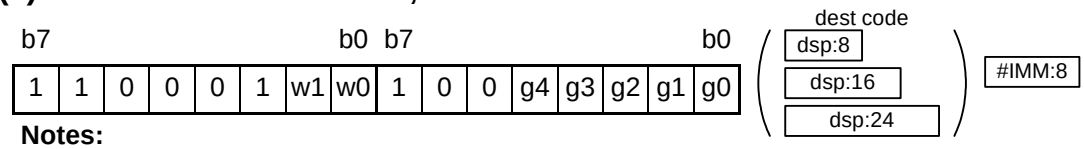
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ROT

ROT

(1) ROT.size #IMM:8,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

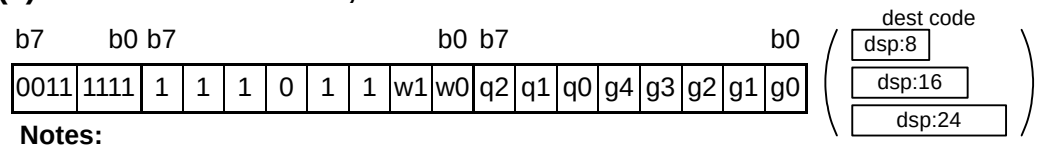
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ROT ROT

(2) ROT.size *src*,*dest*



Notes:

- When src is Bank1 register direct addressing, or when dest is either indirect instruction addressing or Bank1 register direct, the following number is put just before the instruction code:
  01011111, when src is Bank1 register direct addressing
  01101111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when src is Bank1 register direct addressing, and when dest is indirect instruction addressing or Bank1 register direct addressing
- Bits q2 to q0 are specified for the register of src and bits g4 to g0 are for dest.

[Bytes/Cycles]

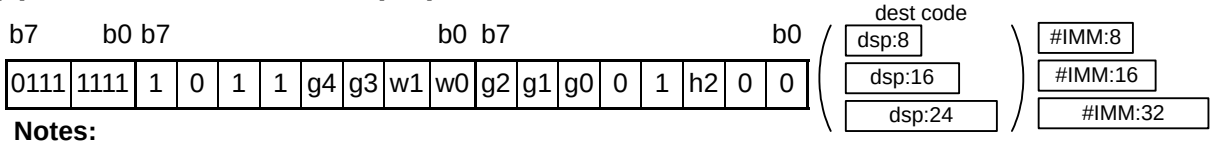
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When src is Bank1 register direct addressing or dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src is Bank1 register direct addressing and dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes increse by 1. When dest is indirect instruciton addressing, the required cycles increases by 2.

ROUND ROUND

(1) ROUND #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
#IMM(EX):16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
#IMM:32	7 / 4	7 / 5	8 / 5	9 / 5	10 / 5	9 / 5	10 / 5

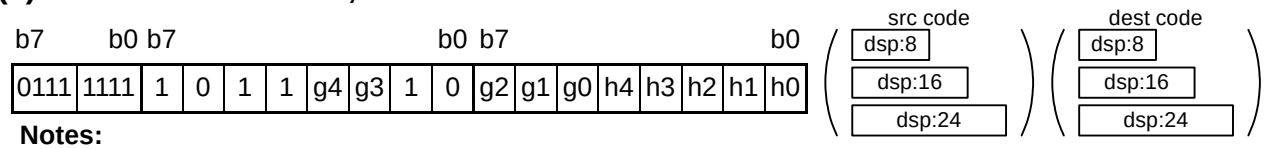
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

ROUND

ROUND

(2) ROUND *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 4	3 / 4	4 / 4	5 / 4	6 / 4	5 / 4	6 / 4
[An]	3 / 5	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
dsp:8[ ]	4 / 5	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
dsp:16[ ]	5 / 5	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24[ ]	6 / 5	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5
dsp:16	5 / 5	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
dsp:24	6 / 5	6 / 5	7 / 5	8 / 5	9 / 5	8 / 5	9 / 5

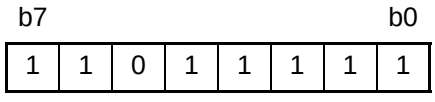
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

RTS

RTS

(1) RTS



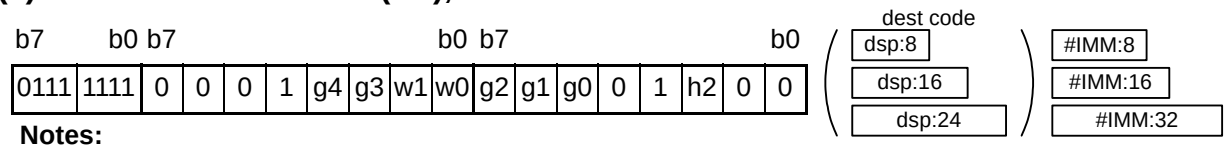
[Bytes/Cycles]

Bytes/Cycles	1 / 5

SBB

SBB

(1) SBB.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

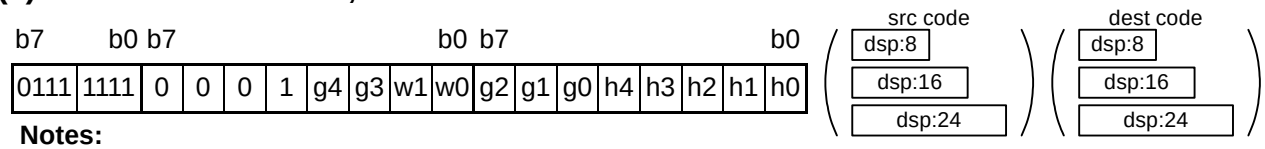
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM(EX):16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
#IMM:32	7 / 1	7 / 2	8 / 2	9 / 2	10 / 2	9 / 2	10 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SBB SBB

(2) SBB.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

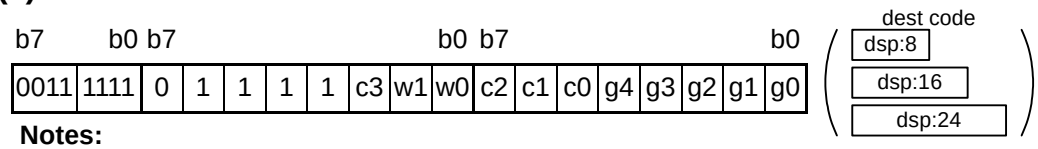
src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
[An]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:8[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:16[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24[ ]	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3
dsp:16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3

Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

SC*Cnd* SC*Cnd*

(1) SC*Cnd*.size *dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits c3 to c0 are specified for *Cnd*. Refer to **4.2.3, "Specifying Conditions"** for details.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SCMPU

SCMPU

(1) SCMPU.size

b7							b0	b7							b0
1	0	0	1	0	1	0	w0	1	0	0	0	1	0	0	0

Note:

- w0 bit is specified for ".B" or ".W". When this bit is set to 0, the size specifier is ".B"; otherwise it is ".W".

[Bytes/Cycles]

Bytes/Cycles	2 / 8+3m*1

Note:

*1. m is the 8-byte word number to be compared.

- If the start address of src and/or dest is not in 8-byte alignment, the required cycles increase by 1.

SHA

SHA

(1) SHA.size #IMM:8,*dest*

b7							b0	b7							b0
1	1	0	1	0	1	w1	w0	0	0	0	g4	g3	g2	g1	g0

dest code: dsp:8 / dsp:16 / dsp:24 — #IMM:8

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

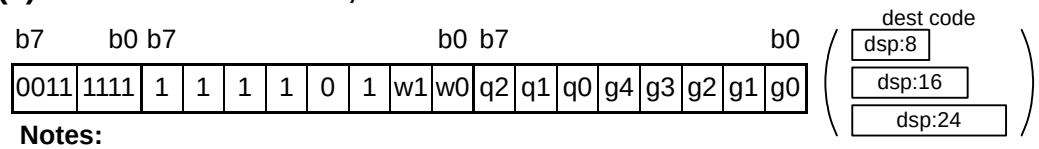
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SHA SHA

(2) SHA.size *src*,*dest*



Notes:

- When src is Bank1 register direct addressing, or when dest is either indirect instruction addressing or Bank1 register direct, the following number is put just before the instruction code:
  01011111, when src is Bank1 register direct addressing
  01101111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when src is Bank1 register direct addressing, and when dest is indirect instruction addressing or Bank1 register direct addressing
- Bits q2 to q0 are specified for the register of src and bits g4 to g0 are for dest.

[Bytes/Cycles]

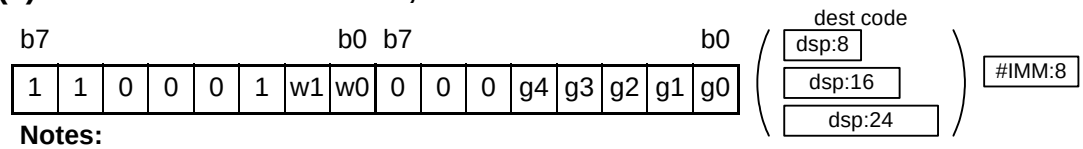
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When src is Bank1 register direct addressing or dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src is Bank1 register direct addressing and dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes increse by 1. When dest is indirect instruciton addressing, the required cycles increases by 2.

SHL SHL

(1) SHL.size:G #IMM:8,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

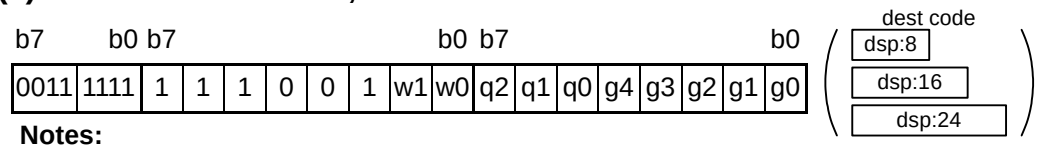
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SHL SHL

(2) SHL.size:G *src*,*dest*



Notes:

- When src is Bank1 register direct addressing, or when dest is either indirect instruction addressing or Bank1 register direct, the following number is put just before the instruction code:
  01011111, when src is Bank1 register direct addressing
  01101111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when src is Bank1 register direct addressing, and when dest is indirect instruction addressing or Bank1 register direct addressing
- Bits q2 to q0 are specified for src (register) and bits g4 to g0 are for dest (operand).

[Bytes/Cycles]

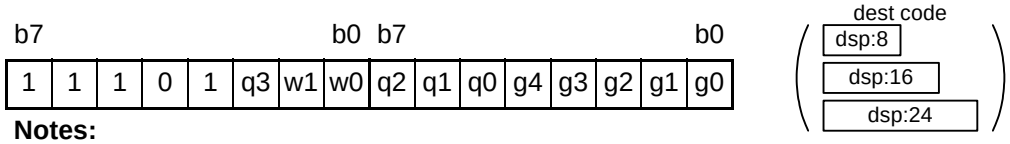
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When src is Bank1 register direct addressing or dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src is Bank1 register direct addressing and dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes increse by 1. When dest is indirect instruciton addressing, the required cycles increases by 2.

SHL SHL

(3) SHL.size:Q #IMM:4,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- 4 bits (q3 to q0) are specified for #IMM:4 and bits g4 to g0 are for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SIN

SIN

(1) SIN.size

b7							b0	b7							b0
1	1	0	0	0	1	w1	w0	0	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 3+2m*1

Note:
*1. m is the number of operation.

SMOVB

SMOVB

(1) SMOVB.size

b7							b0	b7							b0
1	0	1	1	0	1	w1	w0	0	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 3+2m*1

Note:
*1. m is the number of operation.

SMOVF

SMOVF

(1) SMOVF.size

b7							b0	b7							b0
1	1	0	0	0	1	w1	w0	1	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 2+2m*1

Note:
*1. m is the number of operation.

SMOVF

SMOVF

(2) SMOVF.Q

b7							b0	b7							b0
1	1	0	1	0	1	1	0	1	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 4+2m*1

Note:

*1. m is the number of operation.

SMOVU

SMOVU

(1) SMOVU.size

b7							b0	b7							b0
1	1	0	1	0	1	0	w0	1	0	0	0	1	0	0	0

Note:

- w0 bit is specified for ".B" or ".W". When this bit is set to 0, the size specifier is ".B"; otherwise it is ".W".

[Bytes/Cycles]

Bytes/Cycles	2 / 5+3m*1

Notes:

*1. m is the 8-byte word number to be compared.

- If the start address of src and/or dest is not in 8-byte alignment, the required cycles increase by 1.

SOUT

SOUT

(1) SOUT.size

b7							b0	b7							b0
1	0	0	0	0	1	w1	w0	1	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 3+2m*1

Note:

*1. m is the number of operation.

SSTR SSTR

(1) SSTR.size

b7							b0	b7							b0
1	1	0	1	0	1	w1	w0	0	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 5+m*1

Note:

*1. m is the number of operation.

SSTR SSTR

(2) SSTR.Q

b7							b0	b7							b0
1	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 8+m*1

Note:

*1. m is the number of operation.

STC STC

(1) STC *src*,*dest*

b7		b0	b7							b0	b7							b0
0011	1111	1	1	1	1	1	0	1	0	q2	q1	q0	g4	g3	g2	g1	g0	

dest code: dsp:8 / dsp:16 / dsp:24

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- Bits q2 to q0 are specified for the control register of src. Refer to **(5), "Control register direct addressing (CPU)"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

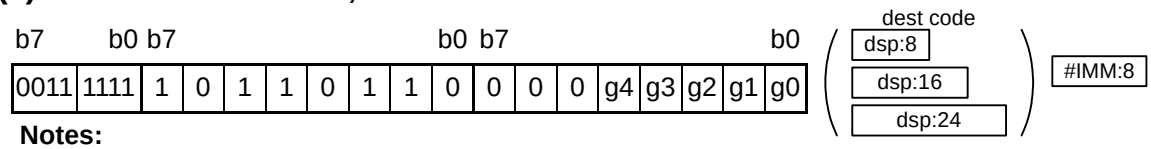
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 2	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

STC STC

(2) STC *src,dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- #IMM:8 is specified for the register of src. Refer to **(6), "Control register direct addressing (DMAC,VCT)"** in **4.2.2, "Specifying the Operand"** for details.

[Bytes/Cycles]

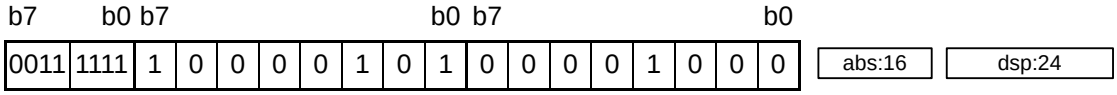
dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 3	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3

Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

STCTX STCTX

(1) STCTX abs:16,dsp:24



[Bytes/Cycles]

Bytes/Cycles	8 / 10+m*1

Note:

*1. m is the number of registers to be transferred.

STNZ

STNZ

(1) STNZ.size #IMM,*dest*

b7 b0	b7 b0	b7 b0															b7 b0
0111	1111	1	0	1	1	0	1	w1	w0	1	0	0	g4	g3	g2	g1	g0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 2	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2

Notes:

- When size specifier (.size) is ".W", the required bytes in the table increases by 1. In the case of ".L ", it increases by 3.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

STOP

STOP

(1) STOP

b7 b0	b7 b0	b7 b0	b7 b0	b7 b0	b7 b0	b7 b0	b7 b0
0011 1111	1010	0100	0000	1000	1100	0010	0001	0000	0000	0110	0000	0000

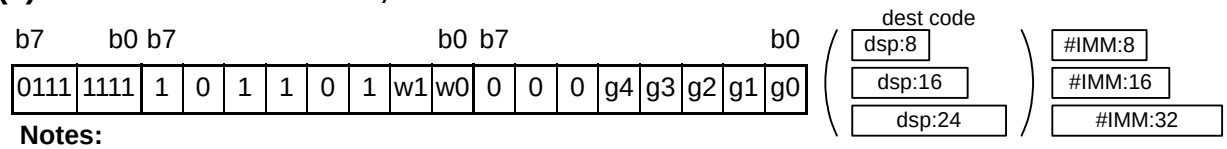
[Bytes/Cycles]

Bytes/Cycles	7 / 8

STZ

STZ

(1) STZ.size #IMM,*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 2	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2

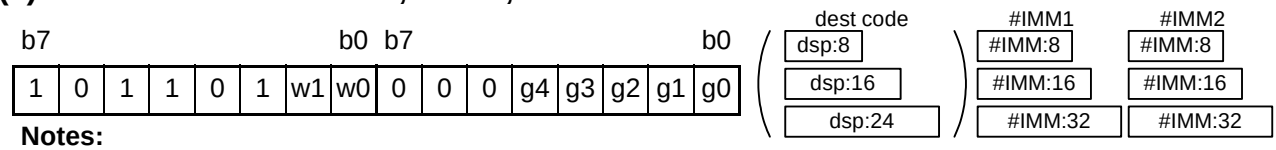
Notes:

- When size specifier (.size) is ".W", 1 is added to the required bytes in the table increases by 1. In the case of ".L", it increases by 3.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

STZX

STZX

(1) STZX.size #IMM1,#IMM2,dest



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01101111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	4 / 4	4 / 4	5 / 4	6 / 4	7 / 4	6 / 4	7 / 4

Notes:

- When size specifier (.size) is ".W", the required bytes in the table increases by 2. In the case of ".L", it increases by 6.
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SUB SUB

(1) SUB.size #IMM(EX),*dest*

b7 b0 b7 b0

1	0	0	1	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0

dest code

dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
#IMM(EX):16	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM:32	6 / 1	6 / 2	7 / 2	8 / 2	9 / 2	8 / 2	9 / 2

Note:

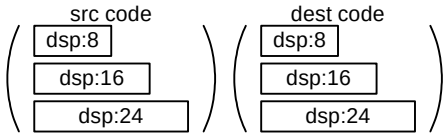
- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SUB

SUB

(2) SUB.size *src*,*dest*

b7							b0	b7							b0
1	0	0	1	g4	g3	w1	w0	g2	g1	g0	h4	h3	h2	h1	h0



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	2 / 1	2 / 2	3 / 2	4 / 2	5 / 2	4 / 2	5 / 2
[An]	2 / 2	2 / 3	3 / 3	4 / 3	5 / 3	4 / 3	5 / 3
dsp:8[ ]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:16[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:16	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:24	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3

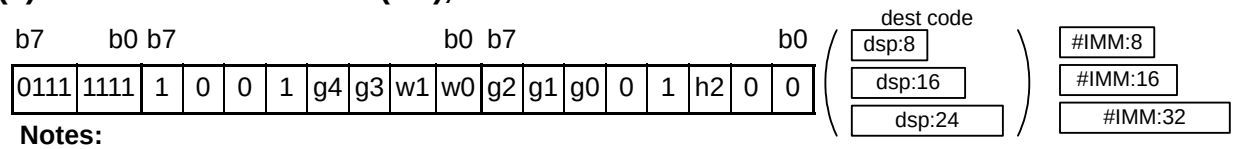
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

SUBF

SUBF

(1) SUBF #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 4	4 / 5	5 / 5	6 / 5	7 / 5	6 / 5	7 / 5
#IMMEX:16	5 / 4	5 / 5	6 / 5	7 / 5	8 / 5	7 / 5	8 / 5
#IMM:32	7 / 4	7 / 5	8 / 5	9 / 5	10 / 5	9 / 5	10 / 5

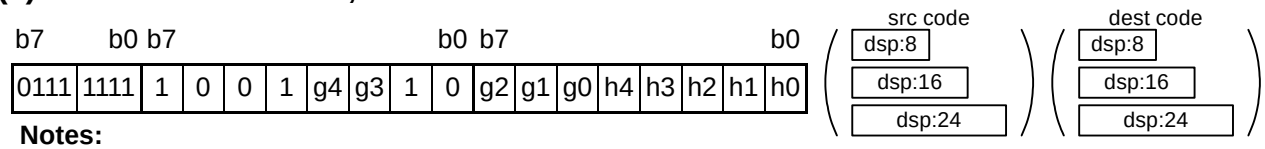
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

SUBF

SUBF

(2) SUBF *src,dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 4	3 / 5	4 / 5	5 / 5	6 / 5	5 / 5	6 / 5
[An]	3 / 5	3 / 6	4 / 6	5 / 6	6 / 6	5 / 6	6 / 6
dsp:8[ ]	4 / 5	4 / 6	5 / 6	6 / 6	7 / 6	6 / 6	7 / 6
dsp:16[ ]	5 / 5	5 / 6	6 / 6	7 / 6	8 / 6	7 / 6	8 / 6
dsp:24[ ]	6 / 5	6 / 6	7 / 6	8 / 6	9 / 6	8 / 6	9 / 6
dsp:16	5 / 5	5 / 6	6 / 6	7 / 6	8 / 6	7 / 6	8 / 6
dsp:24	6 / 5	6 / 6	7 / 6	8 / 6	9 / 6	8 / 6	9 / 6

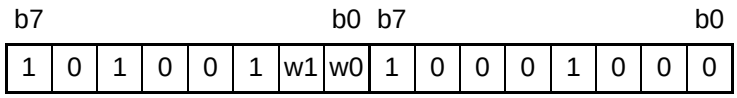
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

SUNTIL

SUNTIL

(1) SUNTIL.size



[Bytes/Cycles]

Bytes/Cycles	2 / 3+3m*1

Note:

*1. m is the number of operation.

SWHILE

SWHILE

(1) SWHILE.size

b7							b0	b7							b0
1	0	1	1	0	1	w1	w0	1	0	0	0	1	0	0	0

[Bytes/Cycles]

Bytes/Cycles	2 / 3+3m*1

Note:

*1. m is the number of operation.

TST

TST

(1) TST.size #IMM(EX),*dest*

b7		b0	b7								b0	b7							b0
0111	1111	0	0	1	0	g4	g3	w1	w0	g2	g1	g0	0	1	h2	0	0		

dest code

dest code	#IMM
dsp:8	#IMM:8
dsp:16	#IMM:16
dsp:24	#IMM:32

Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM(EX):16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
#IMM:32	7 / 1	7 / 2	8 / 2	9 / 2	10 / 2	9 / 2	10 / 2

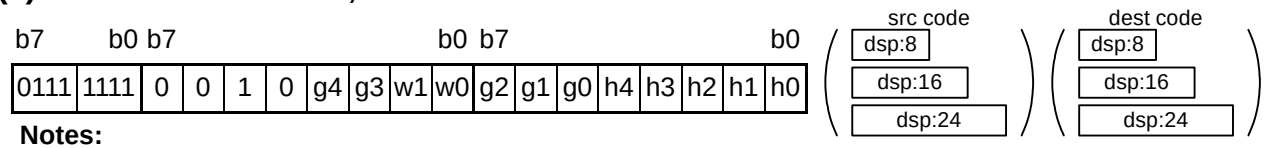
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

TST

TST

(2) TST.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
[An]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:8[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:16[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24[ ]	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3
dsp:16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3

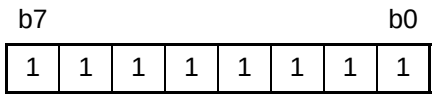
Note:

- When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

UND

UND

(1) UND



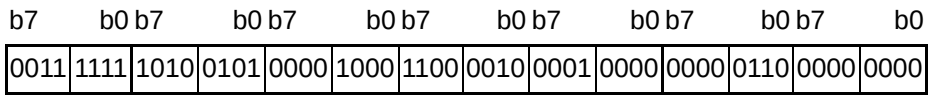
[Bytes/Cycles]

Bytes/Cycles	1 / 12

WAIT

WAIT

(1) WAIT



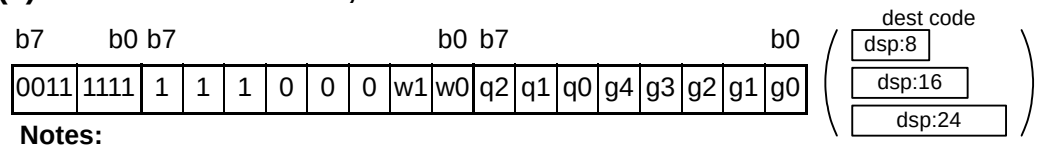
[Bytes/Cycles]

Bytes/Cycles	7 / 8

XCHG

XCHG

(1) XCHG.size *src*,*dest*



Notes:

- When src is Bank1 register direct addressing, or when dest is either indirect instruction addressing or Bank1 register direct, the following number is put just before the instruction code:
  01011111, when src is Bank1 register direct addressing
  01101111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when src is Bank1 register direct addressing, and when dest is indirect instruction addressing or Bank1 register direct addressing
- Bits q2 to q0 are specified for the register of src and bits g4 to g0 are for dest.

[Bytes/Cycles]

dest	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Bytes/Cycles	3 / 3	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3

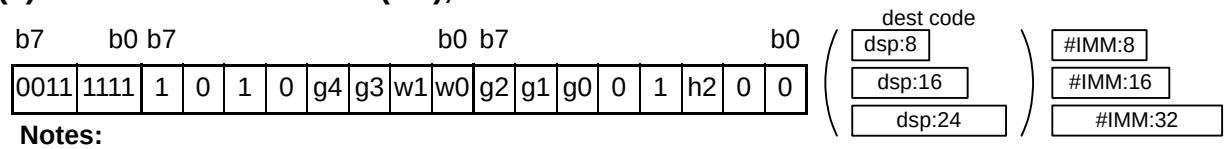
Note:

- When src is Bank1 register direct addressing or dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src is Bank1 register direct addressing and dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes increse by 1. When dest is indirect instruciton addressing, the required cycles increases by 2.

XOR

XOR

(1) XOR.size #IMM(EX),*dest*



Notes:

- When dest is either indirect instruction addressing or Bank1 register direct, 01011111 is put just before the instruction code.
- Bits g4 to g0 are specified for dest.
- h2 bit is specified for src. When this bit is set to 0, the operand is #IMMEX; otherwise it is #IMM.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
#IMM(EX):8	4 / 1	4 / 2	5 / 2	6 / 2	7 / 2	6 / 2	7 / 2
#IMM(EX):16	5 / 1	5 / 2	6 / 2	7 / 2	8 / 2	7 / 2	8 / 2
#IMM:32	7 / 1	7 / 2	8 / 2	9 / 2	10 / 2	9 / 2	10 / 2

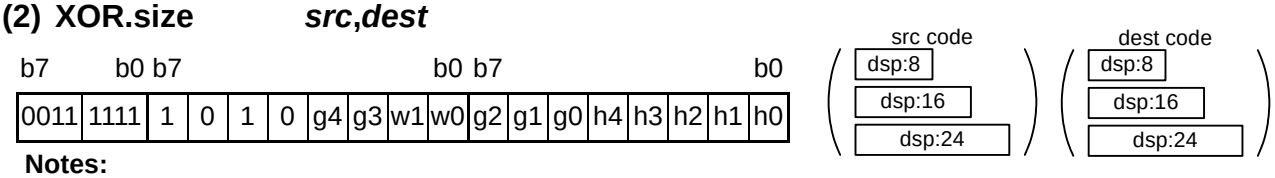
Note:

- When dest is either indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When it is indirect instruction addressing, the required cycles increases by 2.

XOR

XOR

(2) XOR.size *src*,*dest*



Notes:

- In indirect instruction addressing or Bank1 register direct addressing, the followinwg number is put just before the instruction code:
  01101111, when src is indirect instruction addressing or Bank1 register direct addressing
  01011111, when dest is indirect instruction addressing or Bank1 register direct addressing
  01001111, when both src and dest are indirect instruction addressing or Bank1 register direct addressing
- Bits g4 to g0 are spcified for dest and bits h4 to h0 are for src.

[Bytes/Cycles]

src \ *dest*	Register	[An]	dsp:8[ ]	dsp:16[ ]	dsp:24[ ]	dsp:16	dsp:24
Register	3 / 1	3 / 2	4 / 2	5 / 2	6 / 2	5 / 2	6 / 2
[An]	3 / 2	3 / 3	4 / 3	5 / 3	6 / 3	5 / 3	6 / 3
dsp:8[ ]	4 / 2	4 / 3	5 / 3	6 / 3	7 / 3	6 / 3	7 / 3
dsp:16[ ]	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24[ ]	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3
dsp:16	5 / 2	5 / 3	6 / 3	7 / 3	8 / 3	7 / 3	8 / 3
dsp:24	6 / 2	6 / 3	7 / 3	8 / 3	9 / 3	8 / 3	9 / 3

Note:

When either src and/or dest are indirect instruction addressing or Bank1 register direct addressing, the required bytes in the table increases by 1. When src or dest is indirect instruction addressing, the required cycles increases by 2. When both src and dest are indirect instruction addressing, the required cycles increases by 4.

5. Interrupts

5.1 Overview

When an interrupt request is accepted, the instruction branches to the interrupt handler set in the interrupt vector table. The start address of the interrupt handler should be set in the interrupt vector table. Refer to **1.7, "Interrupt Vector Table"** for details.

5.1.1 Interrupt Types

Figure 5.1 shows types of interrupt. Table 5.1 lists interrupt sources (non-maskable) and vector table.

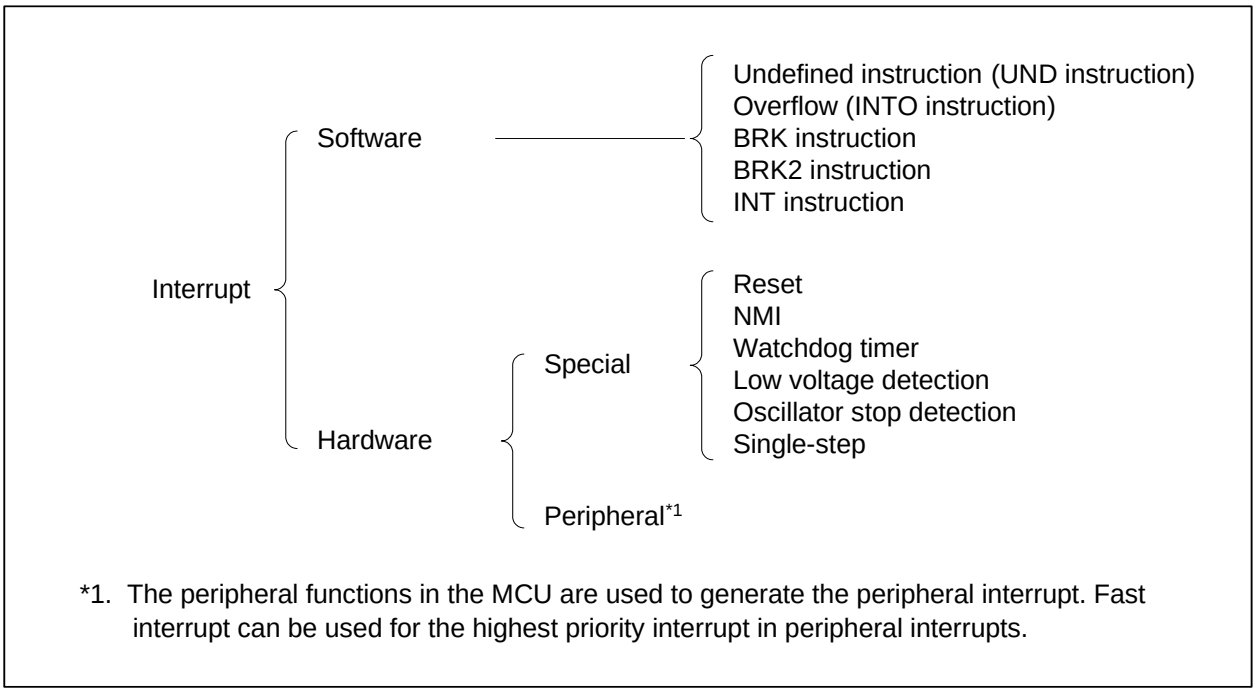


Figure 5.1 Interrupt Types

Table 5.1 Interrupt Sources (non-maskable) and Vector Table

Interrupt source	Vector table addresses address (L) to address (H)	Remarks
Undefined instruction	FFFFFFDCh to FFFFFFDFh	Interrupt generated by UND instruction
Overflow	FFFFFFE0h to FFFFFFE3h	Interrupt generated by INTO instruction
BRK instruction	FFFFFFE4h to FFFFFFE7h	Executed beginning from the address indicated by a vector in relocatable vector table if the bit of address FFFFFFE7h is FFh.
Watchdog timer Low voltage detection Oscillator stop detection	FFFFFFF0h to FFFFFFF3h	
NMI	FFFFFFF8h to FFFFFFFBh	External interrupt generated by driving the $\overline{\text{NMI}}$ pin low.
Reset	FFFFFFFCh to FFFFFFFFh	

The interrupts are also classified into maskable/non-maskable.

(1) Maskable interrupt

Maskable interrupts can be disabled by the interrupt enable flag (I flag).
The priority is configurable by assigning an interrupt request level.

(2) Non-maskable interrupt

Non-maskable interrupts cannot be disabled by the interrupt enable flag (I flag).
The priority is not configurable.

5.1.2 Software Interrupts

Software interrupts are non-maskable. A software interrupt is generated by executing an instruction.
There are five types of software interrupts as follows:

(1) Undefined instruction interrupt

This interrupt is generated when the UND instruction is executed.

(2) Overflow interrupt

This interrupt is generated when the INTO instruction is executed while the O flag is 1. The following instructions may change the O flag to 1, depending on the operation results:

ABS, ADC, ADCF, ADD, ADDF, ADSF, CMP, CMPF, CNVIF, DIV, DIVF, DIVU, DIVX, EDIV, EDIVU, EDIVX, MUL, MULF, MULU, MULX, NEG, RMPA, ROUND, SBB, SCMPU, SHA, SUB, SUBF, SUNTIL and SWHILE

(3) BRK instruction interrupt

This interrupt is generated when the BRK instruction is executed.

(4) BRK2 instruction interrupt

This interrupt is generated when the BRK2 instruction is executed.
This interrupt is used only for development support tool, and users are not allowed to used it.

(5) INT instruction interrupt

This interrupt is generated when the INT instruction is executed with a selected software interrupt number from 0 to 255. Numbers 0 to 127 are designated for peripheral interrupts. That is, the INT instruction with a number from 0 to 127 has the same interrupt handler as that for the peripheral interrupt.
The stack pointer (SP), which contains two types, is specified by the stack pointer select flag (U flag). For numbers 0 to 127, when an interrupt request is accepted, the U flag is saved to select the interrupt stack pointer (ISP) before the interrupt sequence is executed. The saved data of the U flag is restored upon returning from the interrupt handler. For numbers 128 to 255, the stack pointer used before the interrupt request acceptance remains unchanged for the interrupt sequence.

5.1.3 Hardware Interrupts

There are two kinds of hardware interrupts: special interrupt and peripheral interrupt.
In the peripheral interrupt, only a single interrupt with the highest priority can be specified as fast interrupt.

(1) Special interrupts

Special interrupts are non-maskable. There are six interrupts as follows:

(a) Reset

This interrupt occurs when the $\overline{\text{RESET}}$ pin is driven low.

(b) NMI

This interrupt occurs when the $\overline{\text{NMI}}$ pin is driven low.

(c) Watchdog timer interrupt

This interrupt is generated by the first time-out of the watchdog timer.

(d) Low voltage detection interrupt

This interrupt is generated by the low voltage detection circuit.

(e) Oscillator stop detection interrupt

This interrupt is generated by the oscillator stop detection circuit.

(f) Single-step interrupt

This interrupt is used only for development support tool. Users are not allowed to used it. This interrupt is generated after an instruction is executed while the debug flag (D flag) is set to 1.

(2) Peripheral interrupt

This interrupt is generated by internal peripheral functions. The functions and interrupt source types vary with each MCU model. It shares the interrupt vector table with software interrupt numbers 0 to 127 for the INT instruction. This interrupt is maskable. Refer to **"Hardware manual"** for details.
For this interrupt, when an interrupt request is accepted, the stack pointer select flag (U flag) is saved to select the interrupt stack pointer (ISP) before the interrupt sequence is executed. The saved data of the U flag is restored upon returning from the interrupt handler.

(3) Fast interrupt

This interrupt enables the CPU to execute the interrupt response at high speeds. Only a single peripheral interrupt with the highest priority can be designated as the fast interrupt. Execute the FREIT instruction to return from the fast interrupt handler. Refer to **"Hardware manual"** for details.

5.2 Interrupt Control

This section describes how to enable/disable maskable interrupts and how to set the acceptable interrupt request level. The explanation here does not apply to non-maskable interrupts.
The maskable interrupts may be enabled/disabled using the interrupt enable flag (I flag), the interrupt request level select bit and the processor interrupt priority level (IPL). Each interrupt has its interrupt control register with the interrupt request level select bit and the interrupt request bit. The presence/absence of an interrupt request is indicated by the interrupt request bit. The interrupt enable flag (I flag) and the processor interrupt priority level (IPL) are allocated in the flag register (FLG). Refer to **"Hardware manual"** for details on the memory allocation of the interrupt control registers and the register organization.

5.2.1 Interrupt Enable Flag (I Flag)

This flag enables/disables maskable interrupts. When the I flag is set to 1, all maskable interrupts are enabled; when it is set to 0, they are disabled. The I flag automatically becomes 0 after a reset operation. When changing the flag, the change becomes effective from the following timings:

- Changed in the REIT or FREIT instruction, the change becomes effective during the execution of the instruction.
- Changed in the FCLR, FSET, POPC or LDC instruction, the change becomes effective from the next instruction.

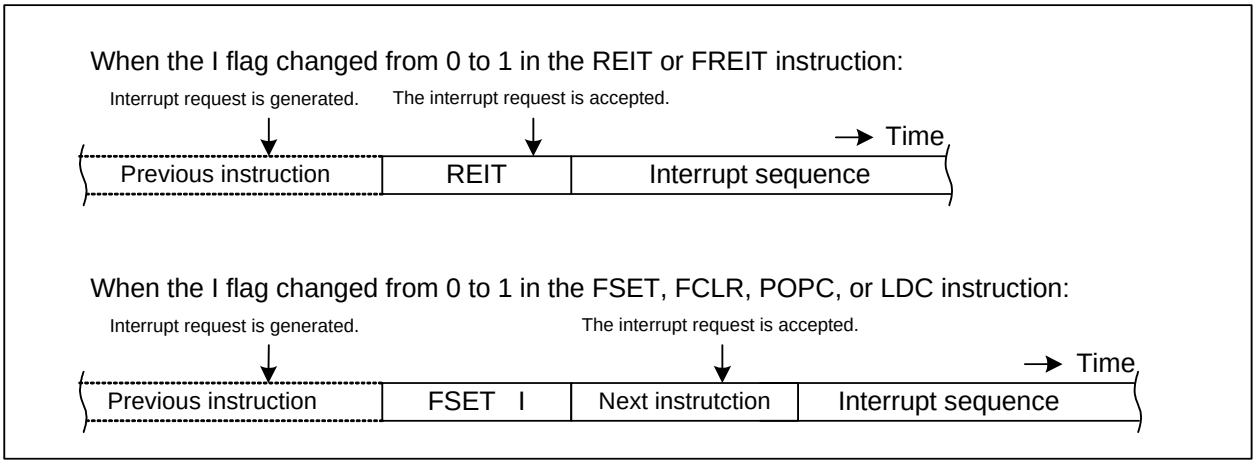


Figure 5.2 The Timing of Interrupt Request Acceptance Affected by the I Flag Change

5.2.2 Interrupt Request Bit

Interrupt request bit is changed to 1 when an interrupt request is generated. It remains 1 until the interrupt request is accepted in which the bit is set to 0.
This bit can be set to 0 by software. (Do not set it to 1).

5.2.3 Interrupt Request Level and Processor Interrupt Priority Level (IPL)

Interrupt request level is set by the interrupt request level select bit in an interrupt control register. When an interrupt request is generated, its interrupt request level is compared with the processor interrupt priority level (IPL). This interrupt is accepted only when its interrupt request level is higher than the processor interrupt priority level (IPL). This means an interrupt whose interrupt priority level is 0 is disabled.
Table 5.2 lists interrupt priority level settings. Table 5.3 lists interrupt enable levels according to the processor interrupt priority level (IPL).

Table 5.2 Interrupt Request Levels

Interrupt Request Level Select Bit			Interrupt Request Level	Priority Order
b2	b1	b0		
1	1	1	Level 7	High
1	1	0	Level 6	
1	0	1	Level 5	
1	0	0	Level 4	
0	1	1	Level 3	
0	1	0	Level 2	
0	0	1	Level 1	Low
0	0	0	Level 0 (Interrupt disabled)	—

Table 5.3 IPL and Interrupt Enable Levels

Processor Interrupt Priority Level (IPL)			Acceptable Interrupt Request Levels
IPL2	IPL1	IPL0	
1	1	1	All maskable interrupts are disabled.
1	1	0	Level 7 only
1	0	1	Level 6 and above
1	0	0	Level 5 and above
0	1	1	Level 4 and above
0	1	0	Level 3 and above
0	0	1	Level 2 and above
0	0	0	Level 1 and above

The following lists the conditions under which an interrupt request is accepted:

- Interrupt enable flag (I flag) = 1
- Interrupt request bit = 1
- Interrupt request level > Processor interrupt priority level (IPL)

The interrupt enable flag (I flag), the interrupt request bit, the interrupt request level select bit, and the processor interrupt priority level (IPL) are all independent of each other, so they do not affect any other bit.

When changing the processor interrupt priority level (IPL) or the interrupt request level, the change becomes effective from the following timings:

- Changed in the REIT or FREIT instruction, the new processor interrupt priority level (IPL) becomes effective during the execution of the instruction.
- Changed in the POPC, LDC, or LDIPL instruction, the new processor interrupt priority level (IPL) becomes effective from the next instruction.
- Changed in an instruction such as MOV, the new interrupt request level becomes effective after one peripheral bus clock cycle.

5.2.4 Rewrite the Interrupt Control Register

When rewriting the interrupt control register, disable all maskable interrupts before the rewrite to ensure that no corresponding interrupt request will be generated.
When enabling the maskable interrupts immediately after the rewrite, depending on the instruction queue status the interrupt enable flag (I flag) may be set to 1 before the rewrite completes. To prevent this, a measure such as NOP insertion may be necessary.
While interrupts are disabled, if an interrupt is generated for the register being rewritten, the interrupt request bit may not be set to 1. Use the following instructions to rewrite the register, if necessary:

- AND
- OR
- BCLR
- BSET

5.3 Interrupt Sequence

The interrupt sequence is performed from when an interrupt request has been accepted until the interrupt handler starts.
For most instructions, when an interrupt request is generated while an instruction is being executed, the requested interrupt is evaluated in the priority resolver after the current instruction is completed. If appropriate, the interrupt sequence starts from the next cycle.
For instructions RMPA, SCMPU, SIN, SMOVB, SMOVF, SMOVU, SOUT, SSTR, SUNTIL, and SWHILE, as soon as an interrupt request is generated, the requested interrupt is evaluated suspending the current instruction being executed. If appropriate, the interrupt sequence starts immediately.
The interrupt sequence is as follows:

(1) The CPU acknowledges the interrupt request to obtain the interrupt information (the interrupt number, and the interrupt request level) from the interrupt controller. Then the corresponding IR bit becomes 0 (no interrupt requested)
(2) The state of the flag register (FLG) before the interrupt sequence is stored to a temporary register *1 in the CPU.
(3) The following bits in the flag register (FLG) become 0:
- The I flag (interrupt enable flag): interrupt disabled
- The D flag (debug flag): single-step interrupt disabled
- The U flag (Stack pointer select flag): ISP selected (Note that the U flag does not change for INT instruction whose software interrupt number is from 128 to 255)

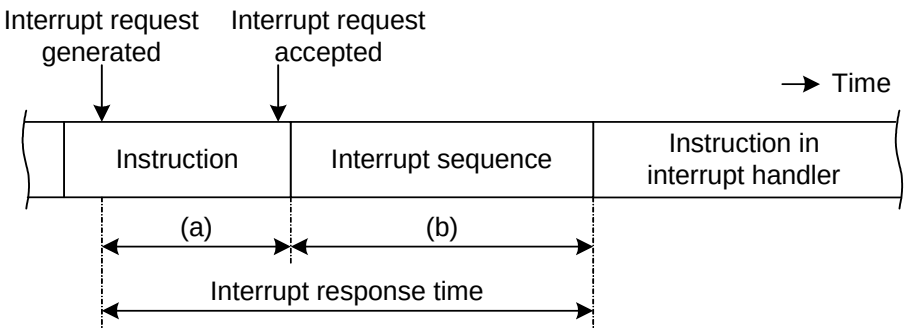
(4) The content of the temporary register *1 in the CPU is saved to the stack area; or to the save flag register (SVF) in case of the fast interrupt.
(5) The content of the program counter (PC) is saved to the stack area; or to the save PC register (SVP) in case of the fast interrupt.
(6) The interrupt request level of the accepted interrupt is set in the processor interrupt priority level (IPL).
(7) The corresponding interrupt vector is read from the interrupt vector table.
(8) This interrupt vector is stored into the program counter (PC).

When the interrupt sequence completes, the interrupt handler is initiated.

Note:
*1. This register is inaccessible to users.

5.3.1 Interrupt Response Time

The interrupt response time, as shown in Figure 5.3 consists of two non-overlapping time segments: (a) the period from when an interrupt request is generated until the instruction being executed is completed; and (b) the period required for the interrupt sequence.



(a) Period from when an interrupt request is generated until when the instruction being executed is completed
(b) Period required to perform the interrupt sequence

Figure 5.3 Interrupt Response Time

Time (a) varies depending on the instruction being executed.
Time (b) is listed in Table 5.4.

Table 5.4 Interrupt Sequence Execution Time

Interrupt	Interrupt Vector Address	Interrupt Sequence Execution Time (internal memory)
Peripherals	Aligned to a 4-byte boundary *1	13 cycles *2
INT instruction	Aligned to a 4-byte boundary *1	11 cycles
BRK instruction (relocatable vector table)	Aligned to a 4-byte boundary *1	16 cycles
Undefined instruction	FFFFFFDCh *3	12 cycles
Overflow	FFFFFFE0h *3	12 cycles
BRK instruction (fixed vector table)	FFFFFFE4h *3	19 cycles
Watchdog timer Low voltage detection Oscillator stop detection	FFFFFFF0h *3	11 cycles
NMI	FFFFFFF8h *3	10 cycles
Single-step BRK2 instruction DBC interrupt	Aligned to a 4-byte boundary *3	19 cycles
Fast interrupt *4	Vector table is internal register	11 cycles

Notes:
*1. The interrupt vectors should be aligned to a 4-byte boundary to shorten the execution time.
*2. If a peripheral bus has three or more waits, the required cycles increases by (waits -2).
*3. Addresses are 4-byte aligned.
*4. The fast interrupt is independent of these conditions.

5.3.2 IPL After Interrupt Request Acceptance

When an interrupt request is accepted, the interrupt request level is set in the processor interrupt priority level (IPL).
If an interrupt request has no interrupt priority level, the value shown in Table 5.5 is set in the IPL.

Table 5.5 Interrupts without Interrupt Priority Level and IPL

Interrupt Sources Without Interrupt Priority Level	IPL Value to be Set
Watchdog timer, NMI, Low voltage detection, Oscillator stop detection	7
Reset	0
Others	unchanged

5.3.3 Register Saving

In the interrupt sequence, the flag register (FLG) and the program counter (PC) values are saved to the stack, in that order. Figure 5.4 shows the stack status before and after an interrupt request is accepted.
In the fast interrupt sequence, the flag register (FLG) and the program counter (PC) values are saved to the save flag register (SVF) and to the save PC register (SVP), respectively.
If there are any other registers to be saved to the stack, save them at the beginning of the interrupt handler. A single PUSHM instruction saves all registers except the stack pointer (SP).

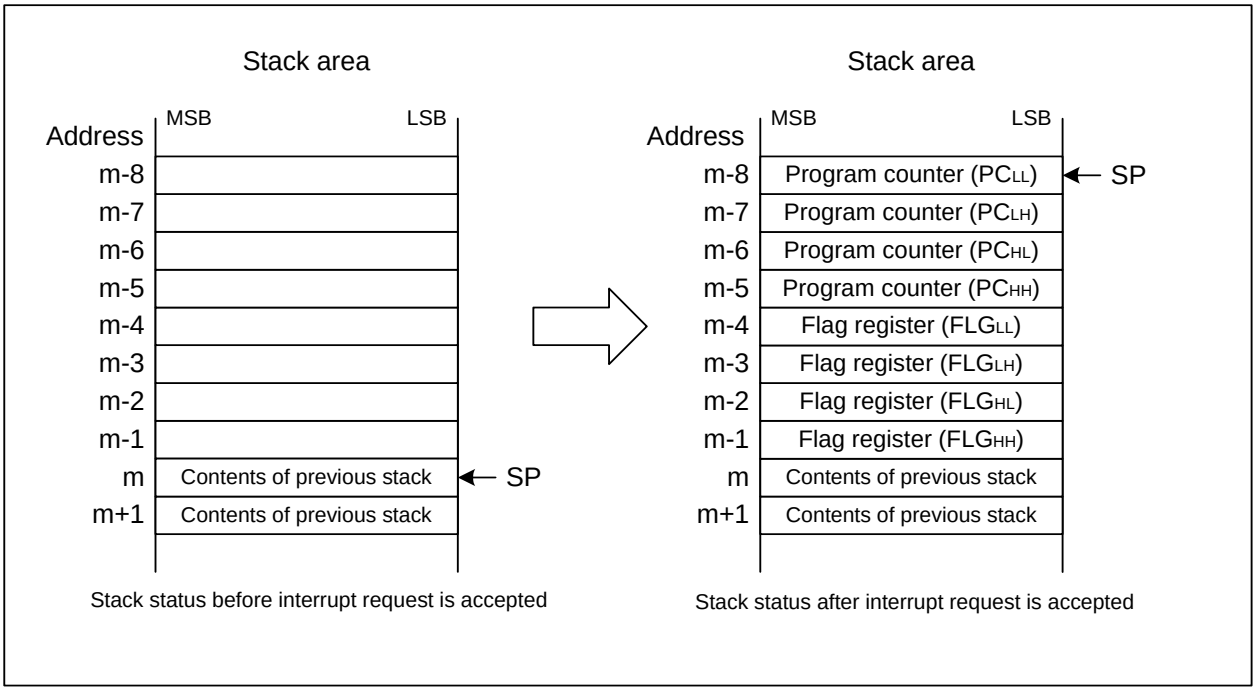


Figure 5.4 Stack Status Before and After an Interrupt Request is Accepted

5.4 Register Restoring

When the REIT instruction is executed at the end of the interrupt handler, the saved values of the flag register (FLG) and the program counter (PC) are restored from the stack area, and the program resumes the operation that has been interrupted. In the fast interrupt, execute the FREIT instruction to restore them from the save registers, instead.
To restore the values of registers, which are saved by software in the interrupt handler, use an instruction such as POPM before the REIT or FREIT instruction.

5.5 Interrupt Priority

If two or more interrupt requests are detected at an interrupt request sampling point, the interrupt request with higher priority is accepted.
For maskable interrupts (peripheral interrupts), the interrupt request level select bits (bits ILVL2 to ILVL0) select a request level. If there are more than two interrupts with the same level, they are accepted according to their relative priority predetermined by the hardware *1.
The priorities of non-maskable interrupts such as the reset (reset has the highest priority) and watchdog timer interrupt are determined by the hardware. The following is the priority order of non-maskable interrupts:

Reset > Watchdog timer / Low voltage detection / Oscillation stop detection > NMI > Peripherals > Single-step

Software interrupts are not governed by priority. They always cause execution to jump to the interrupt handler whenever the relevant instruction is executed.

Note:
*1. The priority order varies with the MCU model. Refer to **"Hardware Manual"** for details.

5.6 Multiple Interrupts

The following shows the internal bit states when control has branched to an interrupt handler:

- The interrupt enable flag (I flag) is 0 (disable interrupts).
- The interrupt request bit for the accepted interrupt is 0.
- The processor interrupt priority level (IPL) equals the interrupt request level of the accepted interrupt.

By setting the interrupt enable flag (I flag) to 1 in the interrupt handler, an interrupt request that has higher priority than the processor interrupt priority level (IPL) can be accepted. Figure 5.5 shows multiple interrupts proceeding.
The interrupt requests that have not been accepted due to their low interrupt priority level are kept pending. When the IPL is restored by an REIT or an FREIT instruction, the pending interrupt requests are accepted if the following condition is satisfied:

Interrupt request level of pending interrupt request > Restored processor interrupt prioroty level (IPL)

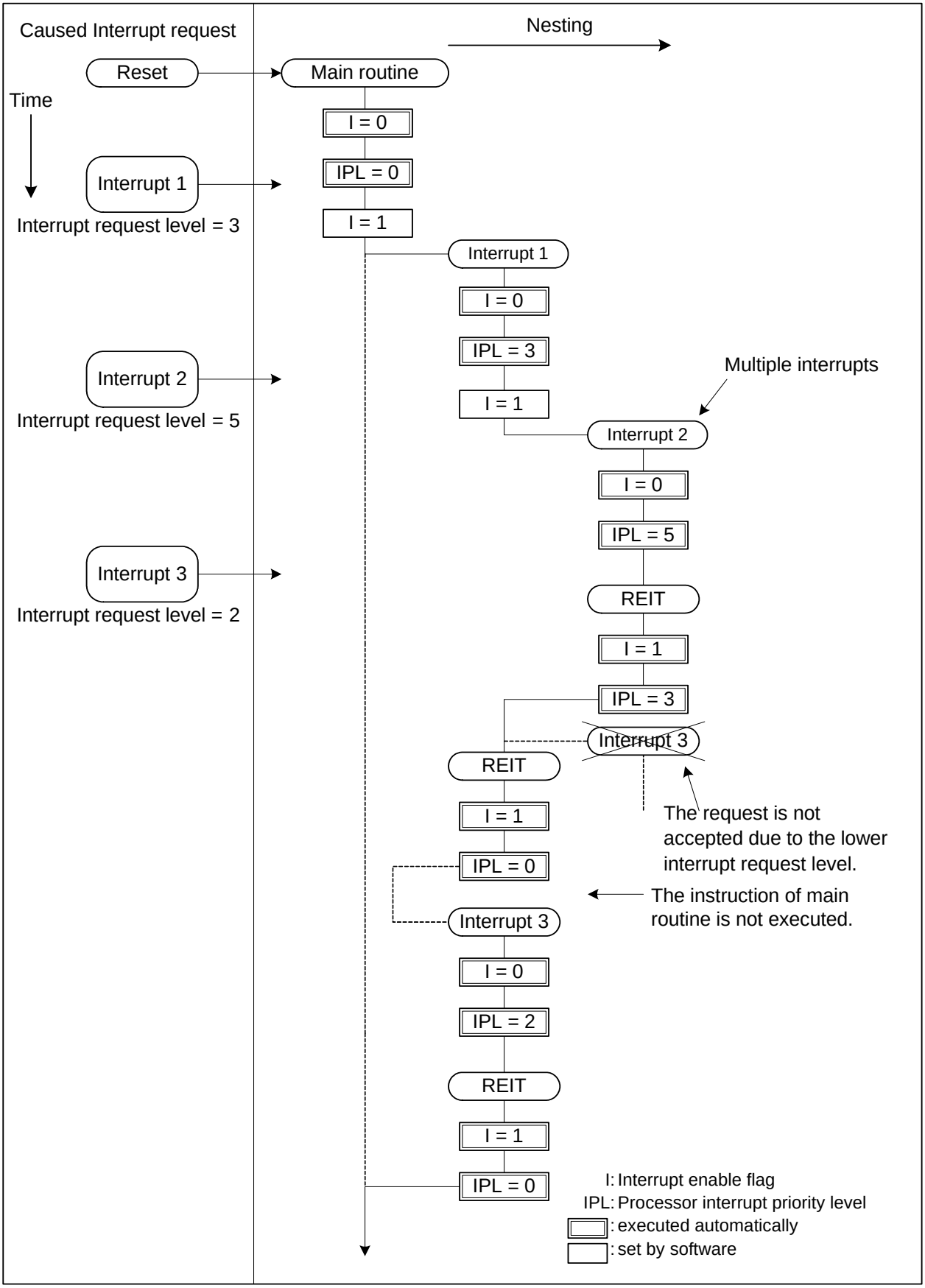


Figure 5.5 Multiple Interrupts

5.7 Notes on Interrupts

Notes on interrupts are listed as follows:

(1) ISP Setting

The interrupt stack pointer (ISP) is initialized to 00000000h after a reset operation. Set a value to the ISP before an interrupt is accepted to avoid undefined behavior.

Especially when using the NMI interrupt, set a value to the ISP at the beginning of a program. The NMI interrupt cannot be disabled if the PM24 bit in the PM2 register is set to 1.

A multiple of 4 should be set to the ISP to shorten the time required for the interrupt sequence.

(2) Rewrite the Interrupt Control Register

When rewriting the interrupt control register, disable all maskable interrupts before the rewrite to ensure that no corresponding interrupt request will be generated.

When enabling the maskable interrupts immediately after the rewrite, depending on the instruction queue status the interrupt enable flag (I flag) may be set to 1 before the rewrite completes. To prevent this, a measure such as NOP insertion may be necessary.

While interrupts are disabled, if an interrupt is generated for the register being rewritten, the interrupt request bit may not be set to 1. Use the following instructions to rewrite the register, if necessary:

- AND
- OR
- BCLR
- BSET

(3) Exit from Stop Mode and Wait Mode

When using a peripheral interrupt to exit stop mode or wait mode, the relevant interrupt must have been enabled and assigned a interrupt request level higher than 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 the flag register (FLG).

Reset and NMI interrupts are independent of the interrupt priority set bits for exiting a stop/wait state to exit stop mode or wait mode.

INDEX

L

M

N

O

P

Q

R

S

U

V

W

Z

REVISION HISTORY	R32C/100 Series Software Manual

Rev.	Date	Description	
		Page	Summary
1.00	Apr 08, 2009	—	First edition published

RENESAS MCU
SOFTWARE MANUAL
R32C/100 SERIES

Publication Date: Apr 08, 2009 Rev.1.00

Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
2-6-2, Otemachi, Chiyoda-ku, Tokyo 100-0004

R32C/100 SERIES
SOFTWARE MANUAL

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ09B0267-0100